

Accelerating open-source biomolecular models with Claude

Claude Science^{1,*}, Richard Shuai^{1,*}, Rohil Badkundri¹, Vincent Fan¹, Kilian Fatras², Lukas Jarosch³, and Amir Shanehsazzadeh^{1,†}

¹Anthropic, San Francisco, CA, USA ²Biohub ³Columbia University, New York, NY, USA

*These authors contributed equally.

†Correspondence: ashanehsazzadeh@anthropic.com

September 17, 2026

Last updated September 21, 2026

Abstract

Open-source deep-learning models for biomolecular structure prediction, protein design, genomics, and protein language modeling are widely used. Running them at scale is expensive, and modeling the largest molecular machines requires more GPU memory than a single node provides. We asked Claude to optimize inference for these models. Supervised by two scientists experienced in biomolecular modeling but not in inference optimization or kernel engineering, Claude produced optimized packages for 36 model implementations, covering more than 30 open-source models, in just under four weeks. Each package offers up to three modes: Exact, which is designed to reproduce the unmodified model’s outputs bit for bit; Fast, which trades a small amount of numerical precision for speed; and Big, which reduces memory use so that larger inputs fit in GPU memory. Exact mode ran the forward pass of 14 structure prediction models on average 1.6× faster than the unmodified models on NVIDIA H100 GPUs, and Fast mode ran the forward pass of the 13 models that have one on average 4.1× faster. The share of acceptably predicted interfaces, pooled over 13 model configurations (1,925 model–target pairs), changed by less than one percentage point in every mode, and no pooled change was distinguishable from zero. Claude also developed FlashPairformer v1, a set of GPU kernels whose triangle attention is on average 2.7× faster than the field-standard kernels at the pair width used by most of these models. Big mode enabled accurate predictions of complexes larger than 10,000 tokens (amino acids, nucleotides, and atoms from small molecules and ions) on a single 8-GPU node, and it ran inference on protein assemblies of up to 70,320 residues on a node of eight B300 GPUs, although these predictions were not accurate. Finally, Claude designed protein binders with *in silico* scores comparable to those of earlier campaigns that could use about 100 times as many GPU hours. It worked from a simplified prompt, with the optimized models on a single H200 GPU for 24 hours. We release the optimized code for all models.

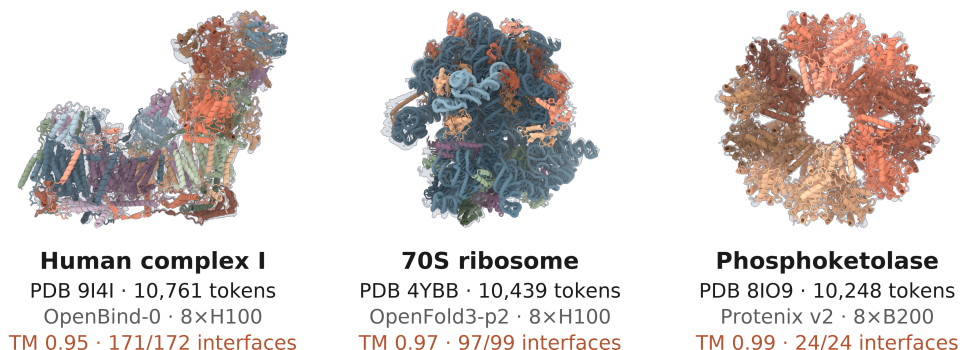


Figure 1. Structures of large molecular machines predicted on a single GPU node with the memory-efficient Big mode. Left to right: human mitochondrial complex I (PDB 9I4I), the *Escherichia coli* 70S ribosome (PDB 4YBB), and phosphoketolase (PDB 8IO9). Predictions, colored by subunit, are superposed on the experimental structures (gray). Labels give the number of tokens, the model and GPUs (one node of eight), the TM-score, and the number of acceptably predicted interfaces (DockQ $\geq$ 0.23). These are three of the predictions in Figure 7.

Introduction

Deep-learning models now predict the three-dimensional structures of proteins and their complexes with other biomolecules [1, 2], generate new protein backbones and sequences [3, 4], and read regulatory information from DNA sequences [5, 6]. Many of these models are open source, and together they have become everyday tools for molecular biologists, including in drug discovery and development. However, running these models at campaign scale demands more compute than most researchers can access. For example, Claude recently orchestrated open-source design and structure prediction models to design protein binders autonomously [7], in single-target campaigns that could spend up to \$10,000 of cloud GPU time per target over 24 hours. That budget corresponds to roughly 2,500 NVIDIA H100 GPU hours at list prices.

We began to explore inference optimizations that let these models run more efficiently, to make this kind of research more accessible. In an early result, Claude Mythos 5.1 accelerated seven open-source deep-learning models for biology by up to 2.5 times, with identical outputs, using custom GPU kernels and cached intermediate results [8]. Here we report a much broader effort, in which an internal general-purpose research model optimized more than 30 open-source models (36 implementations) spanning structure prediction, protein design, genomics, and protein language modeling (Supplementary Information, SI). For each model, Claude produced an optimized package that runs with the original weights and has up to three modes, called Exact, Fast, and Big (described below).

Fast mode ran the forward pass of the structure prediction models on average $4.1\times$ faster than the fastest correct configuration we found for each unmodified model (13 models), and Exact mode $1.6\times$ faster (14 models; Figure 3). Neither mode measurably changed accuracy on a pooled benchmark of biomolecular interfaces (Figure 6). Claude also developed FlashPairformer v1, GPU kernels for the triangle operations that account for much of the cost of these models (Figure 2). Big mode enabled accurate prediction of molecular machines larger than 10,000 tokens on a single 8-GPU node (Figures 1 and 7). It also ran inference on entire viral capsids and protein compartments of up to 70,320 residues on eight B300 GPUs, although these predictions were not accurate (Figure 8). A single Claude model with one H200 GPU for 24 hours designed binders whose *in silico* scores were comparable to those of single-target campaigns that Claude Mythos 5.1 ran with a GPU budget about two orders of magnitude larger (Figure 9). It used the optimized models and a simplified version of our earlier agentic design protocol. We are releasing all of the optimized code.

Results

FlashPairformer: faster kernels for triangle operations

Modern structure prediction models such as AlphaFold3 [2], OpenFold3 [9], and Boltz-2 [10] spend much of their runtime and memory on triangle attention and triangle multiplication, two operations inherited from AlphaFold2 [1]. Both update the representation of each token pair (i, j) with the representations of the pairs that tokens i and j each form with every token k . These operations let the models reason about the geometry of a biomolecular system, but their cost is cubic in the number of tokens. Doubling the size of a system increases their compute roughly eightfold. The standard way to reduce this cost is to write GPU kernels, low-level programs that fuse and schedule many small operations into a few efficient ones [11, 12]. NVIDIA has developed dedicated kernels for triangle attention and multiplication, first in cuEquivariance [13] and more recently in the BioNeMo Inference Runtime (BioNeMo-IR) [14].

We worked with Claude to develop FlashPairformer, our own kernels for both operations, and benchmarked its current version (v1) against the field-standard kernels [13] (Figure 2). At pair width 128, used by AlphaFold3 and Boltz-2, FlashPairformer v1 ran triangle attention $2.7\times$ faster and triangle multiplication $1.7\times$ faster than the field standard [13]. At pair width 256, used by Protenix v2, it ran them $2.9\times$ and $3.2\times$ faster. These speed-ups are geometric means over seven input sizes from 256 to 2,048 tokens on one NVIDIA H100 GPU, and FlashPairformer v1 was faster at every size (Section S9).

Claude accelerated more than 30 biomolecular models

Kernels that transfer across models account for only part of the gains. We also asked Claude to find optimizations specific to each individual model. Typical changes included memoizing intermediate results that the original code recomputed, constant-folding branches that always produce the same result, capturing GPU operations once and replaying them as CUDA graphs without Python overhead, fusing chains of small operations into single kernels, avoiding host–

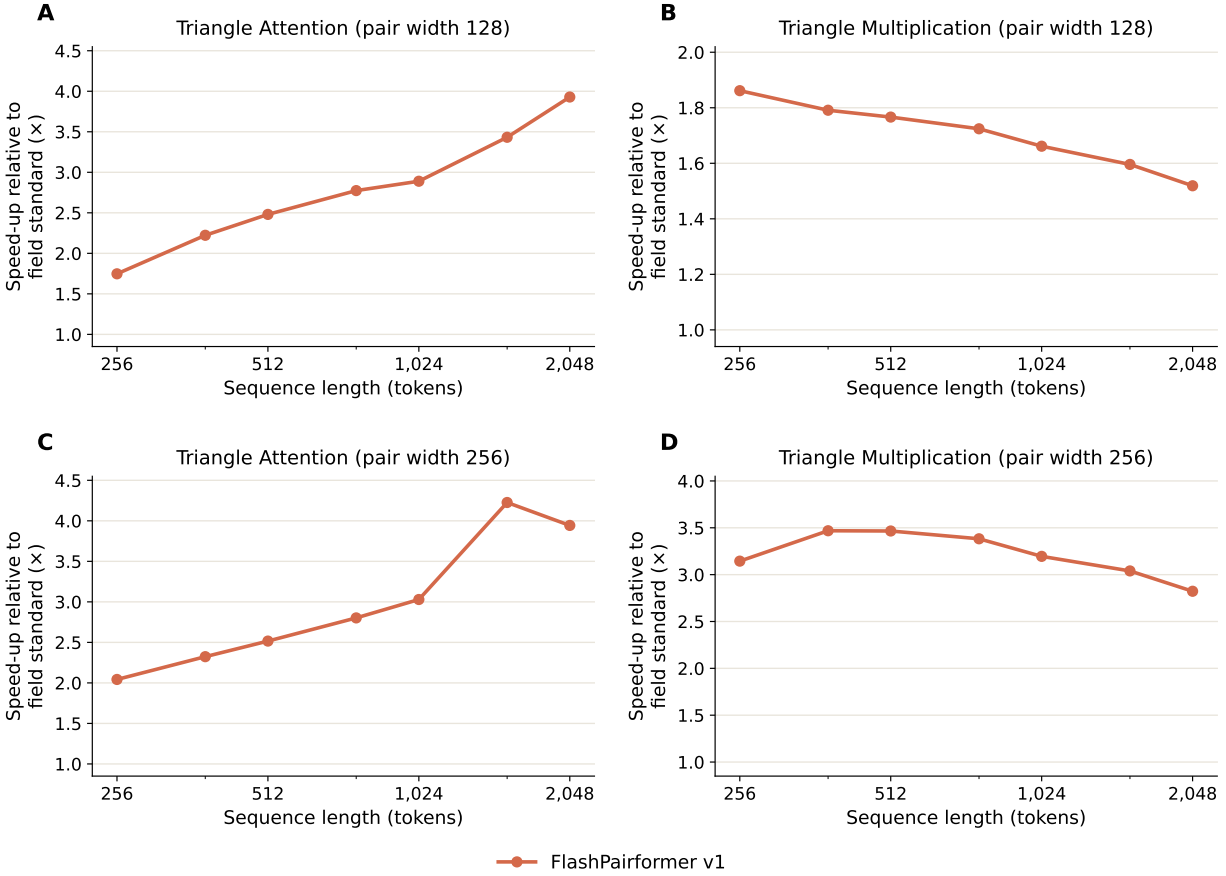


Figure 2. FlashPairformer kernels accelerate triangle attention and triangle multiplication. Speed-up of FlashPairformer v1 relative to the field-standard kernels [13] versus sequence length, for triangle attention (one Pairformer block’s starting- plus ending-node calls) and the full triangle multiplication layer (outgoing plus incoming). **A, B**, pair width 128 (triangle attention with 4 heads of 32 channels), used by AlphaFold3 and Boltz-2. **C, D**, pair width 256 (8 heads of 32 channels), used by Protenix v2. Each point is the field-standard kernel’s time divided by FlashPairformer v1’s time. Each kernel receives its inputs already in its own memory layout, so only the kernel’s work is timed. Times are medians of at least 20 CUDA-graph replays, with the two kernels replayed in interleaved blocks of alternating order. All timings are forward passes at batch size 1 on one NVIDIA H100 80 GB GPU, with unmasked inputs in bfloat16 precision (Section S9).

device transfers by keeping tensors on the GPU, and running data loading and output writing asynchronously so that the GPU is not stalled on the CPU. Sections S1–S6 give each model’s benchmark setup and changes.

Every optimized package runs with the original weights and has up to three modes. *Exact* changes only how the computation is executed. Its outputs were compared bit for bit with the unmodified model’s outputs, for several models with deterministic settings enabled in both. *Fast* also allows changes that can alter results slightly, such as lower-precision arithmetic or fused kernels that round differently. Model-specific tests compared Fast’s outputs with default’s; for most models, the tests checked that Fast’s outputs stay within the seed-to-seed spread of default’s outputs. *Big* reduces peak memory so that larger inputs fit in GPU memory. For example, it processes the largest intermediate tensors in chunks or, on several GPUs, splits the pair representation across the GPUs of one machine. Big was checked like Fast. The SI reports each model’s check outcomes.

We benchmarked each package against a strong baseline, which we call *default*. Default is the unmodified model, run in the fastest correct way available to a knowledgeable user on the same GPU. For example, we enabled the model’s optional fused kernels, used a batch size that saturates the GPU, or let several processes share one GPU. We call the unmodified model run exactly as its authors ship it *base*; wherever default differs from base, the SI states the difference. All speed and memory measurements in this report were made on NVIDIA H100 80 GB GPUs. The timed quantity differs by model family, because users care about different quantities. We timed the forward (prediction) call per complex for structure prediction, each design step for hallucination, designs per GPU-second for structure generation, the end-to-end wall time of a design pass (including start-up) for inverse folding, both the forward pass and the whole task for genomics models, and the time per sequence for protein language models (SI, p. 16).

Figure 3 shows the forward-pass speed-ups for 14 structure prediction models (15 configurations, counting ESMFold2’s two checkpoints separately), each averaged (geometric mean) over the input sizes benchmarked, typically seven sizes from 200 to 1,400 tokens. Fast mode was 2.3× (OpenDDE) to 6.4× (Chai-1) faster than default, and 4.1× faster on average. Exact mode, with bit-identical outputs, was 1.6× faster on average, and Big mode 3.4× faster. AlphaFold3 (PyTorch) is measured against AlphaFold3 (JAX) at default, which is why its Exact bar is below 1×. Measured against its own default configuration, its Exact mode is 2.9× faster (SI). The optimizations also accelerated protein design models of several architectures, including structure-prediction networks used for hallucination [15], diffusion and flow-matching models for structure generation [3, 16–20], and graph neural networks and transformers for inverse folding [4, 21] (Figure 4). No optimized mode measurably changed average design quality in four design-quality benchmarks (SI). For genomics models [5, 6, 22] and protein language models [23, 24], Exact mode made the forward pass 1.2–5.0× faster than default (Figure 5); the SI also covers whole genomics tasks, in which input preparation and output writing can take longer than inference.

Such optimizations can take an experienced engineering team weeks per model, and the work often does not transfer between models. Claude optimized all of these models in just under four weeks, supervised primarily by two members of Anthropic’s technical staff with experience in biomolecular modeling but none in inference optimization or kernel engineering.

Faster modes preserve structure prediction accuracy

We tested whether the optimized modes change accuracy for every structure prediction model except AF2 initial guess. We ran every mode on FoldBench-Lite, which combines part of the FoldBench benchmark [25] with complexes from more recent PDB entries (Methods). For each model, we took the top-ranked prediction by the model’s own confidence score and scored it in two ways. Interfaces (protein–protein, antibody–antigen, protein–peptide, protein–DNA, and protein–RNA) were scored by DockQ [26, 27] and called acceptable at a DockQ score of at least 0.23, the threshold of the CAPRI acceptable-quality class. Single chains were scored by IDDT [28]. The SI also compares ligand poses (ligand RMSD) and the models’ own confidence scores.

Interface accuracy did not change measurably. Pooled over 13 model configurations (1,925 model–target pairs, with 124 to 153 targets per configuration), the share of acceptable interfaces was 54.8% at default, 55.0% for Exact, 54.5% for Fast, and 54.2% for Big (Figure 6B). None of the three pooled changes (+0.2, −0.4, and −0.6 percentage points, computed before rounding) had a 95% confidence interval (CI) that excluded zero. Of 41 per-model comparisons with default, only the Exact mode of ESMFold2 had a 95% CI that excluded zero (+4.6 percentage points). No per-model decrease in interface success was distinguishable from zero. However, six intervals end exactly at zero, including those of three of the four largest decreases, which ranged from 2.2 to 3.5 percentage points (OpenFold3-p2 Fast and Big, RoseTTAFold3 Exact and Big). For example, RoseTTAFold3 Big changed by −3.5 points, 95% CI −7.7 to 0.0, so

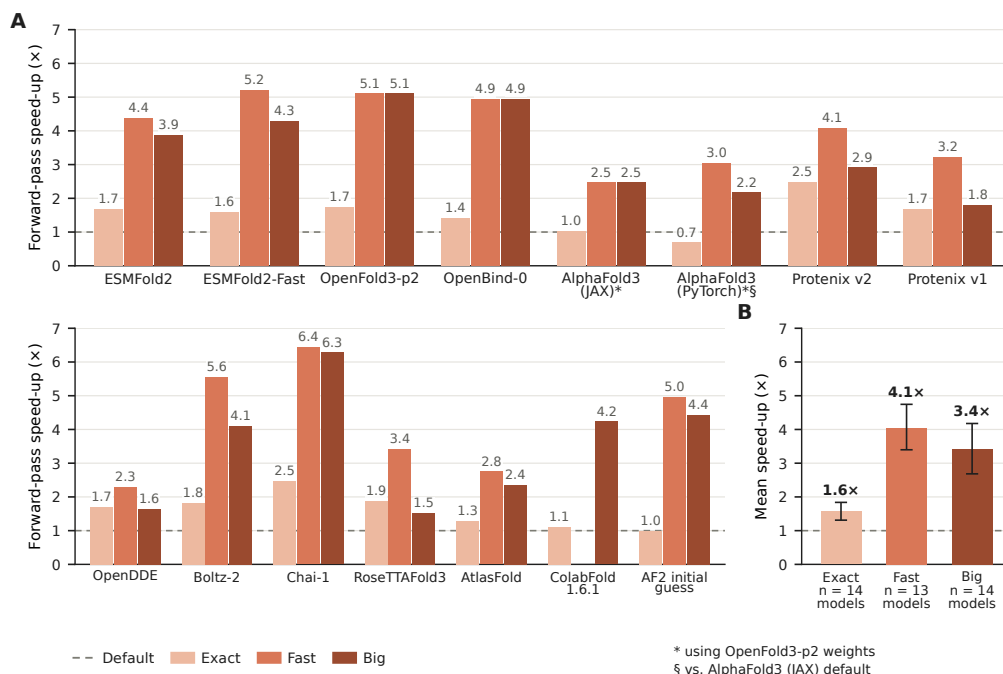


Figure 3. Claude’s optimizations accelerate biomolecular structure prediction models. A, Forward-pass speed-up over default (dashed line, 1x) for each mode on NVIDIA H100 GPUs. Each bar is the geometric mean over the benchmarked input sizes (five for ESMFold2, seven for all other models) of the per-size ratio of default’s forward-call time to the mode’s (for Chai-1, the time of the network itself, without featurization and output). *AlphaFold3 (JAX and PyTorch) was run with OpenFold3-p2 weights. §AlphaFold3 (PyTorch) is compared with the forward call of AlphaFold3 (JAX) at default. ColabFold has no separate Fast mode; its Fast configuration is identical to Big. **B**, Arithmetic mean over models of the per-model bars in A (ESMFold2 counted once, without ESMFold2-Fast); error bars show 95% percentile bootstrap confidence intervals over models (20,000 resamples).

small losses in these modes cannot be ruled out. When each target, rather than each interface, is weighted equally, the interval for OpenFold3-p2 Fast (−3.4 points) ends just below zero.

Single-chain accuracy also changed little. These per-model comparisons are smaller (9–13 chains each). Their 95% CIs excluded zero only for six small decreases in mean IDDT (0.003–0.040) and six small increases (SI).

Exact can differ slightly from default on FoldBench-Lite for a reason unrelated to the optimizations. The benchmark ran each model at its standard inference settings, as users would run it, and several of the models are not deterministic at these settings, so default itself gives slightly different predictions from run to run. This is known for OpenFold3-p2, OpenBind-0, and AlphaFold3 (JAX), and a repeat of default also changed some predictions of Proteinix v1, Proteinix v2, and ColabFold. We therefore checked the bit-identity of Exact and default separately, with deterministic settings enabled in both, and it held on the FoldBench-Lite inputs for 10 of the 11 configurations checked (SI). Small differences between Exact and default on the benchmark therefore most likely reflect run-to-run variation rather than the optimizations.

Big mode enables modeling of massive biomolecular systems

Large molecular machines, such as the ribosome, respiratory complexes, and molecular chaperones, do much of the work in a cell. Many consist of dozens of subunits and function only when the subunits assemble correctly. Predicting the structures of such large systems has typically required inference across several GPU nodes [29], a computing resource that most molecular biologists lack.

We asked Claude to reduce the memory needed to model such systems. The resulting Big mode enabled accurate predictions of systems larger than 10,000 tokens (amino acids, nucleotides, and atoms from small molecules and ions) on a single 8-GPU node (Figure 7). Molecular machines predicted accurately in Big mode include human mitochondrial complex I, the chaperonin TRiC, a proteasome, and a bacterial ribosome, each closely matching its experimental

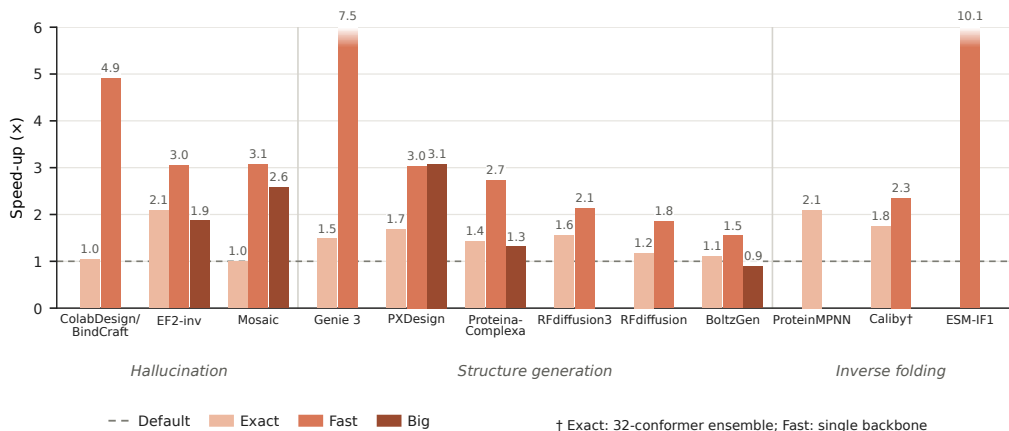


Figure 4. Claude’s optimizations accelerate protein design models spanning hallucination, structure generation, and inverse folding. Speed-up over default (dashed line, 1×) on NVIDIA H100 GPUs, as the geometric mean over the benchmarked input sizes (for Genie 3 and RFdiffusion, the unconditional-design sizes; SI). Clocks follow each model family: time per design step for hallucination, designs per GPU-second for structure generation, and the end-to-end wall time of a design pass, including model loading and other one-time costs, for inverse folding. The hallucination and structure-generation clocks exclude start-up and compilation (end-to-end values: SI). Faded bars extend beyond the axis and are labeled with their values. †For Caliby, Exact is measured on ensemble-conditioned design (32 conformers per backbone) and Fast on single-backbone design. Per-model definitions are given in the SI.

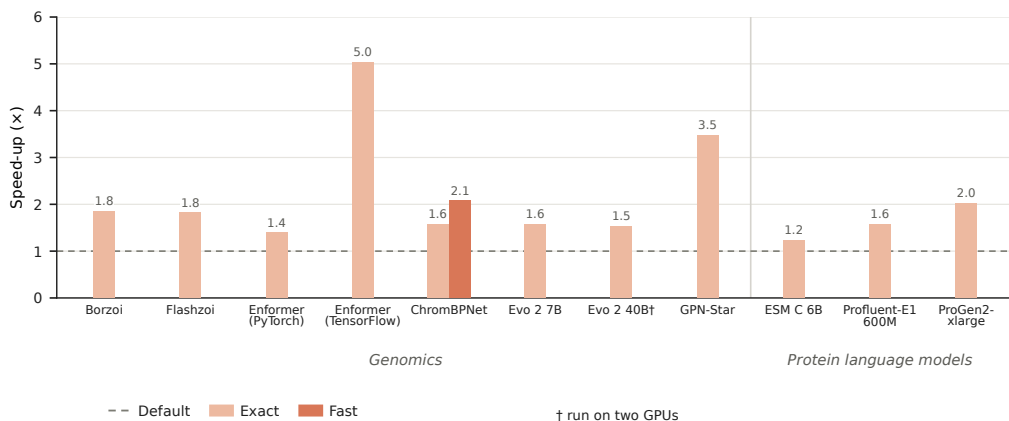


Figure 5. Claude’s optimizations accelerate genomics models and protein language models. Speed-up of each model’s forward pass over default (dashed line, 1×) on one NVIDIA H100 80 GB GPU, computed as the default time divided by the mode’s time for the same input and batch. Each model is timed at its saturating batch size or at a stated batch size; per-model inputs, batches, and timing details are given in the SI. Enformer is shown for both the PyTorch and the original TensorFlow implementation, each against its own default. †, the Evo 2 40B model runs on two GPUs. ESM C and Profluent-E1 are shown at their largest sizes. Exact is bit-identical to default; ChromBPNet’s Fast mode, which runs convolutions in TF32 as the unmodified model does, is not, but stays within default’s own pass-to-pass variation (SI).

structure (TM-scores from 0.92 to 0.997; top three rows of Figure 7). Six of the accurately predicted systems, including three complex I structures and the 70S ribosome, contain more than 10,000 tokens. For comparison, the AlphaFold3 paper highlighted an accurate prediction of the human 40S ribosomal subunit, of 7,663 residues [2]. Flagship Labs recently reported predicting the same *Escherichia coli* 70S ribosome (PDB 4YBB) on a single NVIDIA RTX PRO 6000 GPU with LightFold, its proprietary co-folding model [30]. The yeast TRiC-tubulin complex predicted here (PDB 7YLV) contains 9,611 residues (10,147 tokens, because each non-hydrogen atom of its 47 bound ligands and ions is one token). AlphaFold3, whose architecture most of these models follow, was trained on crops of at most 768 tokens [2]. The six largest accurately predicted systems (all using structural templates) are more than 13 times that size. Not every system was predicted accurately. The bottom row of Figure 7 shows five predictions that deviate substantially

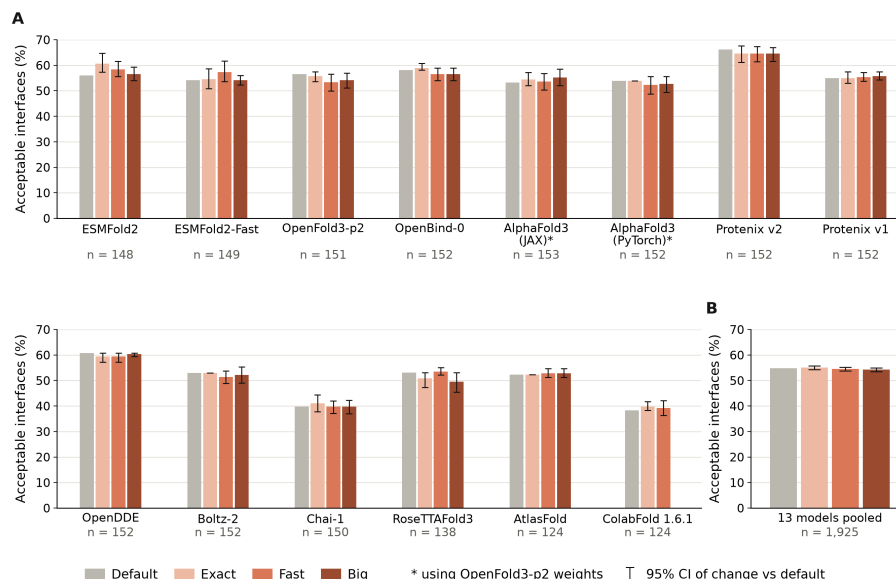


Figure 6. The optimized modes are statistically indistinguishable from default across a pooled set of biomolecular interfaces. **A**, Share of FoldBench-Lite interfaces (protein–protein, antibody–antigen, protein–peptide, protein–DNA, and protein–RNA) whose top-ranked prediction reaches a DockQ score of at least 0.23, for each model and mode; n is the number of targets. Whiskers show the 95% percentile bootstrap CI of the paired change from default (20,000 resamples of targets), drawn at the default rate plus the interval bounds, so that they bracket the top of the mode’s bar. *AlphaFold3 (JAX and PyTorch) used OpenFold3-p2 weights. ColabFold’s second optimized configuration is labeled Fast here; it is the same configuration as Big in Figure 3. **B**, The 13 configurations with all four modes, pooled (ColabFold excluded; 1,925 model–target pairs; the panel label counts both ESMFold2 checkpoints).

from the experimental structures. Two of these five systems, Huc hydrogenase and the SIR2 dodecamer, were predicted accurately by another model.

We then tested the limits of Big mode by asking Claude to predict structures of a size comparable to, and larger than, the largest assemblies previously reported for models of this class [29]. Claude ran structure prediction on entire viral capsids and protein compartments of roughly 31,000 to 70,000 residues (Figure 8), using a single node of eight B300 GPUs and, for four of the seven runs, a shortened protocol (a single trunk pass without recycling). While all seven predictions were produced, none was accurate. Instead, each prediction collapsed into a compact, interpenetrating ball roughly a quarter of the diameter of the real assembly, with TM-scores of 0.08–0.14 where scored. We hypothesize that this collapse reflects a failure to generalize, because these assemblies are 40 to 90 times larger than AlphaFold3’s largest training crop. The shortened protocol may contribute, but the AAV2 run collapsed despite recycling and templates. The runs nevertheless show that inference at this scale now requires only one GPU node, which lowers the barrier to modeling increasingly complex biological systems as the models themselves improve.

Claude efficiently designs *de novo* protein binders

In our earlier binder design study [7], Claude worked from a roughly 16,000-word prompt that encouraged it to use sub-agents. Each single-target campaign could spend up to \$10,000 on Modal cloud GPUs (roughly 2,500 H100 GPU hours at list prices) within 24 hours. Here we gave a single Claude model one NVIDIA H200 GPU and 24 hours of wall time, with no internet access, no sub-agents, and no human steering of the designs. Its roughly 1,100-word prompt briefly described the target, pointed to the target’s structures, a folder of 46 papers from the recent protein design literature (largely the same as in the earlier study), and a reference sheet for the pre-installed design tools, including the accelerated models, and asked for 30 designs that make a stated score, which weighs design quality and diversity, as high as possible (Section S8.3 gives the formula). We ran three Claude models (Mythos 5.1, Mythos 5, and Opus 5) five times each against the 16 targets of the earlier study. In this report, we score the designs by ipSAE alone [31], an *in silico* binding score that has been shown to predict experimental binding [32].

The median-scoring and highest-scoring designs of all three Claude models, averaged over the 16 targets, approached or exceeded the corresponding scores of the final designs from 16 single-target campaigns in which Mythos

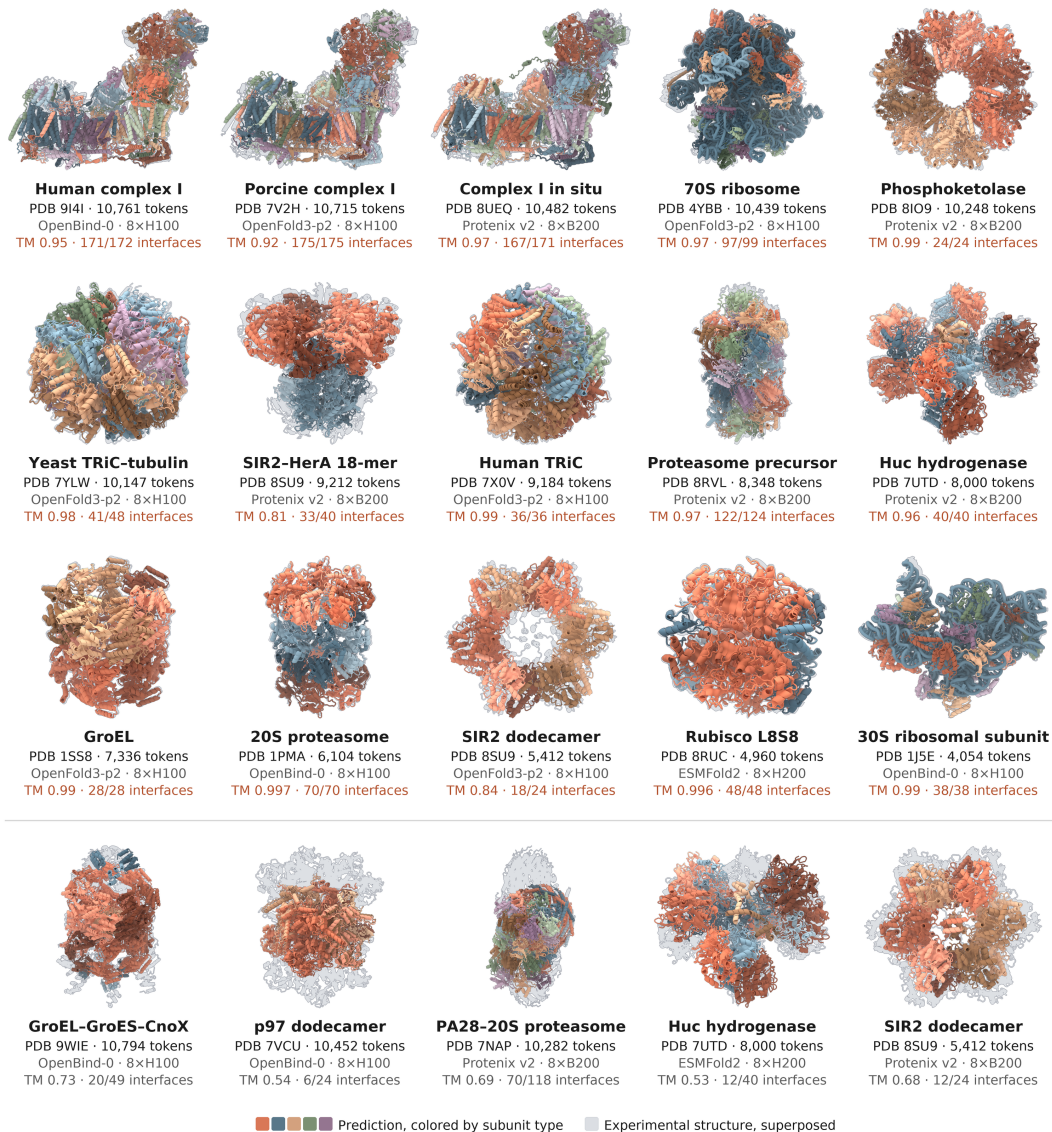


Figure 7. Big mode enables accurate prediction of biomolecular systems of up to 10,761 tokens on a single GPU node. Each tile shows a prediction (colored by subunit type) superposed on the experimental structure (gray), with the PDB entry and number of tokens, the model and hardware (eight GPUs of the stated type), its TM-score against the experimental structure, and the number of interfaces predicted acceptably (DockQ ≥ 0.23) out of all interfaces. The top three rows show accurate predictions (TM-score ≥ 0.8 , $\geq 70\%$ of interfaces acceptable, no clash-flagged chain; criteria fixed before scoring), the bottom row inaccurate ones.

5.1 could use sub-agents (Figure 9; Methods). The single-GPU runs used a GPU budget about 100 times smaller (24 H200 GPU hours, against up to \$10,000 or roughly 2,500 H100 GPU hours at list prices). After 24 hours, the median design scored 0.785 for Opus 5, 0.781 for Mythos 5.1, and 0.739 for Mythos 5, against 0.749 for the earlier campaigns. The best design scored 0.833, 0.825, and 0.813, respectively, against 0.817. In the hourly records, the target-averaged median score reached the earlier campaigns' median after 12 hours for Opus 5 runs (at a combined GPU and token cost of about \$117–136 per run) and after 13 hours for Mythos 5.1 runs (about \$157–196 per run), but not within 24 hours for Mythos 5 runs. Designs from these runs have not been tested experimentally. Section S8 reproduces the prompt of these runs and reports further results, including runs without the accelerated models.

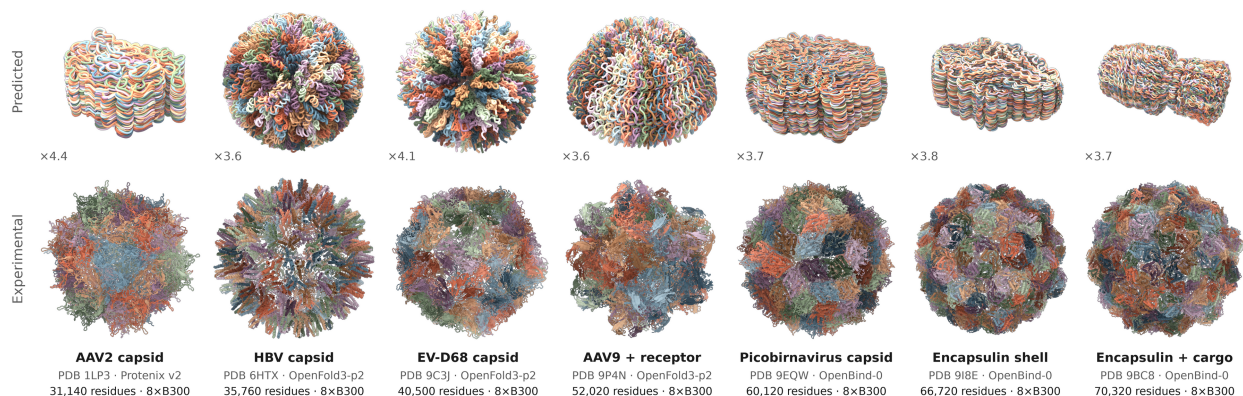


Figure 8. Big mode runs structure prediction on assemblies of roughly 31,000 to 70,000 residues on a single NVIDIA B300 node, but the predictions collapse. Top, single-sample predictions for seven complete viral capsids and protein shells; bottom, the experimental assemblies each run was asked to reproduce (PDB 1LP3, 6HTX, 9C3J, 9P4N, 9EQW, 9I8E, and 9BC8). Labels give the model and the number of residues (one token per residue for these all-protein systems); all runs used one node of eight B300 GPUs, and the AAV2, EV-D68, and HBV runs used separate tensor-parallel versions of the models. The EV-D68, AAV9-with-receptor, picobirnavirus, and encapsulin-shell runs used a single trunk pass without recycling, and the AAV2 run used 10 recycling steps and crystal-structure templates. Every run completed, but every prediction collapsed (TM-scores of 0.14, 0.08, and 0.08 for the three that were scored). Each tile is scaled to fill its box; the gray $\times n$ gives each prediction’s magnification relative to the experimental assembly beneath it.

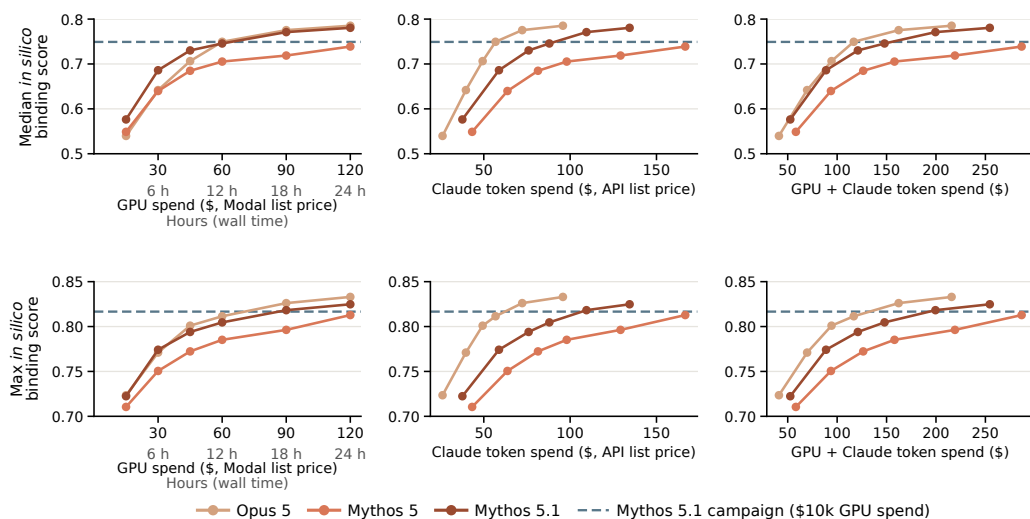


Figure 9. A single Claude model with one NVIDIA H200 GPU and 24 hours of wall time designs *de novo* binders with *in silico* scores comparable to earlier campaigns that could use about 100 times as many GPU hours. Each curve shows, averaged over 16 targets and up to five independent runs per target, the median (top) or maximum (bottom) *in silico* binding score of the designs a Claude model had selected at 3, 6, 9, 12, 18, and 24 hours. The *in silico* binding score is ipSAE [31], which measures, from 0 to 1, a structure predictor’s confidence in the predicted interface between binder and target. A design’s score is the lower of its complex’s two directional ipSAE values (best of five seeds), averaged over ESMFold2, ESMFold2-Fast, and Protenix v2; for the two GDF-8 targets, half of the corresponding score against the related protein GDF-11 is subtracted. Scores are plotted against estimated GPU cost (left; \$5 per GPU hour, see Methods; wall-clock hours below), Claude token cost at API list prices (center), and their sum (right). The dashed line marks the median or maximum score of the final 30 designs of earlier single-target campaigns by Claude Mythos 5.1, which could use sub-agents and up to \$10,000 of GPU time per target, scored the same way and averaged over the same targets (Methods). Vertical axes do not start at zero.

Discussion

Within four weeks, Claude accelerated more than 30 widely used biomolecular models, made most of them more memory-efficient, and checked nearly all optimized modes against default. Its FlashPairformer v1 kernels, for the triangle operations that account for much of the cost of modern structure predictors, ran triangle attention 2.7–2.9 $\times$ and triangle multiplication 1.7–3.2 $\times$ faster on average than the field-standard kernels [13], depending on the pair width. Its memory optimizations allow accurate modeling of molecular machines larger than 10,000 tokens on one GPU node. The optimized models also supported *in silico* binder design of comparable quality on a GPU budget about 100 times smaller, with a simpler agentic protocol. Given a well-specified objective and suitable tools, Claude is able to optimize its designs effectively with minimal guidance.

Several limitations apply. All speed and memory results come from NVIDIA H100 GPUs at the batch sizes, input sizes, and upstream versions stated in the SI, and they may differ elsewhere. For example, our ColabFold speed-ups are measured against version 1.6.1; later ColabFold releases add optional fused kernels of their own, which we have not benchmarked [33, 34]. Our summary bars also average over the input sizes we benchmarked. Exact and Fast can use more memory than default (up to 3.2 times; SI), so Big is the mode to use when memory is limiting. We measured downstream accuracy for the design models and for every structure prediction model except AF2 initial guess. Genomics and protein language models were only compared with default’s outputs, which all matched. A few checks of other models did not pass (SI). Predictions far beyond the models’ training context collapsed (Figure 8), so Big mode extends what can be computed, not what the models have learned. The design results are *in silico* scores rather than experimental measurements. We did not separate the contributions of the optimized models and of the simpler protocol, and the token costs are list-price estimates.

Our results suggest that frontier AI models can help scientists build and maintain their computational tools with greater speed and ease. We are releasing the optimized code so that the community can use it.

Methods

Models and optimized packages. The 36 optimized packages fall into six families: co-folding and structure prediction (14), hallucination (3), structure generation (6), inverse folding (3), genomics (7), and protein language models (3). AlphaFold3 was optimized in two community implementations, a JAX fork of Google DeepMind’s inference code and xfold, an independent PyTorch reimplement, both run with OpenFold3-p2 weights rather than AlphaFold3’s parameters; OpenFold3 was run with its preview-2 weights (OpenFold3-p2) and with the OpenBind-0 weights [9]; and Enformer was optimized in two implementations (SI). The structure prediction models are ESMFold2 [35], OpenFold3 [9], AlphaFold3 [2, 36, 37], Protenix v1 and v2 [38–40], OpenDDE [41], Boltz-2 [10], Chai-1 [42], RoseTTAFold3 [43], AtlasFold [44], ColabFold [45], and AF2 initial guess [46]. The design models are ColabDesign/BindCraft [15, 47], ESMFold2-inverse hallucination (EF2-inv, built on ESMFold2 [35]), Mosaic [48], Genie 3 [18], PXDesign [19], Proteina-Complexa [20], RFdiffusion3 [16], RFdiffusion [3], BoltzGen [17], ProteinMPNN [4], Caliby [49], and ESM-IF1 [21]. The genomics models are Evo 2 [22], Borzoi [6], Flashzoi [50], ChromBPNet [51], Enformer [5], and GPN-Star [52], and the protein language models are ESM C [24], Profluent-E1 [53], and ProGen2 [23].

Benchmark protocol. Default is the unmodified model at the fastest correct configuration we could find on the same GPU (SI, p. 16). Speed-ups are ratios of default time to mode time at the same input size and settings, with batch sizes set by the per-model rules in the SI. Each mode was timed in the same runs as default. Times exclude warm-up, except for the inverse-folding clock and the whole-task genomics clock, which are end-to-end wall times that include model loading and other one-time costs. Peak memory is the device-level peak above idle reported by the NVIDIA Management Library unless the SI states otherwise. Exact mode was checked for bit-identical outputs on model-specific input sets (for several models, with deterministic settings), setting aside outputs that default itself did not reproduce. Fast and Big modes were checked by model-specific tests: staying within default’s seed-to-seed spread (structure prediction and inverse folding), a step-by-step comparison of the design optimization (hallucination), or a bitwise comparison (some structure generation and genomics packages). Values here are rounded; the SI truncates speed-ups (SI, p. 16).

Structure prediction accuracy. FoldBench-Lite combines complexes from FoldBench [25], complexes from PDB entries released from July 2025 onward (selected with rules adapted from FoldBench, for diversity of sequence and size), and additional antibody–antigen complexes. Of its 249 targets, Figure 6 uses the 171 (273 interfaces: 132 antibody–antigen, 61 protein–protein, 31 protein–DNA, 27 protein–RNA, and 22 protein–peptide) with at least one such interface

scored under default and all optimized modes by at least one model; of the other 78, 76 are protein–ligand complexes or protein, DNA, or RNA monomers, which the SI reports by model. Each model is shown on the targets where default and all its modes were scored (AtlasFold and ColabFold, which model proteins only, on protein-only targets with ligands, ions, and residue modifications removed). Each model and mode was run on the same seeds for every target, and the top-1 prediction was the sample ranked best by the model’s own metric over the seeds common to all four modes (ties: lowest seed, then lowest sample index). Interfaces were scored with DockQ [26, 27] and called acceptable at DockQ $\geq$ 0.23. Confidence intervals come from a paired percentile bootstrap over targets within each model (20,000 resamples, shared by mode and default), combined across models for the pooled intervals.

Large systems. The 20 predictions in Figure 7 (18 assemblies, 17 PDB entries) were selected after scoring from our 190 predictions of 20 assemblies (19 PDB entries) of 4,000 to 11,000 tokens made with up to four models each; they ran in Big mode on one node of eight H100, H200, or B200 GPUs, with the pair representation split by rows across the eight GPUs (a setup not benchmarked in the SI), and were scored against the deposited structures by TM-score [54] and per-interface DockQ. Tiles show each run’s most confident of five samples (for the 70S ribosome and GroEL, the sample with the highest TM-score); runs that failed the accuracy criteria were repeated with new seeds and the best seed is shown, as for four accurate tiles (PDB 8IO9, 7YLV, 7X0V, and GroEL). All predictions except those of ESMFold2, the 20S proteasome, and the SIR2 dodecamer used templates released by 30 September 2021, AlphaFold3’s training cutoff (for older entries, 60 days before release); five accurately predicted entries (PDB 1J5E, 1PMA, 1SS8, 4YBB, and 8RUC) predate that date and may be in the models’ training data.

Binder design. Each run (Results) returned 30 designs, folded with ESMFold2, ESMFold2-Fast, and Protenix v2 (five seeds each) and scored by ipSAE_{min} [31] as in the Figure 9 caption. At each checkpoint, each run’s median and best scores are averaged over runs, then over targets. The reference lines come from 16 single-target campaigns that Claude Mythos 5.1 ran on August 8–11, 2026, whose final 30 designs were rescored the same way. Each could use sub-agents and up to \$10,000 of GPU time, as could the campaigns of our earlier study [7] (run by Claude Mythos Preview and, for three targets, also Claude Opus 4.8). The GPU rate of \$5 per hour is the Modal list price of one H200 GPU (\$4.54 per hour) plus an allowance for host CPU and memory (e.g., \$0.44 per hour for four physical CPU cores and 32 GiB), rounded up [55]. Token costs are estimates at API list prices (for Mythos 5.1, those of Claude Fable 5.1) for the design model’s own tokens, at both the five-minute and one-hour prompt-cache write rates. In the hourly records, the reference median (0.7491) is first reached at 12 hours by Opus 5 (0.7494) and at 13 by Mythos 5.1 (0.7507).

Data and code availability

The optimized code for all models is available at <https://github.com/anthropics/uplifting-biomolecular-modeling>.

Acknowledgments

We thank the developers of the open-source models optimized in this work for releasing their code and weights, and NVIDIA for the cuEquivariance and BioNeMo-IR software. We thank Aleksei Lorenz for improvements to Claude Science; Xander Balwit for editorial review; Nathan Frey for discussions and review; and Martin Pacesa (University of Zurich), Mohammed AlQuraishi and Christina Floristean (Columbia University), and Roshan Rao and Zeming Lin (Biohub) for discussions and for reviewing parts of this work ahead of its release.

Competing interests

C.S., R.S., R.B., V.F., and A.S. are affiliated with Anthropic.

Use of AI tools

Claude, an AI model developed by Anthropic, performed the optimization work and data analysis described here and drafted this report under human supervision.

References

- [1] John Jumper et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596:583–589, 2021. doi:10.1038/s41586-021-03819-2.
- [2] Josh Abramson et al. Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, 630:493–500, 2024. doi:10.1038/s41586-024-07487-w.
- [3] Joseph L. Watson et al. *De novo* design of protein structure and function with RFdiffusion. *Nature*, 620:1089–1100, 2023. doi:10.1038/s41586-023-06415-8.
- [4] Justas Dauparas et al. Robust deep learning-based protein sequence design using ProteinMPNN. *Science*, 378:49–56, 2022. doi:10.1126/science.add2187.
- [5] Žiga Avsec et al. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, 18:1196–1203, 2021. doi:10.1038/s41592-021-01252-x.
- [6] Johannes Linder, Divyanshi Srivastava, Han Yuan, Vikram Agarwal, and David R. Kelley. Predicting RNA-seq coverage from DNA sequence as a unifying model of gene regulation. *Nature Genetics*, 57:949–961, 2025. doi:10.1038/s41588-024-02053-6.
- [7] Claude Science and Amir Shanhazzadeh. Autonomous *de novo* protein binder design with Claude. Technical report, Anthropic, August 2026. <https://www-cdn.anthropic.com/30bf50e22a01388bb29bf077ee3f244531594b7a.pdf>. Data and prompts: <https://huggingface.co/datasets/Anthropic/claude-protein-binder-design>.
- [8] Anthropic. Introducing Claude Fable 5.1 and Claude Mythos 5.1. <https://www.anthropic.com/claude-fable-and-mythos-5-1>, 2026.
- [9] The OpenFold3 Team. OpenFold3-preview: open-source biomolecular structure prediction (software; openfold3 releases 0.4.1 and 0.5.0; the 0.5.0 default checkpoint is OpenBind-0). <https://github.com/aqlaboratory/openfold-3>, 2026. Release 0.4.1, doi:10.5281/zenodo.19485145; release 0.5.0, doi:10.5281/zenodo.22042719.
- [10] Saro Passaro et al. Boltz-2: Towards accurate and efficient binding affinity prediction. bioRxiv, 2025. doi:10.1101/2025.06.14.659707.
- [11] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2205.14135.
- [12] Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019. doi:10.1145/3315508.3329973.
- [13] NVIDIA. cuEquivariance: a library of low-level primitives and tensor operations to accelerate equivariant neural networks. <https://github.com/NVIDIA/cuEquivariance>, 2026.
- [14] NVIDIA. BioNeMo inference runtime: easy, fast, and memory-efficient structure prediction inference. <https://github.com/NVIDIA-BioNeMo/BioNeMo-Inference-Runtime>; documentation: <https://docs.nvidia.com/bionemo/inference-runtime/overview/>, 2026.
- [15] Martin Pacesa et al. One-shot design of functional protein binders with BindCraft. *Nature*, 646:483–492, 2025. doi:10.1038/s41586-025-09429-6.
- [16] Jasper Butcher et al. *De novo* design of all-atom biomolecular interactions with RFdiffusion3. bioRxiv, 2025. doi:10.1101/2025.09.18.676967.
- [17] Hannes Stark et al. BoltzGen: Toward universal binder design. bioRxiv, 2025. doi:10.1101/2025.11.20.689494.
- [18] Yeqing Lin et al. Fast and ultra-capable protein design: advancing the frontier through atomistic SE(3)-equivariance with Genie 3. bioRxiv, 2026. doi:10.64898/2026.05.01.722168.
- [19] Protenix Team et al. PXDesign: Fast, modular, and accurate *de novo* design of protein binders. bioRxiv, 2025. doi:10.1101/2025.08.15.670450.
- [20] Kieran Didi et al. Scaling atomistic protein binder design with generative pretraining and test-time compute. arXiv:2603.27950, 2026. doi:10.48550/arXiv.2603.27950.
- [21] Chloe Hsu et al. Learning inverse folding from millions of predicted structures. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022. bioRxiv doi:10.1101/2022.04.10.487779.
- [22] Garyk Brixi et al. Genome modeling and design across all domains of life with Evo 2. bioRxiv, 2025. doi:10.1101/2025.02.18.638918.
- [23] Erik Nijkamp, Jeffrey A. Ruffolo, Eli N. Weinstein, Nikhil Naik, and Ali Madani. ProGen2: Exploring the boundaries of protein language models. *Cell Systems*, 14:968–978, 2023. doi:10.1016/j.cels.2023.10.002.
- [24] ESM Team. ESM Cambrian: Revealing the mysteries of proteins with unsupervised learning. EvolutionaryScale website, <https://www.evolutionaryscale.ai/blog/esm-cambrian>, 2024.
- [25] Sheng Xu et al. Benchmarking all-atom biomolecular structure prediction with FoldBench. *Nature Communications*, 17:442, 2025. doi:10.1038/s41467-025-67127-3.
- [26] Sankar Basu and Björn Wallner. DockQ: A quality measure for protein-protein docking models. *PLOS ONE*, 11:e0161879,

2016. doi:10.1371/journal.pone.0161879.
- [27] Claudio Mirabello and Björn Wallner. DockQ v2: improved automatic quality measure for protein multimers, nucleic acids, and small molecules. *Bioinformatics*, 40:btac586, 2024. doi:10.1093/bioinformatics/btac586.
 - [28] Valerio Mariani, Marco Biasini, Alessandro Barbato, and Torsten Schwede. IDDT: a local superposition-free score for comparing protein structures and models using distance difference tests. *Bioinformatics*, 29(21):2722–2728, 2013. doi:10.1093/bioinformatics/btt473.
 - [29] Dejun Lin, Simon Chu, Vishanth Iyer, et al. Fold-CP: A context parallelism framework for biomolecular modeling. arXiv:2603.14806, 2026. doi:10.48550/arXiv.2603.14806.
 - [30] Flagship Labs. Structure prediction at the scale of biology. The Labs Report (Flagship Labs), <https://flagshiplabs.substack.com/p/structure-prediction-at-the-scale>, August 2026.
 - [31] Roland L. Dunbrack. Rēs ipSAE loquuntur: What’s wrong with AlphaFold’s ipTM score and how to fix it. bioRxiv, 2025. doi:10.1101/2025.02.10.637595.
 - [32] Max D. Overath et al. Predicting experimental success in *de novo* binder design: A meta-analysis of 3,766 experimentally characterised binders. bioRxiv, 2025. doi:10.1101/2025.08.14.670059.
 - [33] ColabFold developers. ColabFold 1.6.2 release notes (published july 14, 2026). <https://github.com/sokrypton/ColabFold/releases/tag/v1.6.2>, 2026.
 - [34] ColabFold developers. ColabFold 1.6.3 release notes (published september 14, 2026). <https://github.com/sokrypton/ColabFold/releases/tag/v1.6.3>, 2026.
 - [35] Salvatore Candido et al. Language modeling materializes a world model of protein biology. bioRxiv, 2026. doi:10.64898/2026.06.03.729735.
 - [36] sokrypton. alphafold3: a fork of the AlphaFold 3 inference pipeline that can load OpenFold3 weights (release v3.1.4). <https://github.com/sokrypton/alphafold3>, 2026.
 - [37] Shenggan. xfold: Democratizing AlphaFold3, a PyTorch reimplement to accelerate protein structure prediction. <https://github.com/Shenggan/xfold>, 2026.
 - [38] ByteDance AML AI4Science Team et al. Protenix – advancing structure prediction through a comprehensive AlphaFold3 reproduction. bioRxiv, 2025. doi:10.1101/2025.01.08.631967.
 - [39] Yuxuan Zhang et al. Protenix-v1: Toward high-accuracy open-source biomolecular structure prediction. bioRxiv, 2026. doi:10.64898/2026.02.05.703733.
 - [40] Yuxuan Zhang et al. Protenix-v2: Broadening the reach of structure prediction and biomolecular design. bioRxiv, 2026. doi:10.64898/2026.04.10.717613.
 - [41] Aureka AI OpenDDE Project. Folding, reasoning, and scaling with open-source drug discovery engine. arXiv:2607.03787, 2026. doi:10.48550/arXiv.2607.03787.
 - [42] Chai Discovery et al. Chai-1: Decoding the molecular interactions of life. bioRxiv, 2024. doi:10.1101/2024.10.10.615955.
 - [43] Nathaniel Corley et al. Accelerating biomolecular modeling with AtomWorks and RF3. bioRxiv, 2025. doi:10.1101/2025.08.14.670328.
 - [44] Seonghwan Seo. AtlasFold: trainable protein folding and co-folding models with a protein language model (release 1.0.0). <https://github.com/SeonghwanSeo/atlasfold>, 2026.
 - [45] Milot Mirdita et al. ColabFold: making protein folding accessible to all. *Nature Methods*, 19:679–682, 2022. doi:10.1038/s41592-022-01488-1.
 - [46] Nathaniel R. Bennett et al. Improving *de novo* protein binder design with deep learning. *Nature Communications*, 14:2625, 2023. doi:10.1038/s41467-023-38328-5.
 - [47] sokrypton. ColabDesign: making protein design accessible to all via Google Colab. <https://github.com/sokrypton/ColabDesign>, 2026.
 - [48] Escalante Bio. mosaic: composite-objective protein design. <https://github.com/escalante-bio/mosaic>, 2025.
 - [49] Richard W. Shuai, Tianyu Lu, Subhang Bhatti, Petr Kouba, and Po-Ssu Huang. Ensemble-conditioned protein sequence design with Caliby. bioRxiv, 2025. doi:10.1101/2025.09.30.679633.
 - [50] Johannes C. Hingerl, Alexander Karollus, and Julien Gagneur. Flashzoi: an enhanced Borzoi for accelerated genomic analysis. *Bioinformatics*, 41:btaf467, 2025. doi:10.1093/bioinformatics/btaf467.
 - [51] Anusri Pampari et al. ChromBPNet: bias factorized, base-resolution deep learning models of chromatin accessibility reveal cis-regulatory sequence syntax, transcription factor footprints and regulatory variants. bioRxiv, 2024. doi:10.1101/2024.12.25.630221.
 - [52] Chengzhong Ye et al. Predicting genome-wide functional constraints with GPN-Star. *Nature*, 2026. doi:10.1038/s41586-026-11005-5.
 - [53] Sarthak Jain et al. E1: Retrieval-augmented protein encoder models. bioRxiv, 2025. doi:10.1101/2025.11.12.688125.
 - [54] Yang Zhang and Jeffrey Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, 57:702–710, 2004. doi:10.1002/prot.20264.

- [55] Modal Labs. Modal pricing. <https://modal.com/pricing>, 2026. Accessed September 16, 2026.
- [56] Biohub. ESMFold2 and ESMFold2-Fast: model weights and the esm inference package. <https://huggingface.co/biohub/ESMFold2>, 2026.
- [57] Biohub. esm: inference code for ESM C, ESMFold2 and related protein models. <https://github.com/Biohub/esm>, 2026.
- [58] OpenBind. OpenBind-0: advancing open molecular structure prediction. <https://openbind.uk/news/blog-openbind-0-advancing-open-molecular-structure-prediction/>, 2026.
- [59] Richard Evans et al. Protein complex prediction with AlphaFold-Multimer. bioRxiv, 2021. doi:10.1101/2021.10.04.463034.
- [60] Lorenzo Tarricone, Helen E. Eisenach, Aiko Muraishi, and Charlotte M. Deane. Design-CP: Context parallelism for design of protein nanoparticles. arXiv:2607.05439, 2026. doi:10.48550/arXiv.2607.05439.
- [61] Meta Fundamental AI Research. Evolutionary scale modeling (esm), fair-esm 2.0.1. <https://github.com/facebookresearch/esm>, 2022.
- [62] Johannes C. Hingerl. borzoi-pytorch: PyTorch implementation of Borzoi and Flashzoi. <https://github.com/johahi/borzoi-pytorch>, 2025.
- [63] Phil Wang. enformer-pytorch: implementation of Enformer in PyTorch. <https://github.com/lucidrains/enformer-pytorch>, 2021.
- [64] Google DeepMind. deepmind-research repository, enformer directory. <https://github.com/google-deepmind/deepmind-research>, 2021.
- [65] Casper A. Goverde, Martin Pacesa, Nicolas Goldbach, et al. Computational design of soluble and functional membrane protein analogues. *Nature*, 631:449–458, 2024. doi:10.1038/s41586-024-07601-y.

Supplementary Information

Accelerating open-source biomolecular models with Claude

S1 Co-folding and structure prediction	19
S1.1 ESMFold2	19
S1.2 ESMFold2-Fast	23
S1.3 OpenFold3-p2	27
S1.4 OpenBind-0	31
S1.5 AlphaFold3 (JAX)	35
S1.6 AlphaFold3 (PyTorch)	39
S1.7 Protenix v2	43
S1.8 Protenix v1	47
S1.9 OpenDDE	51
S1.10 Boltz-2	55
S1.11 Chai-1	59
S1.12 RoseTTAFold3	63
S1.13 AtlasFold	67
S1.14 ColabFold (AF2-Multimer)	71
S1.15 AF2 initial guess	75
S2 Hallucination	77
S2.1 ColabDesign	77
S2.2 ESMFold2-inverse hallucination	79
S2.3 Mosaic	81
S3 Structure generation	83
S3.1 Genie 3	83
S3.2 PXDesign	85
S3.3 Proteina-Complexa	87
S3.4 RFDiffusion3	89
S3.5 RFDiffusion	91
S3.6 BoltzGen	93
S4 Inverse folding	95
S4.1 ProteinMPNN	95
S4.2 Caliby	97
S4.3 ESM-IF1	99
S5 Genomics models	101
S5.1 Evo 2	101
S5.2 Borzoi	104
S5.3 Flashzoi	106
S5.4 ChromBPNet	108
S5.5 Enformer	110
S5.6 Enformer (original)	112
S5.7 GPN-Star	114
S6 Protein language models	116

S6.1	ESM C	116
S6.2	Profluent-E1	118
S6.3	ProGen2-xlarge	121
S7	Design quality under the optimized modes	123
S7.1	BinderBench: binder design	123
S7.2	UnconditionalBench: unconditional backbone generation	124
S7.3	SCBench and NSRBench: inverse folding	125
S8	Binder design: prompt and further results	127
S8.1	Prompt	127
S8.2	Further results	127
S8.3	Prompt texts	130
S9	Kernel benchmark	140

For each optimized model, we report the benchmark setup, the baseline, Claude’s changes, and the results. Sections are grouped by model family: co-folding and structure prediction (Section S1), hallucination (Section S2), structure generation (Section S3), inverse folding (Section S4), genomics models (Section S5), and protein language models (Section S6). Each subsection opens with a short introduction to the model, followed by the benchmark setup, the sampling settings, the available modes, the main changes (listed roughly in order of their estimated contribution), the results, and the figures. The sampling settings (recycles, sampling steps, samples or designs per call, model weights, and inputs) are the same for default and every mode, and upstream’s default is given wherever our setting differs from it. For structure prediction models, recycling was set to give at least 11 trunk passes and diffusion sampling to give at least 5 samples per call, keeping upstream’s values where they were already higher; sampling steps were left at upstream’s defaults. For most structure prediction models, FoldBench-Lite accuracy follows the results. The figures show the headline measurements, with a single value label where points nearly coincide. Companion clocks, comparisons with base, and a few other results, such as Evo 2’s multi-sequence scoring speed-up, appear only in the text, and plotted values are not repeated in tables, except for the kernel benchmark (Table S1).

Default and base. Each optimized package is compared with a baseline that we call *default* here and `stock` in the code implementation: the unmodified upstream model run in the fastest correct way we could find on the same GPU. Default can differ from running the package exactly as its authors ship it, which we call *base* here and `default` in the code implementation. Typical differences are enabling optional fused kernels that the package supports but does not turn on by itself, running at a batch size that saturates the GPU, keeping one process alive for many inputs, or letting several processes share one GPU through NVIDIA’s Multi-Process Service (MPS). Each subsection states only what default adds beyond base. We benchmark against default rather than base so that the reported speed-ups reflect our changes, not configuration choices a knowledgeable user could make without our code. Where the difference between base and default is itself large, the subsection says so.

Software versions and seeds. Most subsections name the package release and checkpoint benchmarked. For every model, the exact upstream commit and the surrounding software stack (PyTorch, JAX, Triton, cuEquivariance or TensorFlow builds), on which kernel availability, bit-identity and speed depend, are listed in that model’s `STOCK.md` and `environment/requirements.lock` in the released repository (see Data and code availability). For the two models whose upstream-pinned framework cannot run on an H100, ProGen2-xlarge and Enformer (original), the subsection names the version used instead. Every stochastic model was run with fixed random seeds, identical for default and every mode wherever upstream accepts a seed, which is what makes Exact’s identity checks and the paired comparisons of modes possible; the forward-only models are deterministic.

Modes. *Exact* changes only how the computation is executed and is designed to produce outputs bit-identical to default; this was checked for every model, and the few checks that did not pass are stated in the subsections. Several models are not bit-for-bit reproducible from run to run at their production settings; for these, identity was checked with the model’s deterministic options enabled, and cases in which default itself was not reproducible were set aside. *Fast* additionally allows changes that can alter outputs slightly, such as lower-precision arithmetic or fused kernels that round differently, and was compared with default by model-specific tests: for most models, whether outputs stay within the spread of default’s own outputs across random seeds or repeated runs (for AF2 initial guess, within a fixed tolerance); for hallucination, a step-by-step comparison of the design optimization; and for Evo 2, a bitwise comparison. For structure generation models other than PXDesign, and for BoltzGen Big, no test with an acceptance threshold is reported; Section S7 compares their designs with default’s. Each subsection states what was checked. *Big*

is the memory-lean mode for larger inputs, with model-dependent effects on peak memory and one-GPU reach; “Big on two GPUs” (labeled Big $\times 2$ in the figures) spreads the computation over two GPUs. Not every model has every mode.

Timed quantities. All speed and memory measurements were made on NVIDIA H100 80 GB GPUs. The quantity that is timed, which we call the clock, depends on the model family, because users of different models care about different things:

- **Co-folding and structure prediction:** the time per complex spent inside the model’s forward (prediction) call, in steady state, excluding warm-up and compilation; what the call includes (for example, input featurization or output writing) varies by model and is stated in each subsection. Input sizes range from 200 to 1,400 tokens; for ESMFold2, speed-ups stop at 1,000 tokens, above which default runs out of memory. All inputs of the co-folding and hallucination benchmarks, and of RFdiffusion’s binder design benchmark, are proteins: protein complexes in the speed and memory benchmarks and, for reach, larger protein assemblies. Because these inputs contain no ligands and co-folding models give each residue one token, the input sizes in tokens given in the text equal the total residues over all chains shown on the figures’ “protein length (residues)” axes. Reach figures show the largest input that completed on one or two GPUs; in every search, the next size tested, 1,000 tokens larger, did not complete. Reach searches test whether an input fits rather than how fast it runs, and for most co-folding models some or all of them used shortened runs with fewer recycling steps, trunk passes, or diffusion steps than the speed benchmarks, in several cases checked against a full-settings run at 3,000 or 4,000 tokens. Reach values are therefore memory limits at the stated number of GPUs, except where an input-size limit ends the search: Chai-1 accepts at most 2,048 tokens, and unmodified Protenix v2 refuses inputs above 2,560 tokens. For Protenix v1 and Protenix v2, shortened runs at 4,000 tokens showed 16% and 6% lower peak memory than full-settings runs, and in Protenix v1’s two-GPU search the next size failed with a communication timeout, counted as running out of memory, so these two-GPU values are the least certain; for AlphaFold3 (PyTorch), OpenDDE, and RoseTTAFold3 on two GPUs, the reported reach may be off by one step of the size ladder.
- **Hallucination:** the steady-state time per design step (one forward and backward pass of the design loss, including any scoring models besides the structure predictor).
- **Structure generation:** designs per GPU-second over the steady-state sampling span, with the whole-pass time as a companion.
- **Inverse folding:** the end-to-end wall time of a design pass, from process start to the last output written, including model loading and other one-time costs; a steady-state sampling clock that excludes these costs is reported alongside.
- **Genomics models:** the forward pass per call for every model. For the three models benchmarked on an end-to-end task (Borzoi, Flashzoi, and ChromBPNet), the wall time of the whole task (including reading and preparing inputs and writing outputs) is shown as well, because this work outside the model can dominate the task time; for Evo 2, generation is timed over the whole generation call and reported per token. Enformer, Enformer (original), and GPN-Star were benchmarked on the forward call only.
- **Protein language models:** the time per sequence inside the forward call. ESM C and Profluent-E1 are timed at batch size 1 and at a saturating batch, the largest batch that default completes on the GPU, with default and the optimized mode run at that same batch; the saturating batch is the fairer comparison for throughput and is the headline condition. Profluent-E1 is also timed with retrieval-augmented inputs. ProGen2 is timed on its two shipped tasks, scoring (one sequence per call) and generation (80 samples per call), where the whole generation call is timed.

A speed-up is the ratio of default’s time to the mode’s time (or of the mode’s throughput to default’s) under otherwise identical settings, unless the subsection states a different batch size or process count. In the figures, a dashed line at $1\times$ marks default.

Memory. Peak memory is the device-level peak above idle reported by the NVIDIA Management Library (NVML), measured in dedicated runs, unless a subsection states otherwise. Some JAX and TensorFlow models reserve GPU memory in large blocks as they run; for these, device-level ratios near or above $1\times$ can reflect the size of the reserved blocks rather than the memory actually in use, and the subsection notes this.

Structure prediction accuracy. Every structure prediction subsection except AF2 initial guess, which was not part of this benchmark, ends with two figures on FoldBench-Lite, the benchmark of main-text Figure 5 (described in the main-text Methods). The first reports accuracy: for each interface class, the share of interfaces whose top-ranked prediction is acceptable, meaning a DockQ score of at least 0.23 [26, 27] (for ligands, a ligand root-mean-square deviation, RMSD, below 2 Å; unlike FoldBench, we do not also require a protein–ligand interaction IDDT (LDDT-PLI) above 0.8), and for protein, DNA, and RNA monomers, the mean IDDT [28] of the top-ranked prediction. The top-ranked prediction is the sample with the best value of the model’s own ranking score over the seeds common to default and all modes, and each model is shown only on the interfaces, ligands, and chains scored under default and all its modes. Intervals are 95% percentile bootstrap intervals from 10,000 resamples of targets (main-text Figure 5’s intervals, computed separately, can differ in the last digit); each mode’s paired change from default is computed on the same resamples. When all of a mode’s changes in a class go in one direction, the interval of its paired change can end exactly at zero; such an interval is counted as including zero, although a small change in that direction cannot be ruled out. These intervals are approximate when few targets change; of the 270 paired comparisons whose interval is not the single point zero, 15 exclude zero,

about as many as expected by chance (13.5). The second figure compares each mode with default point by point: on the left, the accuracy of the top-ranked prediction for each interface, ligand, and chain (DockQ, ligand RMSD, and lDDT, on the units of the first figure); on the right, the model's own confidence scores (ipTM and pTM, the interface and overall predicted TM-scores, and mean pLDDT, the predicted local distance difference test), target by target. Mean pLDDT comes from each model's reported scores or, for models that do not report it (AlphaFold3 (JAX), AlphaFold3 (PyTorch), Chai-1, ESMFold2, and RoseTTAFold3), from the predicted structure files, as the mean over the residues of the polymer chains of each residue's average per-atom pLDDT.

As in the main text, this benchmark ran every model at its production settings. At these settings, default is known not to be reproducible from run to run for OpenFold3-p2, OpenBind-0, and AlphaFold3 (JAX), a repeat of default changed some predictions of Protenix v1, Protenix v2, and ColabFold, and default is known to be reproducible for AlphaFold3 (PyTorch). Exact, which is bit-identical to default under deterministic settings, can therefore differ from default here, as it does for Boltz-2 in one seed each of at least 25 targets, with no change in reported accuracy. For eight models (OpenFold3-p2, OpenBind-0, Protenix v2, Protenix v1, OpenDDE, Boltz-2, RoseTTAFold3, and AtlasFold), the Exact predictions come from a repeat Exact run of the benchmark (for four OpenFold3-p2 targets, from the first run), as in main-text Figure 5.

Rounding. In this supplement, speed-ups and other speed ratios are truncated to the digits shown and never rounded up (a measured speed-up of 3.269 is printed as 3.26× or 3.2×), so that a printed speed-up never overstates the measurement; peak-memory ratios are rounded to the nearest value, since truncating them would overstate the saving. Text and figures follow the same rule. All other quantities, such as accuracy and confidence scores, are rounded to the nearest value. The main text rounds all values to the nearest value, so a speed-up there can exceed the corresponding value here by one unit in the last digit.

Scope. Numbers are reported per model and per input size (the series of input sizes is sometimes called a ladder); this supplement does not average across models, and geometric means in the subsections are over a mode's plotted sizes, then truncated. All speed, memory, and reach figures in this supplement were generated by a single plotting package directly from the benchmark tables, and the accuracy and confidence figures directly from the FoldBench-Lite scoring records.

S1 Co-folding and structure prediction

These models predict the three-dimensional structure of a protein or of a complex of proteins, nucleic acids and small molecules. Speed-ups are for the forward (prediction) call per complex; reach figures show the largest complex that completed on one or two GPUs. Every section except AF2 initial guess also reports accuracy and confidence scores on FoldBench-Lite (p. 16).

S1.1 ESMFold2

ESMFold2 predicts biomolecular complex structures from sequence. A 6B-parameter ESM C language model embeds each chain, a recycling pair trunk builds the pair representation, and a diffusion module samples all-atom coordinates. Confidence heads report pLDDT, pTM, ipTM, and predicted aligned error (PAE). Two checkpoints were released. ESMFold2, covered here, has a 48-layer trunk and a multiple-sequence alignment (MSA) module and was always run with MSAs. The single-sequence ESMFold2-Fast is covered in Section S1.2. We benchmarked ESMFold2 [35, 56] with a July 2026 development version of Biohub `esm` [57] (labeled 3.3.0; not released on PyPI) and the Biohub fork of `transformers` 4.57.6.

Benchmark setup. Inputs are complexes of 200 to 1,400 tokens in steps of 200, with eight different complexes per size. Complexes are folded one per call, with five diffusion samples per call. Each pass runs 14 folds at 200 and 400 tokens (eight at larger sizes), drawn from these eight complexes; its first two folds are warm-ups excluded from the statistic, so 12 (six) are timed. Each size is run in three replicate passes. The headline clock is the per-complex forward call, the wall time of the `fold()` call, which for this model includes featurization, the MSA module, the trunk, diffusion, the confidence heads, and the copy back to the host. A whole-pass wall time is also reported. Default runs out of memory at 1,200 and 1,400 tokens, so speed-ups are reported from 200 to 1,000 tokens. Peak GPU memory is the NVML device peak above idle, from dedicated memory runs. Default enables two of the library’s documented speed settings, a fused kernel backend and no pair chunking. Base (the library as shipped, with reference einsum attention and a pair chunk size of 64) leaves both off and is 1.6–7.5× slower than default, depending on input size.

Sampling settings. Default and every mode used the same sampling settings: 20 recycling loops (21 trunk passes) and a 200-step Karras sampling schedule, of which 134 steps are executed, and 5 diffusion samples per call, with the confidence head run once per sample. The loop count and schedule are the `fold()` call’s defaults in the pinned library release (named in `esmfold2/STOCK.md` in the released repository), and 5 samples replaces its default of 1. Reproductions should set all three explicitly: the checkpoint’s own configuration file stores smaller values (3 loops, 14 sampling steps), which `fold()` overrides, and later library releases changed the call defaults (10 loops and a 100-step schedule executing 68 steps), which gives a different, trunk-dominated workload. The weights are the ESMFold2 Full checkpoint, which includes the MSA module. Inputs are precomputed ColabFold-search MSAs (UniRef30 and the ColabFold environmental database), subsampled to a depth of 1,024 sequences, and no templates; each call folds one complex.

Modes. Exact, Fast, and Big were measured on one GPU, and Big also on two GPUs for the reach analysis. Exact reschedules default’s arithmetic without changing it and is bit-identical to default under deterministic settings, including on the inputs of the accuracy benchmark. Fast adds fused kernels whose results can differ slightly and was checked against default’s own seed-to-seed spread, since inference is stochastic by design. Big trades speed for lower peak memory so that larger inputs fit.

Optimizations. The optimized package leaves the model and its settings unchanged and adds the following:

- **CUDA graphs (Exact and Fast).** The pair trunk, the language-model and MSA encoders, and the diffusion sampler are recorded once as CUDA graphs and replayed without Python overhead. This gives the largest gain at small and medium input sizes.
- **Fused triangle-multiplication kernels (Fast and Big).** Kernels tuned per GPU and input size fuse the pair trunk’s triangle-multiplication updates. This becomes the dominant gain from about 600 tokens, where the trunk’s cost grows fastest.
- **Diffusion on the GPU.** In Exact and Fast, the 134 executed diffusion steps run on the GPU as replayed CUDA graphs. Fast and Big also fuse each diffusion-transformer step and align structures on the GPU.
- **Fused transition, atom-transformer, and MSA kernels.** The pair transition is one fused kernel in every mode (bit-exact in Exact). Fast and Big also fuse the atom transformer into three kernels and four operations of the MSA module.
- **Memory savings.** Fast and Big free pair tensors once they are no longer needed. Big also drops CUDA graphs and their memory pools, runs the confidence head once per sample, streams the language model’s blocks from pinned host memory above 1,500 tokens, and, on two GPUs, splits the pair representation into row blocks, lowering peak memory rather than time.

Results. Forward-call speed-ups are largest for Fast, 4.29–4.44× at every input size, and for Big from 600 tokens (4.32–4.44×). Big is slower at small inputs (2.74× at 200 tokens), and Exact’s gains are smaller (1.52–1.81×). Whole-pass speed-ups are lower

(1.32–3.32× across modes), because each pass also pays fixed costs such as loading weights. Exact uses more peak memory than default at the two smallest sizes (1.14× and 1.03×) and Fast at the smallest (1.13×), because of fixed-size CUDA graph memory pools, but both use less at larger sizes, down to 0.80× and 0.58× at 1,000 tokens. Big uses about as much memory as default at 200 tokens (0.99×) and much less at larger sizes, down to 0.34× at 1,000 tokens. Big raises the largest input that completes from 1,000 tokens for default to 3,000 tokens on one GPU and to 5,000 tokens on two GPUs. The two-GPU value comes from shortened probe runs (one trunk pass; diffusion samples unchanged), checked against a full-settings run at 3,000 tokens whose peak memory matched, so it shows what fits in memory, not a run time.

Accuracy on FoldBench-Lite. On the 216 interfaces of 148 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 56.0% of interfaces under default and for 60.6%, 58.3%, and 56.5% under Exact, Fast, and Big (Figure S4). The only paired changes from default whose 95% confidence intervals exclude zero are a gain of 6.0 points for Exact on antibody–antigen interfaces and a gain of 4.6 points for Exact on all interfaces. Exact is bit-identical to default under deterministic settings (Modes). Confidence scores track default closely (Figure S5): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are at most 0.004 for ipTM, at most 0.003 for pTM, and at most 0.11 points for mean pLDDT.

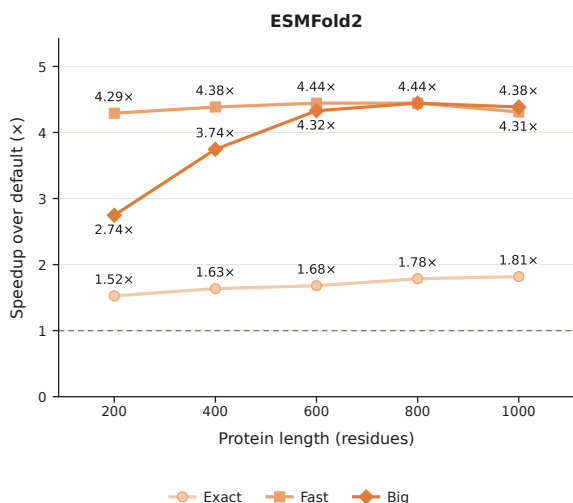


Figure S1. ESMFold2: forward-call speed-up over default. Speed-up on the per-item forward-call clock versus input size, one line per mode (Exact, Fast, Big); the dashed line marks default (1×).

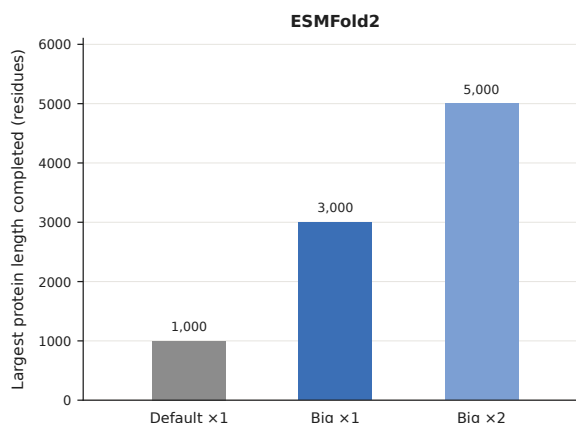


Figure S2. ESMFold2: largest input completed. Largest complex size (tokens) completed by default and Big on one H100 GPU and by Big on two.

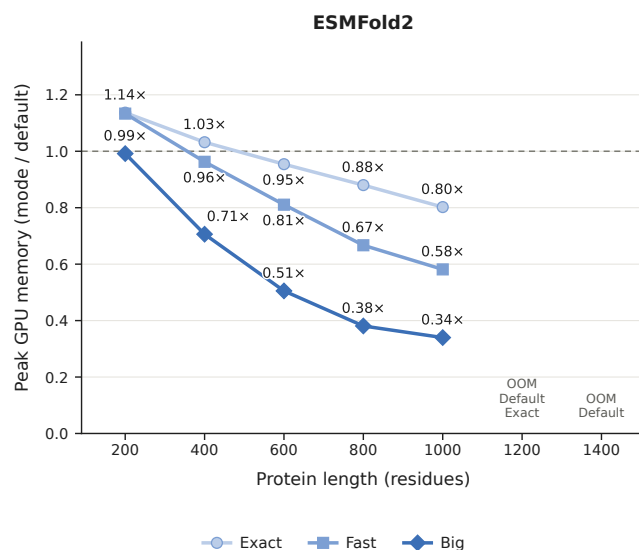


Figure S3. ESMFold2: peak GPU memory relative to default. Peak GPU memory (NVML device peak above idle, from dedicated memory runs) for each mode as a ratio to default, versus input size; the dashed line marks default (1×); OOM, out of memory.

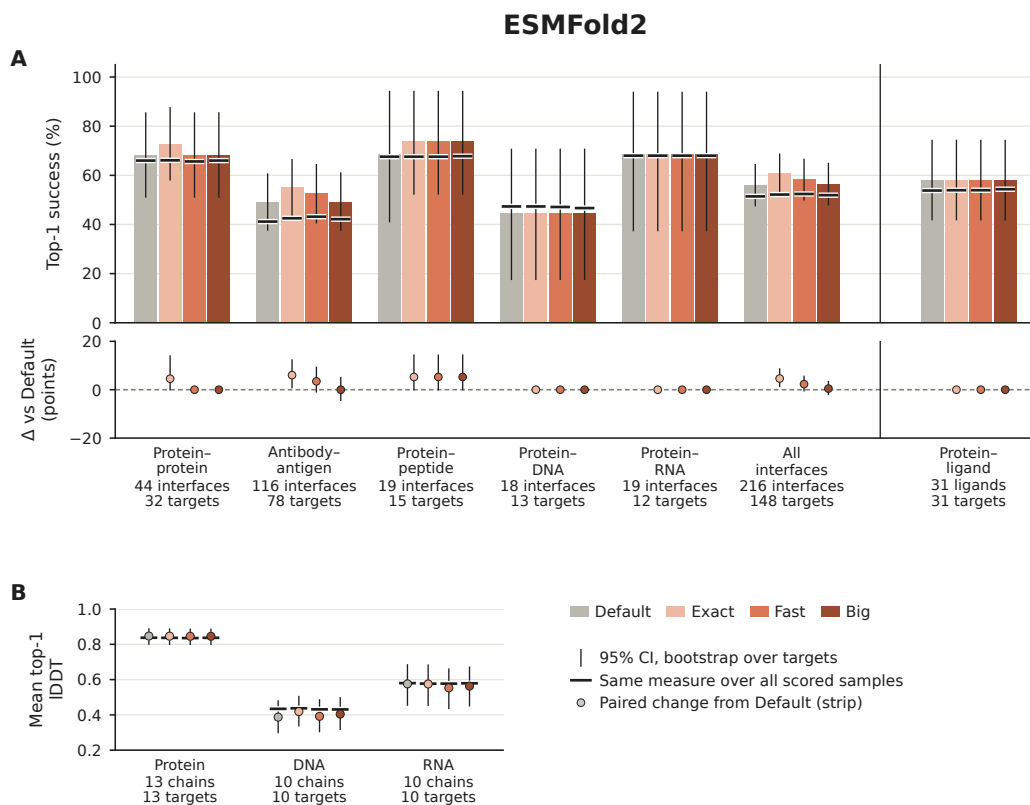
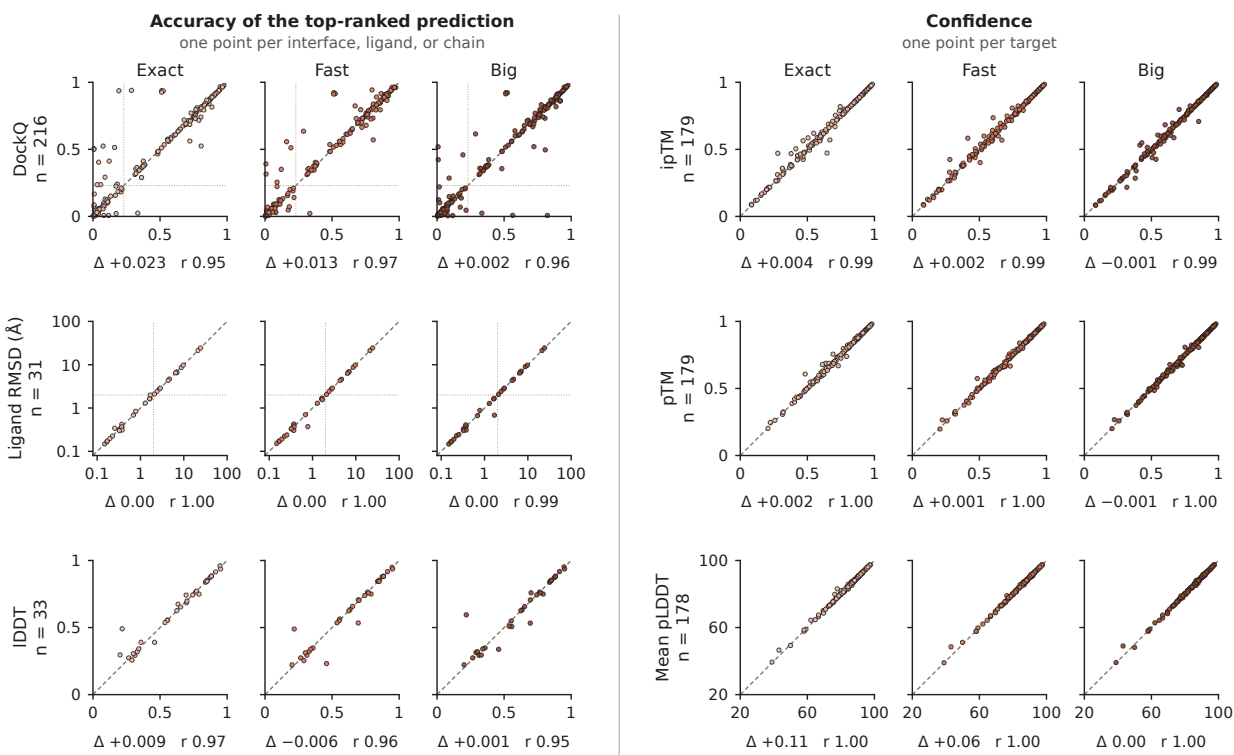


Figure S4. ESMFold2: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

ESMFold2



x = Default, y = mode; dashed line, y = x

Figure S5. ESMFold2: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, y = x. Left, accuracy of the top-ranked prediction (the predictions of Figure S4), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.2 ESMFold2-Fast

ESMFold2-Fast is the single-sequence checkpoint of ESMFold2 (Section S1.1) [35, 56]. It has the same components: a 6B-parameter ESM C language model, a recycling pair trunk, a diffusion module, and confidence heads. It has a 24-layer trunk instead of a 48-layer one and no multiple-sequence alignment (MSA) module. It was benchmarked with the same development version of Biohub `esm` [57] and the same fork of `transformers` as ESMFold2.

Benchmark setup. Inputs, clocks, and memory measurements are those described for ESMFold2 (Section S1.1): complexes of 200 to 1,400 tokens, eight per size, folded one complex per call with five diffusion samples per call. Timing is on the per-complex forward call (the wall time of the `fold()` call, which for this checkpoint includes featurization, the trunk, diffusion, the confidence heads and the copy back to the host); a whole-pass wall time is also reported. Default runs out of memory at 1,200 and 1,400 tokens, so speed-ups are reported from 200 to 1,000 tokens. Default enables two of the library’s documented speed settings, a fused kernel backend and no pair chunking, which base (the library as shipped, with reference einsum attention and a pair chunk size of 64) leaves off. Base is 1.3–6.3× slower than default, depending on input size.

Sampling settings. Default and every mode used the sampling settings of ESMFold2, set explicitly as described in Section S1.1: 21 trunk passes, a 200-step Karras schedule of which 134 steps are executed, and 5 diffusion samples per call. This checkpoint takes single-sequence input, since it has no MSA module, and no templates are used.

Modes. Exact, Fast, and Big were measured on one GPU, and Big also on two GPUs for the reach analysis. Exact reschedules default’s arithmetic without changing it and is bit-identical to default under deterministic settings. Fast adds fused kernels whose results can differ slightly and was checked against default’s own seed-to-seed spread, since inference is stochastic by design. Big trades speed for lower peak memory so that larger inputs fit.

Optimizations. The optimized package and its changes are those described for ESMFold2 (Section S1.1). These are CUDA graphs (Exact and Fast), fused triangle-multiplication kernels (Fast and Big), diffusion on the GPU, a fused pair transition in every mode, a fused atom transformer (Fast and Big), and the memory savings of Fast and Big. Two differences follow from the architecture. This checkpoint has no MSA module, so the MSA encoder graph and the MSA kernels do not apply. For inputs of up to 512 tokens, Exact and Fast also record a whole recycle of the trunk as one CUDA graph.

Results. Forward-call speed-ups are largest for Fast, 4.92–5.49× at every input size, and for Big from 600 tokens (5.26–5.34×). Big is slower at small inputs (2.64× at 200 tokens), and Exact’s gains are smaller (1.46–1.72×). Whole-pass speed-ups are lower (1.26–3.63× across modes), because each pass also pays fixed costs such as loading weights. Exact uses more peak memory than default at the two smallest sizes (1.14× and 1.02×) and Fast at the smallest (1.13×), because of fixed-size CUDA graph memory pools. Both use less at larger sizes, down to 0.81× and 0.56× at 1,000 tokens. Big uses about as much memory as default at 200 tokens (1.00×) and much less at larger sizes, down to 0.31× at 1,000 tokens. Big raises the largest input that completes from 1,000 tokens for default to 3,000 tokens on one GPU and to 6,000 tokens on two GPUs. The two-GPU value comes from shortened probe runs (one trunk pass; diffusion samples unchanged), checked against a full-settings run at 3,000 tokens whose peak memory matched, so it shows what fits in memory, not a run time.

Accuracy on FoldBench-Lite. On the 218 interfaces of 149 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 54.1% of interfaces under default and for 54.6%, 57.3%, and 54.1% under Exact, Fast, and Big (Figure S9). The only paired change from default whose 95% confidence interval excludes zero is a gain of 7.6 points for Fast on antibody–antigen interfaces. Confidence scores track default closely (Figure S10): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are at most 0.003 for ipTM, at most 0.002 for pTM, and at most 0.04 points for mean pLDDT.

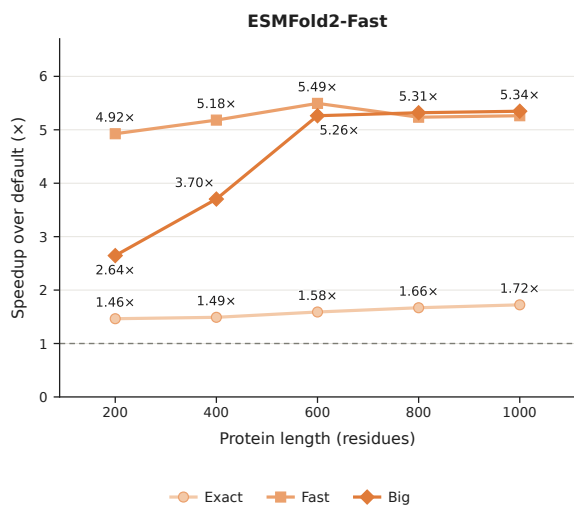


Figure S6. ESMFold2-Fast: forward-call speed-up over default. Speed-up on the per-item forward-call clock versus input size, one line per mode (Exact, Fast, Big); the dashed line marks default (1x).

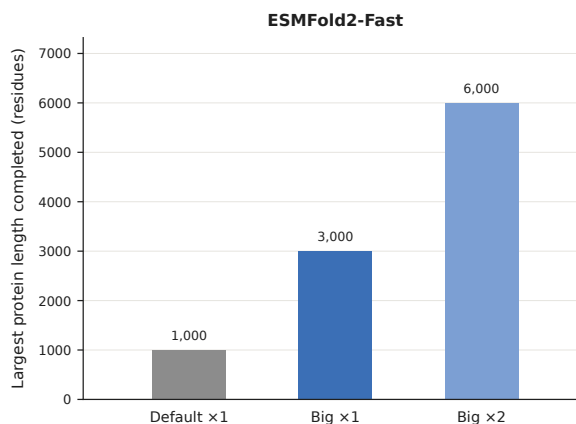


Figure S7. ESMFold2-Fast: largest input completed. Largest complex size (tokens) completed by default and Big on one H100 GPU and by Big on two.

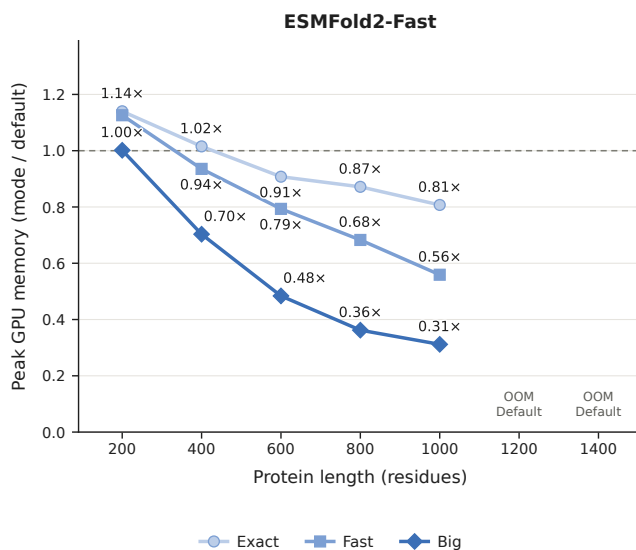


Figure S8. ESMFold2-Fast: peak GPU memory relative to default. Peak GPU memory (NVML device peak above idle, from dedicated memory runs) for each mode as a ratio to default, versus input size; the dashed line marks default (1x); OOM, out of memory.

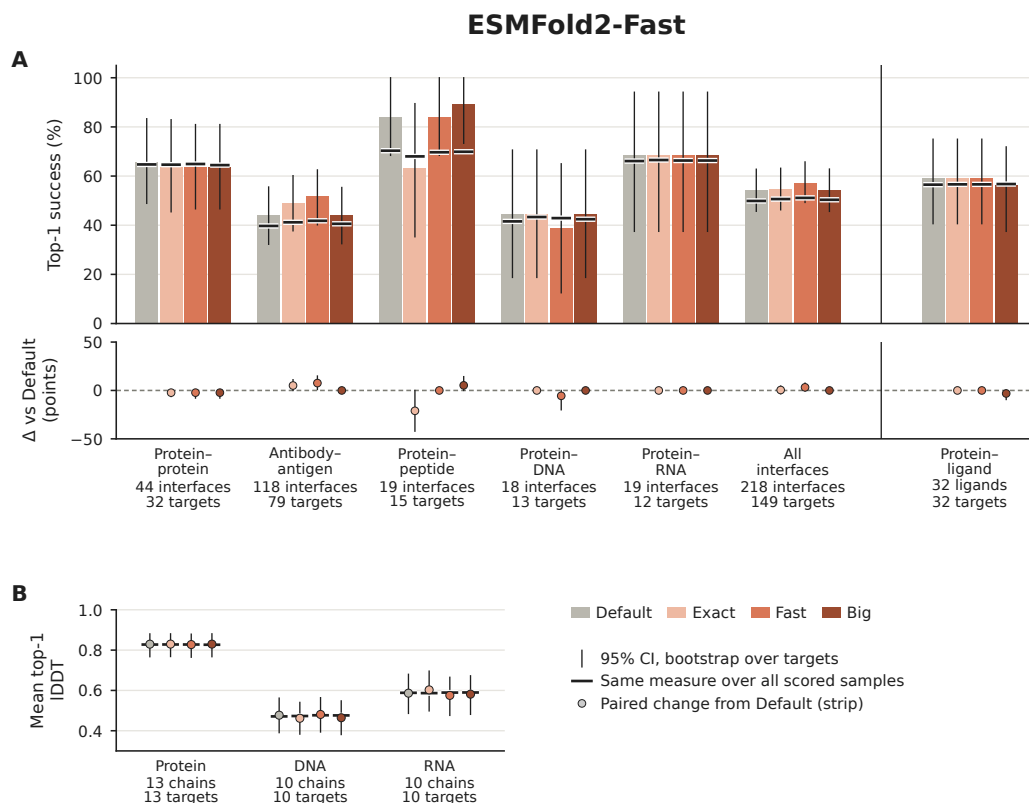


Figure S9. ESMFold2-Fast: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

ESMFold2-Fast

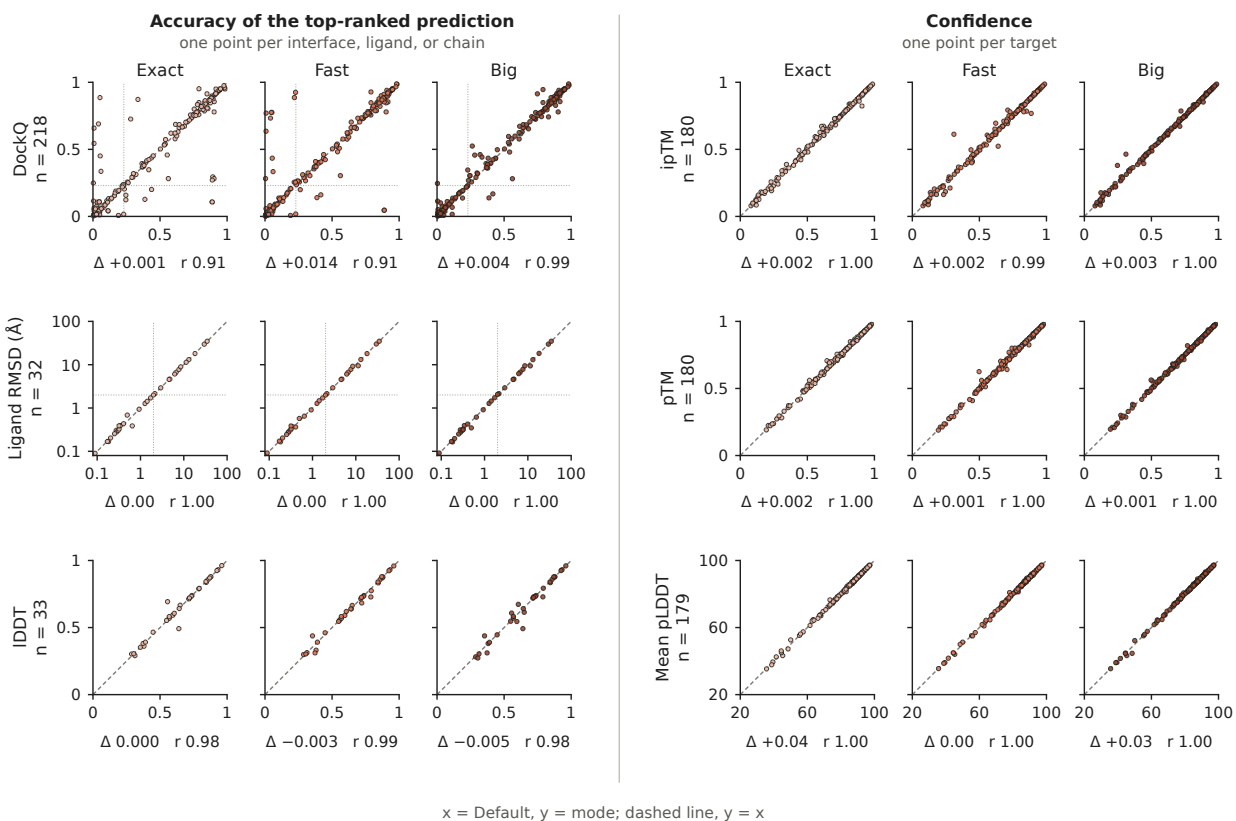


Figure S10. ESMFold2-Fast: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S9), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 $\text{\AA}$). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in $\text{\AA}$); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.3 OpenFold3-p2

OpenFold3-p2 is an open AlphaFold3-style all-atom structure predictor of proteins, nucleic acids, and ligands, with an MSA module and template embedder feeding a recycled Pairformer trunk, a diffusion sampler that draws multiple candidate structures, and confidence heads that score them. Benchmarks pin the upstream `openfold3` 0.4.1 PyPI release with the preview-2 model weights [9].

Benchmark setup. Timing used seven input sizes from 200 to 1,400 tokens (steps of 200), each with two source protein complexes using precomputed MSAs and no templates, folded one query per call. The model has no batch axis. All configurations, default included, used the sampling settings below. Speed-ups may differ at upstream’s defaults. Three replicate runs per size were made on one H100 80 GB GPU, each bracketing every mode between two default passes and discarding the first, warm-up item of every pass. Exact was timed in three separate runs of the same design. The headline clock is the per-item forward-call time from the same internal timer on every mode, excluding host-side featurization and output writing. A companion whole-pass wall-clock time is also reported (Results). Peak GPU memory is the NVML device high-water mark above an idle baseline, from dedicated memory-only runs. For this model, default adds two settings available through upstream’s own interface – `bfloat16-mixed` precision and `cuEquivariance` triangle kernels – on top of the bare command (base, which was not timed separately). Base otherwise runs in `float32` with these kernels off, so speed-ups reflect the fastest configuration reachable through upstream’s interface rather than its weakest setting.

Sampling settings. Default and every mode used the same sampling settings: 10 recycles (11 trunk passes; upstream default 3 recycles), 200 diffusion steps, and 5 diffusion samples per query, with one query per call. The model is OpenFold3 0.4.1 with the preview-2 weights. Inputs are precomputed MSAs, with no MSA server and no templates.

Modes. Exact, Fast, and Big are all implemented for this model. Exact is verified bit-identical to default under a deterministic recipe. This was confirmed on all 435 predictions of the identity-check set, and again with the kernels used for the Exact timings. Fast drops the bit-identical requirement for additional kernels. It passes a seed-spread accuracy guard requiring each prediction’s $C\alpha$ -RMSD to default’s seed-42 structure to fall inside the spread default itself shows across seeds. Big keeps Fast’s numerics, passes the same guard, and trades some of Fast’s memory-costly components for memory-saving computation at larger sizes. Big on two GPUs additionally splits the pair representation by row across both GPUs to reach still larger inputs.

Optimizations. The optimized package attaches the following changes when OpenFold3’s model modules are imported, without editing upstream code:

- **CUDA-graph replay of the diffusion step.** One denoising step is captured once per input shape and replayed for the remaining steps and samples, removing repeated launch overhead. This is the largest single gain at small and mid input sizes (Exact up to 512 tokens; Fast uncapped).
- **Fused pair-block kernels.** Triangle attention, triangle multiplication, and pair transition become fused kernel chains in every pair stack, including the MSA module and template stacks, in Fast and Big. Exact uses equivalent kernels checked bit-for-bit against default’s output first.
- **Step-invariant hoists in the rollout.** Quantities unchanged between diffusion steps – the conditioned pair representation, atom-path reference embeddings, and a key-mask bias – are computed once per rollout instead of every step, in Exact and Fast.
- **Fused diffusion-transformer block, reduced precision (Fast and Big).** The token-level diffusion-transformer block runs as one fused schedule under `bfloat16` autocast and atom attention as a fused windowed kernel, trading bounded numerical drift for speed.
- **Row-block streaming for large inputs.** Above roughly 1,400 tokens, Big streams pair-representation and confidence computations in row blocks against a pinned host-memory snapshot so a second copy of the largest tensors is never resident in GPU memory. Below that size, Big behaves like Fast, which is why the two track each other closely across the tested ladder.

Results. For Fast and Big, the forward-call speed-up is largest at 200 tokens (8.77 $\times$ and 8.54 $\times$) and levels off at 4.3–4.7 $\times$ from 600 tokens, with a geometric mean of 5.10 $\times$ for both across the ladder. Exact gives a smaller gain (1.43–1.94 $\times$). The whole-pass wall-clock speed-up (which also counts model loading, featurization, and output writing) is lower than the forward-call speed-up for Fast (2.46–3.62 $\times$) and Big (3.68–5.11 $\times$), and higher for Exact (1.91–2.18 $\times$), as its start-up and host-side changes also shorten time outside the forward call. All three modes can use more peak GPU memory than default at small sizes. Fast and Big use 1.5 $\times$ and 1.2 $\times$ at 200 and 400 tokens, because of resident graph pools and cached tensors. Exact uses up to 2.6 $\times$ through 600 tokens, mostly because its bit-exact triangle-attention kernel needs more memory (peaks of 23.5 and 34.4 GB at 400 and 600 tokens). From 800 to 1,200 tokens they use 0.37–0.70 $\times$ of default’s peak, and at 1,400 tokens slightly less than default (0.92–0.97 $\times$). On reach, default completes inputs up to 3,000 tokens on one GPU. Big extends this to 6,000 tokens on one GPU and to 17,000 tokens on two (from shortened probe runs; possibly one 1,000-token step too high at full settings). Exact and Fast were not part of the reach search.

Accuracy on FoldBench-Lite. On the 251 interfaces of 151 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 56.6% of interfaces under default and for 55.8%, 53.4%, and 54.2% under Exact, Fast, and Big (Figure S14). The only paired changes from default whose 95% confidence intervals exclude zero are a change of -0.005 in mean protein-monomer IDDT under Fast and Big. Confidence scores track default closely (Figure S15): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are at most 0.004 for ipTM, at most 0.002 for pTM, and at most 0.06 points for mean pLDDT.

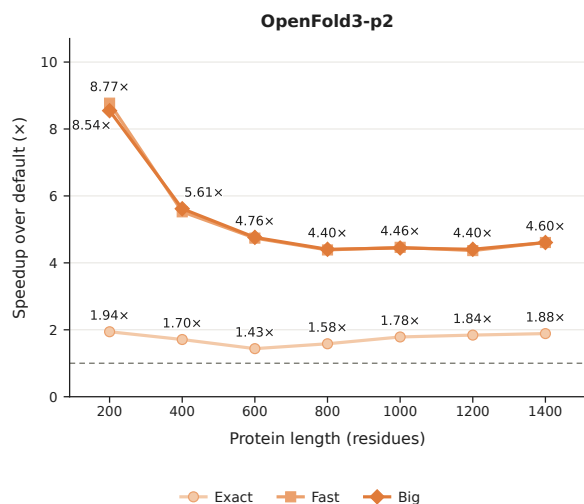


Figure S11. OpenFold3-p2: forward-call speed-up over default versus input size on H100. Each line (Exact, Fast, Big) shows the speed-up on the per-item forward-call clock at each input size on the 200–1,400-token ladder; the dashed line marks default (1×). Fast and Big nearly coincide, so the Fast line is largely hidden under Big.

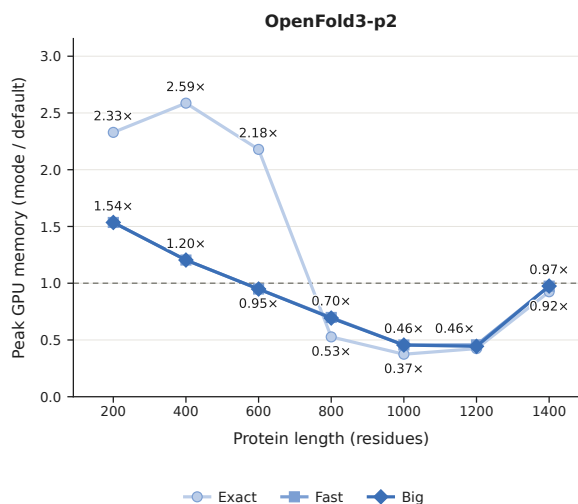


Figure S12. OpenFold3-p2: peak GPU memory relative to default versus input size on H100. Each line (Exact, Fast, Big) shows the NVML device peak memory above idle, measured in dedicated memory-only runs, as a ratio to default’s peak at the same input size; the dashed line marks default (1×). Fast and Big nearly coincide, so the Fast line is largely hidden under Big.

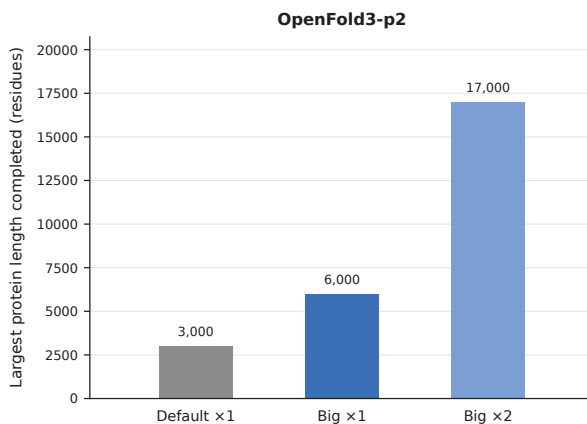


Figure S13. OpenFold3-p2: largest input complex completed on H100. Bars show the largest input size (tokens) that completed for default on one GPU, Big on one GPU, and Big on two GPUs.

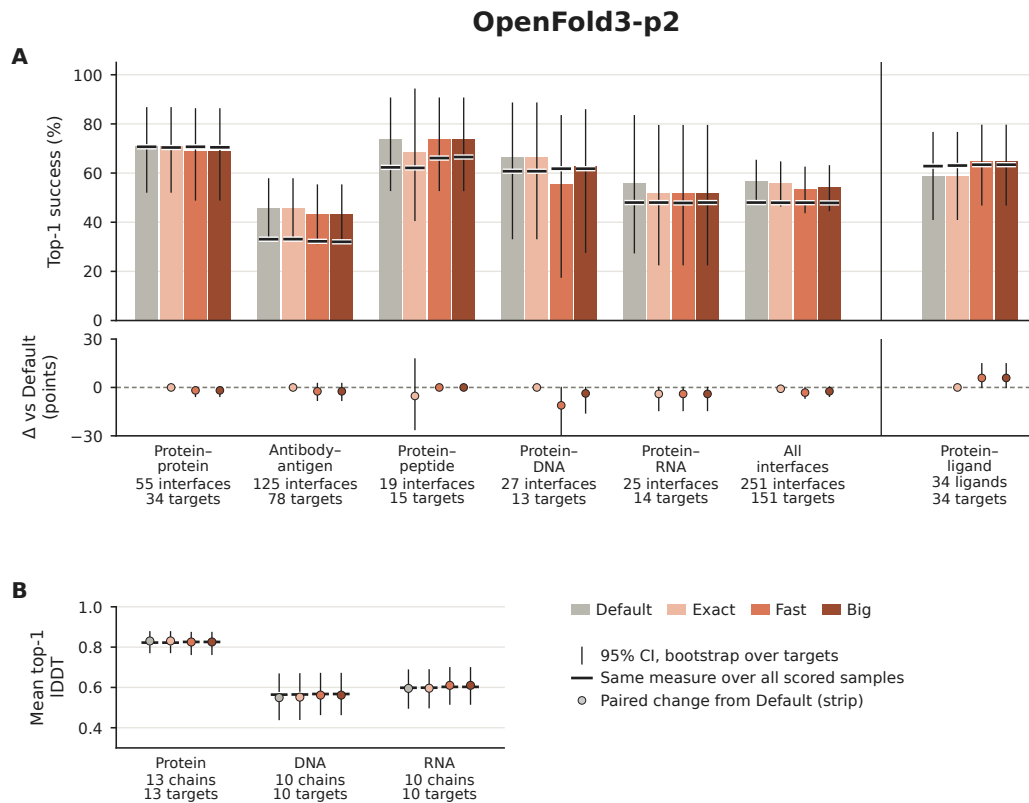


Figure S14. OpenFold3-p2: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

OpenFold3-p2

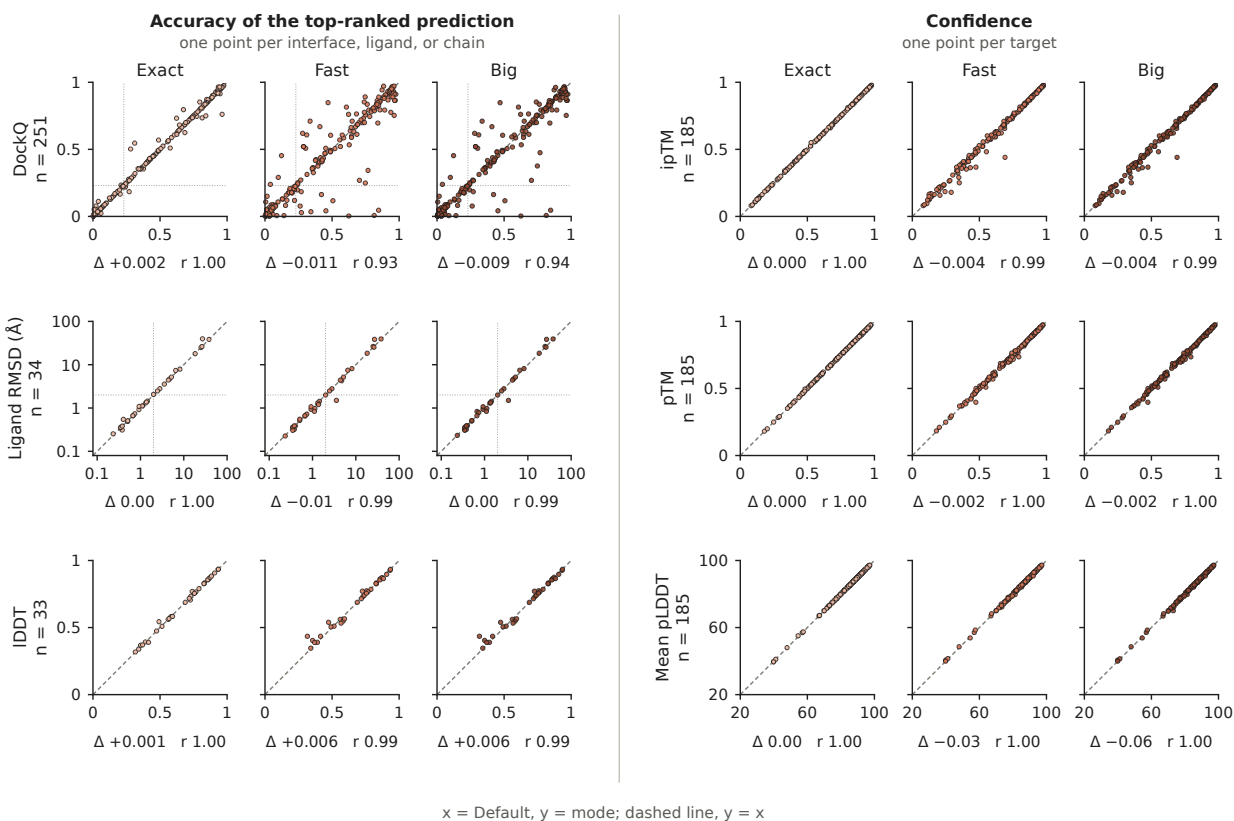


Figure S15. OpenFold3-p2: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S14), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.4 OpenBind-0

OpenBind-0 is the AlphaFold3-class co-folding checkpoint of `3-ob-2025-06-30-174k.pt`, released by OpenBind [58] and benchmarked on OpenFold3, a biomolecular structure predictor. The optimized package targets upstream openfold3 0.5.0 (tag v0.5.0) [9].

Benchmark setup. The benchmark predicts structures for complexes of increasing size, from 200 to 1,400 input tokens, one complex per forward call, on an H100 80 GB GPU. It runs several replicate passes, with an untimed warm-up pass excluded from timing. The headline speed-up is the ratio of forward-call time per item – the model’s own structure-prediction call, excluding model loading, featurization, warm-up, and output writing – for default and for each mode in the same run. A companion whole-pass wall-clock time, which additionally includes process launch and output writing, was also recorded (not shown). The whole-pass speed-ups are smaller than the forward call’s for Fast and Big, though slightly larger on average for Exact, because it carries one-time costs the optimized package does not shrink. Peak GPU memory is the device’s NVML high-water mark (including the CUDA context and allocator cache) above an idle baseline. The framework allocator’s own peak was recorded for context (not shown), never substituted for the device meter. Default here is upstream OpenFold3 run in the fastest configuration reachable through its own flags – cuEquivariance triangle kernels enabled, DeepSpeed attention disabled, mixed bfloat16 precision, and a fixed chunk size – rather than its out-of-the-box defaults. This ensures the comparison is against the strongest configuration of the unmodified package. The package is on average slower than default when run exactly as shipped (base: float32 precision with TensorFloat-32 matrix multiplication, its own chunk-size tuner, no cuEquivariance kernels), at about 0.68× default’s speed (geometric mean across the sizes tested), though it is faster than default at the smallest size tested (indicative only; timing drift exceeded the 5% limit).

Sampling settings. Default and every mode used the same sampling settings: 10 recycles (11 trunk passes; upstream default 3 recycles), 200 diffusion steps, and 5 diffusion samples per call. The model is OpenFold3 0.5.0 with its default checkpoint, OpenBind-0 (`3-ob-2025-06-30-174k.pt`). Inputs are precomputed MSAs, with no MSA server; templates are enabled (`--use-templates true`) but none are supplied, and each call predicts one complex.

Modes. Exact, Fast, and Big all exist for this model. Exact limits changes to scheduling, placement, caching, and arithmetic proven equal to default’s own output in-process. Under a deterministic setting, its outputs were verified byte-identical to default. Fast keeps Exact’s placement and caching but replaces the bit-identical kernels with fused and bfloat16 kernels, with an accuracy check that keeps results within default’s own seed-to-seed variation. Big applies Fast’s optimizations below a size gate and switches to memory-streaming optimizations above it, also verified to stay within default’s seed-to-seed variation. On two GPUs, Big additionally shards the computation across devices and matches the single-GPU run within tolerance rather than bit for bit.

Optimizations. The largest gains come from changes to the diffusion sampling roll-out and the trunk’s pair-representation stack, with smaller optimizations hiding I/O and process start-up cost.

- **CUDA-graph capture and replay of the diffusion step.** One denoising step is recorded once per input shape – a CUDA graph avoids reissuing each GPU instruction separately – and replayed for the remaining steps and samples; in Exact, the largest single optimization for inputs up to 512 tokens.
- **Fused, reduced-precision diffusion module (Fast and Big).** From 600 input tokens up, the sampling roll-out runs in bfloat16 (a lower-precision numeric format) with fused transformer blocks – several small GPU operations combined into one – while keeping attention, coordinate updates and sampler arithmetic in float32; at 200 and 400 tokens it stays in float32 under Fast and Big.
- **Fused pair-stack kernels (Fast and Big).** Triangle multiplication, triangle attention, and the pair transition are replaced with fused implementations in every trunk block, the main source of speed-up at medium and large input sizes. Exact swaps in faster kernels only where proven bit-identical.
- **One-time caching and overlapped output writing.** Pair and atom tensors that stay constant across the diffusion roll-out are computed once and reused, and writing one item’s output overlaps with running the next item, hiding I/O behind compute.
- **Memory-streaming pair stack for large inputs (Big).** Above a size gate, the pair stack streams from a pinned host-memory snapshot (system memory reserved for fast transfer to the GPU) and confidence heads are evaluated in row blocks instead of all at once, trading the caching optimizations above for lower peak memory. On two GPUs the pair representation is also split across devices.

Results. Fast and Big give large, fairly flat speed-ups across the tested input sizes, with geometric-mean forward-call speed-ups of 4.93× (Fast) and 4.94× (Big) over default. The two coincide because the tested inputs sit below Big’s memory-saving size gate. Exact gives smaller, less consistent gains, and in one case runs slower than default on the forward-call clock. Peak GPU memory shows an inverse pattern at small input sizes. Exact (at 200 and 400 tokens) and Fast and Big (at 200 tokens only) can exceed default’s

peak there because of resident caching and graph-pool buffers, while at larger sizes all three modes fall below default's peak. On one GPU, default completes structures up to 3,000 tokens before running out of memory, whereas Big extends this to 6,000 tokens on one GPU and 17,000 tokens on two GPUs. Big's one-GPU value was measured at full settings; the two-GPU value comes from shortened probe runs (one recycle; diffusion samples unchanged), checked against a full-settings run at 4,000 tokens whose peak memory matched, so it shows what fits in memory, not a run time.

Accuracy on FoldBench-Lite. On the 251 interfaces of 152 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 58.2% of interfaces under default and for 59.0%, 56.6%, and 56.6% under Exact, Fast, and Big (Figure S19). The only paired changes from default whose 95% confidence intervals exclude zero are a change of +0.001 in mean protein-monomer IDDT under Fast and Big and a change of -0.013 in mean RNA-monomer IDDT under Fast and Big. Confidence scores track default closely (Figure S20): across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r = 1.00$ (to two decimals), and the mean paired differences are below 0.0005 for ipTM and pTM and at most 0.02 points for mean pLDDT.

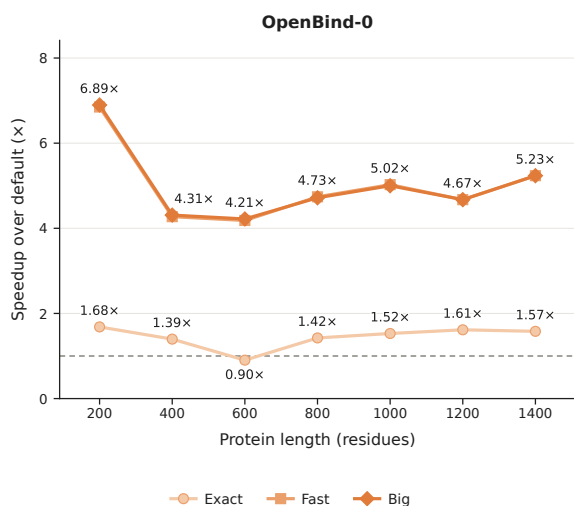


Figure S16. OpenBind-0: forward-call speed-up over default on H100. Speed-up over default (default's forward-call time divided by the mode's, both per item) versus input size in tokens, one line per mode (Exact, Fast, Big); the dashed line marks default (1x). Fast and Big coincide (they differ by at most 0.05x), so the Fast line is hidden under Big; the labels at 800 and 1,000 tokens are Fast's (Big: 4.71x and 5.00x).

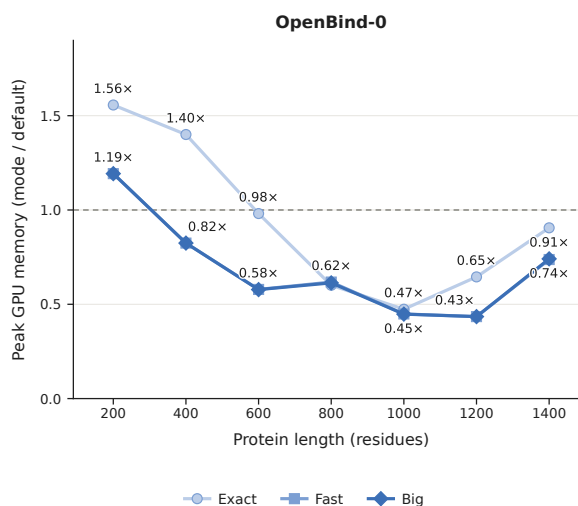


Figure S17. OpenBind-0: peak GPU memory relative to default on H100. Peak device memory (NVML peak above idle) for each mode divided by default's peak in the same run, versus input size in tokens, one line per mode (Exact, Fast, Big); the dashed line marks default (1x). Fast and Big coincide, so the Fast line is hidden under Big; the single label at 800 tokens is Fast's (Exact 0.60x, Big 0.61x).

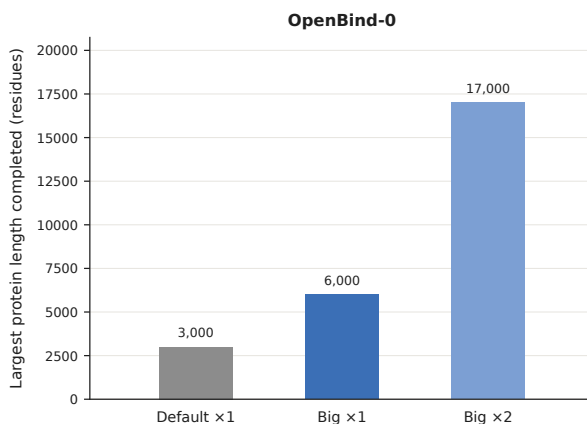


Figure S18. OpenBind-0: largest completed input size on H100. Largest input complex, in tokens, that completes, one bar per configuration (Default x1, Big x1, Big on two GPUs); larger bars indicate a larger complex reachable before running out of GPU memory.

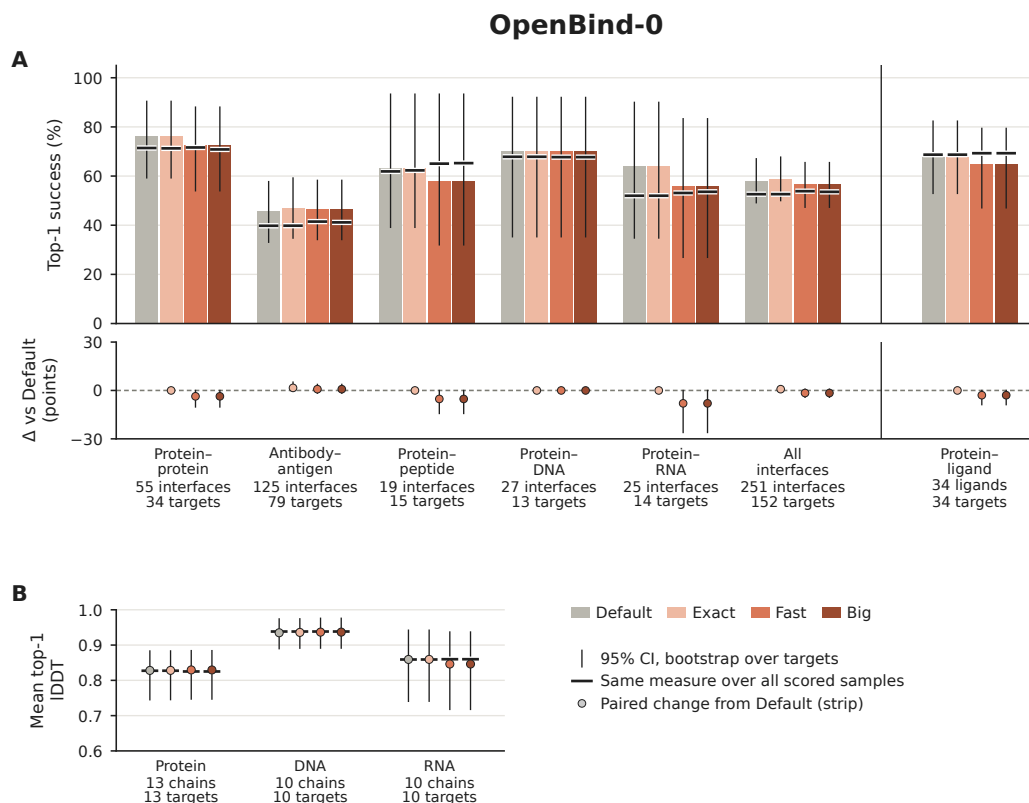


Figure S19. OpenBind-0: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

OpenBind-0

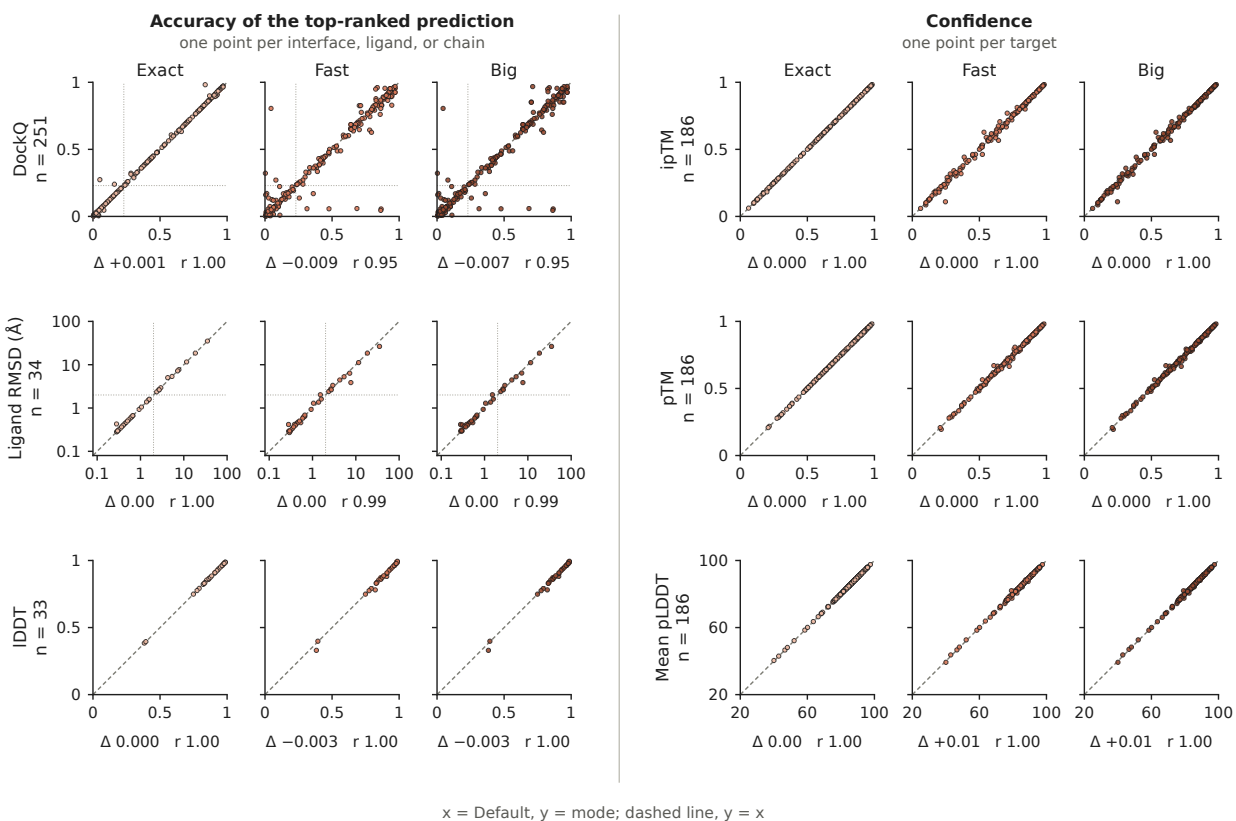


Figure S20. OpenBind-0: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S19), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.5 AlphaFold3 (JAX)

AlphaFold3 (JAX) predicts three-dimensional structures of multi-chain and protein–ligand complexes (“cofolding”) from sequence and pre-computed multiple-sequence alignments, run through a community JAX/Haiku fork [36] (tag v3.1.4) of the AlphaFold3 inference code [2], using the public OpenFold3-preview2 checkpoint [9] instead of AlphaFold3’s own parameters.

Benchmark setup. Speed is measured on seven input sizes from 200 to 1,400 tokens, each built from two source complexes rendered as sequence-substituted copies. A separate set of sizes from 3,000 to 7,000 tokens (1,000-token steps) is used only to find the largest complex that default (on one GPU) and Big (on one and two GPUs) can fold. AlphaFold3 has no cross-complex batching, so each call processes one complex, with the model’s five diffusion samples drawn inside that call. Default and each mode are timed over three replicate passes per input size on one H100 80 GB GPU. The timing median excludes the two warm-up items of each pass; the first warm-up item includes compilation. The reported speed-up times only the jitted forward call itself – the trunk passes, diffusion sampling, and confidence head – excluding host-side featurization and output writing. The per-item wall-clock time (Results) includes that work, as does the whole-pass wall-clock time (with start-up and compilation; geometric-mean speed-ups 1.53×, 2.80×, and 2.82× for Exact, Fast, and Big). Peak GPU memory is the NVML device high-water mark above idle from a dedicated, untimed pass with the allocator in growth mode, so the reading tracks actual use rather than a reserved pool. Because the meter still moves in coarse allocator-region steps, ratios just above 1× (Fast and Big at 1.01× at 400 and 600 tokens) can reflect allocator granularity rather than a genuine difference in memory used. Default here means the upstream package run at its own shipped defaults (which already include triton-based flash attention and bfloat16) plus one added flag, a padding-bucket list matched to the sizes the optimized modes compile, so every configuration compiles and runs on the same padded input. This keeps the comparison to the fastest correct configuration of the unmodified package rather than crediting the optimized modes with an easier, coarser padding choice. Where base’s own bucket list pads a size further than default’s list, base’s forward call is 11–43% slower than default; base’s forward call runs at 0.84× default’s speed, averaged over the ladder (geometric mean).

Sampling settings. Default and every mode used the same sampling settings, which are upstream’s defaults: 10 recycles (11 trunk passes), 200 diffusion steps, and 5 diffusion samples per call. Inputs are precomputed unpaired MSAs supplied inline, with no templates. The weights are the OpenFold3 preview-2 checkpoint (`of3-p2-155k.pt`) ported to the AlphaFold3 parameter layout; no AlphaFold3 parameters were used.

Modes. Exact, Fast, and Big are all present, with Big also run on two GPUs. Exact preserves default’s exact arithmetic and gave output bit-identical to default under deterministic settings in two identity checks. In the second, 297 output files from 27 input cases were set aside because default itself does not reproduce them across containers, so identity was confirmed on the remaining outputs. Fast substitutes reduced-precision and approximate kernels; its accuracy guard is a comparison against default’s own seed-to-seed spread, and its outputs stayed within that spread. Big runs Fast’s program, with the same kernels, up to 1,408 padded tokens – the two are identical at every input size on the main ladder – and only diverges above that size, where it row-shards several computations to cap memory. Big on one and on two GPUs passed dedicated seed-spread identity checks against default, which include inputs above 1,408 tokens and a templated input set.

Optimizations. The gains come from a mix of numerical kernel fusion, host/GPU overlap, and memory-aware scheduling for large inputs, implemented in the optimized package that provides the Exact, Fast, and Big modes.

- **Fused pairformer and diffusion kernels.** Custom fused GPU kernels replace separate triangle-attention, triangle-multiplication, and transition steps in the trunk, and separate attention and matrix-multiply steps in the diffusion sampler, with combined operations, some using lower-precision (bfloat16) arithmetic. This is the main source of the forward-call speed-up in Fast and Big.
- **Host-side featurization/writer overlap.** Upcoming items are featurized on the CPU in three worker processes while the GPU runs the current item, and a writer thread writes each item’s outputs behind the next item’s inference, instead of doing this work serially around each call. This overlap operates in Exact, Fast, and Big alike and is why their per-item wall-clock gains exceed their forward-call gains.
- **Fused triangle-multiplication kernel.** A single fused kernel avoids storing an extra transposed intermediate tensor in triangle multiplication, lowering peak memory even where no lower-precision arithmetic is used.
- **Row-sharded compute for large inputs (Big).** Above 1,408 padded tokens, pair-transition, triangle-multiplication, and diffusion-conditioning computations are processed in row chunks instead of all at once, capping peak memory so substantially larger complexes can be folded on one GPU.
- **Hoisted diffusion conditioning and logits (Fast and Big).** Per-block pair conditioning and logit projections that do not change across the 200 denoising steps are computed once per call and reused, instead of being recomputed at every step of the diffusion sampler. Big drops the hoisted logit buffer above 1,408 padded tokens.

Results. Across the speed ladder, Fast reaches a geometric-mean forward-call speed-up of 2.46 $\times$ over default, and Big matches Fast almost exactly at every input size tested. Exact’s forward call stays near parity with default since it changes no numerics. Overlapping featurization and writing with inference still gives it a per-item wall-clock gain of about 1.5 $\times$ (geometric mean), and the same overlap lifts Fast’s and Big’s per-item wall-clock gains to about 3.6 $\times$, above their forward-call gains alone. Peak memory follows a similar pattern. At the longest input sizes the optimized modes use about 0.64 $\times$ default’s memory, a reduction traced to the fused triangle-multiplication kernel rather than to lower-precision arithmetic, since Exact shows the same drop. On the reach search, Big folds complexes up to 6,000 tokens on one GPU, well beyond default’s limit. Running Big across two GPUs does not extend that reach further, since the per-device compilation workspace is not divided between the cards. Big’s one-GPU value was measured at full settings; its two-GPU runs above 4,000 tokens were shortened probe runs (one recycle; diffusion samples unchanged), checked against a full-settings run at 4,000 tokens whose peak memory matched.

Accuracy on FoldBench-Lite. On the 252 interfaces of 153 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 53.2% of interfaces under default and for 54.4%, 53.6%, and 55.2% under Exact, Fast, and Big (Figure S24). The only paired change from default whose 95% confidence interval excludes zero is a change of +0.006 in mean protein-monomer IDDT under Fast. Confidence scores track default closely (Figure S25): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r = 1.00$ (to two decimals), and the mean paired differences are at most 0.001 for ipTM, below 0.0005 for pTM, and at most 0.08 points for mean pLDDT.

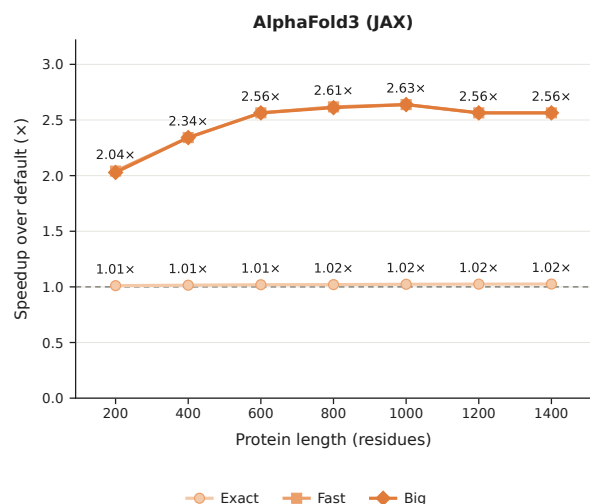


Figure S21. AlphaFold3 (JAX): forward-call speed-up over default. Each line shows the per-item forward-call speed-up (default time divided by mode time) for Exact, Fast, and Big versus input size in tokens on the 200–1,400-token ladder; the dashed line marks default (1 $\times$). Fast and Big coincide on this input-size ladder, so the Fast line is hidden under Big.

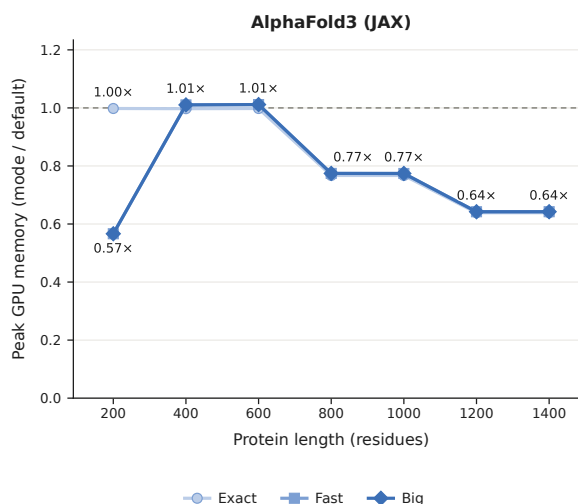


Figure S22. AlphaFold3 (JAX): peak GPU memory relative to default. Lines show each mode’s NVML device peak memory above idle, as a ratio to default’s peak at the same input size in tokens, from dedicated memory runs; the dashed line marks default (1 $\times$). Fast and Big coincide, so the Fast line is hidden under Big.

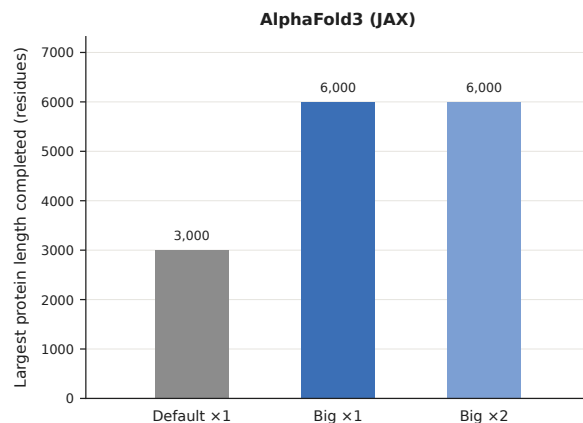


Figure S23. AlphaFold3 (JAX): largest complex folded per configuration. Bars show the largest input size, in tokens, that completed without running out of memory for default on one GPU, Big on one GPU, and Big on two GPUs, from the size-spaced reach search on H100.

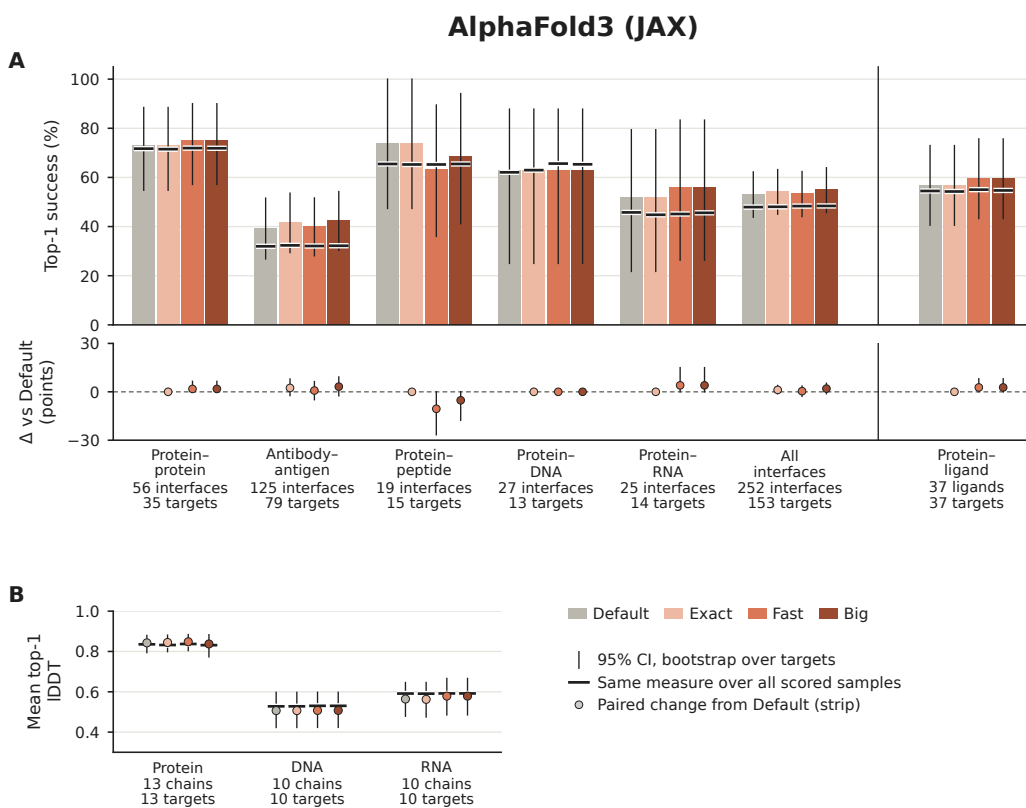


Figure S24. AlphaFold3 (JAX): accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

AlphaFold3 (JAX)

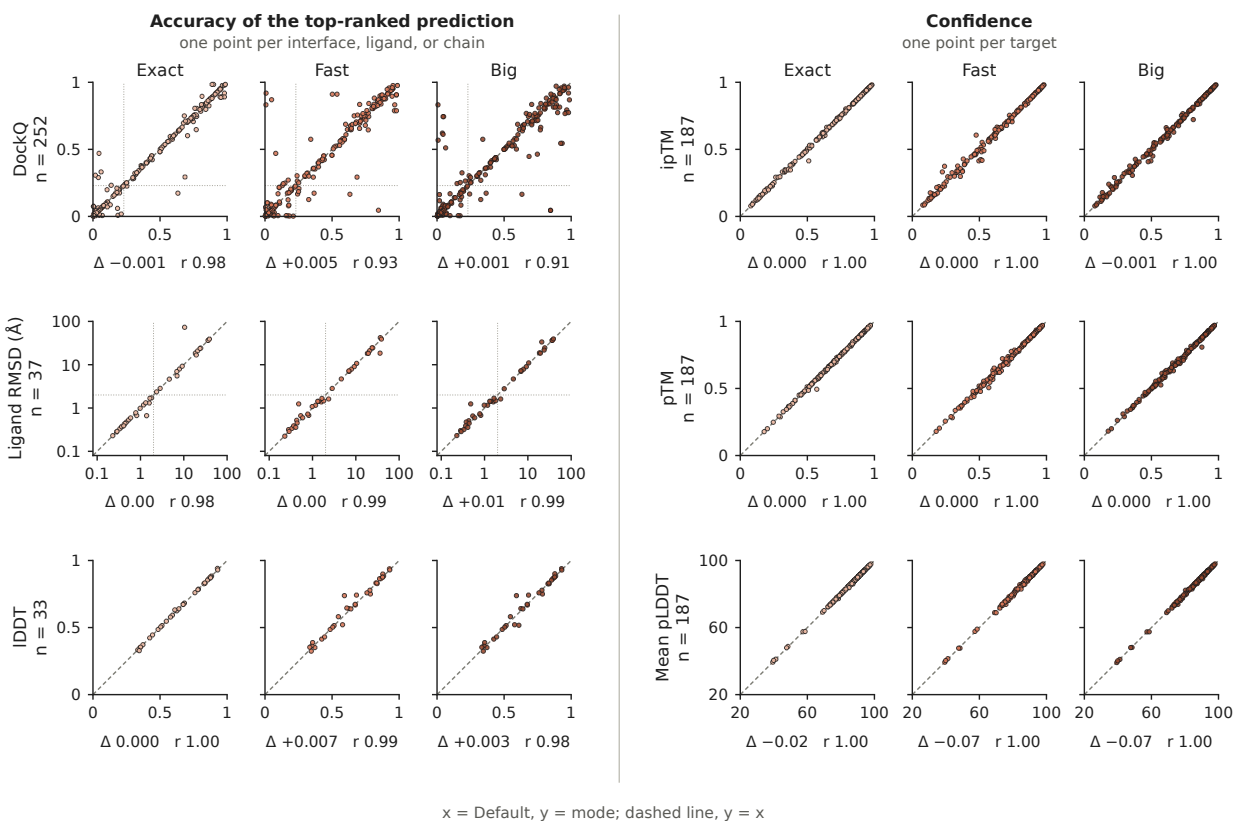


Figure S25. AlphaFold3 (JAX): accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S24), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.6 AlphaFold3 (PyTorch)

AlphaFold3 (PyTorch) refers to xfold [37], a PyTorch re-implementation of the AlphaFold3 network [2], benchmarked here on the converted OpenFold3-preview2 checkpoint rather than on AlphaFold3’s own parameters.

Benchmark setup. The task is single-complex structure prediction: one fold input per call, run with the sampling settings below. It is evaluated across seven PDB-derived input sizes from 200 to 1,400 tokens with 8 content-distinct items per size, on an H100 80 GB GPU. Each input size was measured over 3 replicate runs. Each run excluded an untimed warm-up item and included one pass each of Exact, Fast, and Big bracketed by two default passes (12 timed items per pass at 200 and 400 tokens, 6 above; 158 of 162 per mode completed). The headline speed-up is on the per-item forward-call time – the CUDA-synchronized model call covering the trunk, diffusion sampler, and confidence head, excluding featurization and output writing – with whole-pass wall time (process start to exit) reported alongside as context. Peak GPU memory is the NVML device high-water mark above idle, measured from outside the running process during a dedicated memory-only pass, so it includes model load and compile/autotune workspace, not only tensor activations. Base (xfold exactly as shipped) cannot load the OpenFold3-preview2 weight layout at all. Default therefore runs the optimized package with every acceleration mode switched off: base’s own shipped kernels, plus the minimal loading fix these weights require and the restoration of AlphaFold3 behaviors the re-implementation had dropped (among them, 11 trunk passes for 10 recycles and confidence scoring on all five samples). No separately runnable unmodified baseline exists, so every mode is compared against this closest-runnable form of the shipped code, in the same run and on the same inputs.

Sampling settings. Default and every mode used the same sampling settings, which are the package’s defaults: 10 recycles (11 trunk passes), 200 diffusion steps, and 5 diffusion samples per call, all scored by the confidence head. Inputs are a precomputed unpaired MSA for each chain, with no templates. The weights are the OpenFold3 preview-2 checkpoint ported to the AlphaFold3 parameter layout; no AlphaFold3 parameters were used.

Modes. Exact, Fast, and Big are all present. Exact computes the diffusion sampler’s shared per-step conditioning once per sample, replays each denoiser step as a captured CUDA graph and uses trunk kernels built to match default’s rounding and accumulation order. Its outputs were verified bit-identical to default on its identity input set. Fast, the optimized package’s primary accelerated mode, is guarded by requiring its outputs to stay within the spread default itself shows across seeds, and it passed this check. Big keeps Fast’s kernel numerics but restructures execution to reduce memory use, and passed the same accuracy check.

Optimizations. Labels give the modes that use each change.

- **CUDA-graph step replay (Exact, Fast).** The diffusion sampler’s shared per-step conditioning is computed once per sample (as it also is in Big), and each denoiser step is captured as a CUDA graph (recording the GPU operations once and replaying them without per-step launch overhead) across all 5 samples and 200 steps.
- **bfloat16 weights (Fast, Big).** Model weights are cached in bfloat16 instead of float32, cutting the data moved and letting matrix multiplications run faster at a small, bounded numerical cost.
- **Fused diffusion-transformer kernel with batched samples (Fast, Big).** The diffusion transformer’s repeated blocks are replaced by one fused kernel, and all five diffusion samples are advanced together per step rather than one at a time.
- **Fused trunk kernels (Fast, Big).** Triangle multiplication, triangle attention, the pair-transition block, pair-bias attention, and the MSA-module operations are merged into fused kernels that cut the number of separate GPU launches through the trunk’s many blocks and recycles. Exact instead uses bit-matching kernels for triangle multiplication and the transition, plus changes that reduce data movement.
- **Memory-recomposed execution for Big.** For Big, the CUDA graph is dropped in favor of running steps eagerly, diffusion buffers and prior-recycle embeddings are freed as soon as they are no longer needed, the sampler’s pair conditioning is evaluated in chunks, and a fragmentation-resistant allocator mode is used. This trades some speed for much greater size headroom, including a variant that shards the pair representation across two GPUs.

Results. Exact’s forward-call speed-up averages 2.9× (geometric mean over the seven input sizes) against this model’s own default; the main-text comparison with AlphaFold3 (JAX) at default is a different baseline. Fast is the fastest mode at every input size on the forward-call clock. Big trails it once its CUDA graph is removed. Exact – limited to kernels that match default bit-for-bit – gives the smallest gains, particularly at larger sizes where the trunk, served largely by the upstream package’s own kernels under Exact, dominates the call. For Fast and Big, speed-ups on the forward clock are consistently higher than on the whole-pass wall time, since featurization and output writing outside the forward call do not shrink with the optimized kernels. For Exact the relationship can reverse, since its overlapped featurization and writing can make whole-pass speed-ups similar to, or larger than, forward speed-ups. Whole-pass geometric means are 3.0× (Exact), 9.1× (Fast), and 7.4× (Big). On peak memory, Exact and Fast sit modestly

above default because they hold a captured-graph memory pool that default's eager execution never allocates, plus cached bfloat16 casts (Exact) or compile workspaces, input padding, and batched-sample activations (Fast). Big, in contrast, lowers peak memory at every input size by dropping that pool and freeing buffers eagerly. Big therefore completes larger complexes than default on one GPU (4,000 versus 2,000 tokens; default ran out of memory at 3,000 tokens in the search shown) and, sharded across two GPUs, 10,000 tokens before running out of memory, a value from one-recycle probe runs that may be off by one size step (p. 16).

Accuracy on FoldBench-Lite. On the 243 interfaces of 152 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable (DockQ ≥ 0.23) for 53.9% of interfaces under default and for 53.9%, 52.3%, and 52.7% under Exact, Fast, and Big (Figure S29). No paired change from default, for any interface class, for ligands or for monomers, has a 95% confidence interval that excludes zero. Confidence scores track default closely (Figure S30): across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r = 1.00$ (to two decimals), and the mean paired differences are at most 0.005 for ipTM, at most 0.002 for pTM, and at most 0.09 points for mean pLDDT.

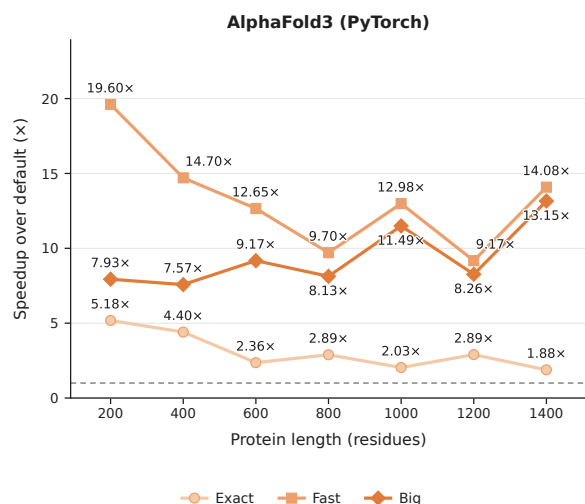


Figure S26. AlphaFold3 (PyTorch): forward-call speed-up over default by input size. Speed-up (default's forward-call time divided by the mode's, on the same CUDA-synchronized per-item forward clock) for Exact, Fast, and Big versus input size (200–1,400 tokens) on H100; the dashed line marks default (1x).

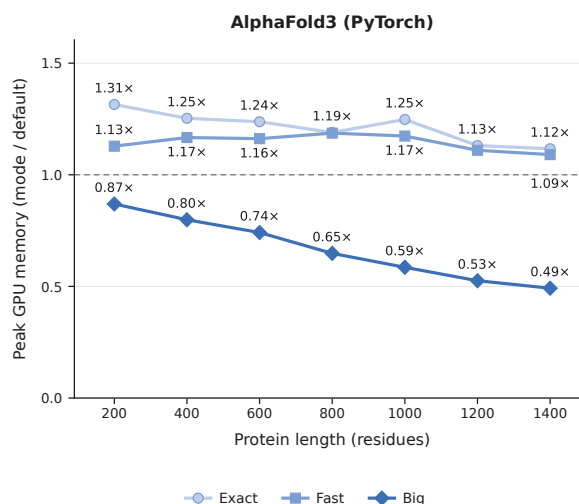


Figure S27. AlphaFold3 (PyTorch): peak GPU memory relative to default. Peak NVML device memory above idle for Exact, Fast, and Big, expressed as a ratio to default at the same input size, measured in dedicated memory-only runs; the dashed line marks default (1x).

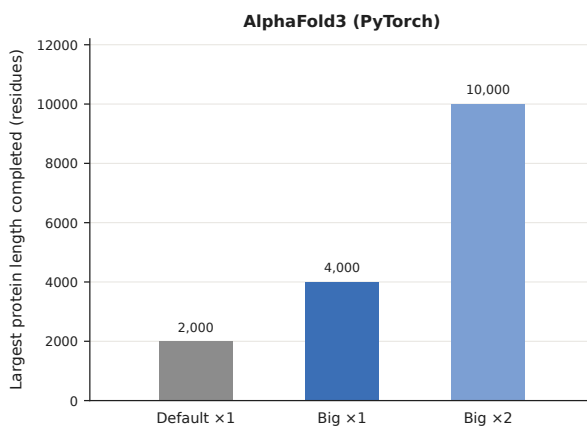


Figure S28. AlphaFold3 (PyTorch): largest completed input size (reach). Largest input size, in tokens, that completes for default, Big on one GPU, and Big on two GPUs; each bar stops at the largest size that ran without exhausting GPU memory.

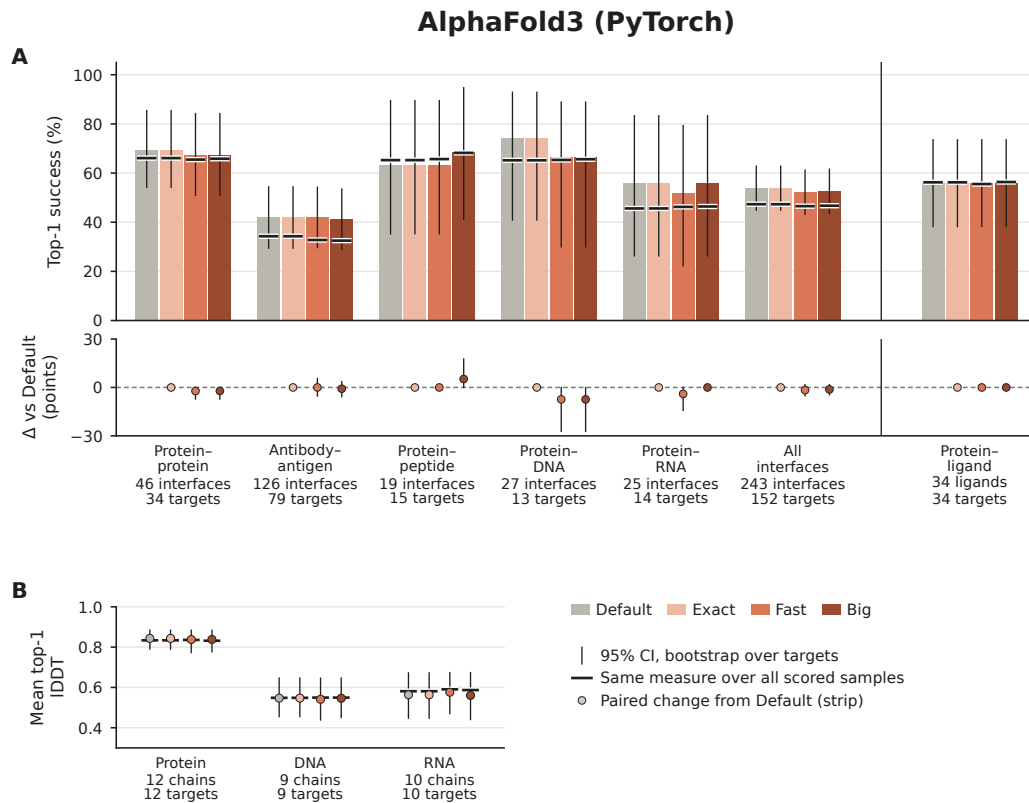


Figure S29. AlphaFold3 (PyTorch): accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

AlphaFold3 (PyTorch)

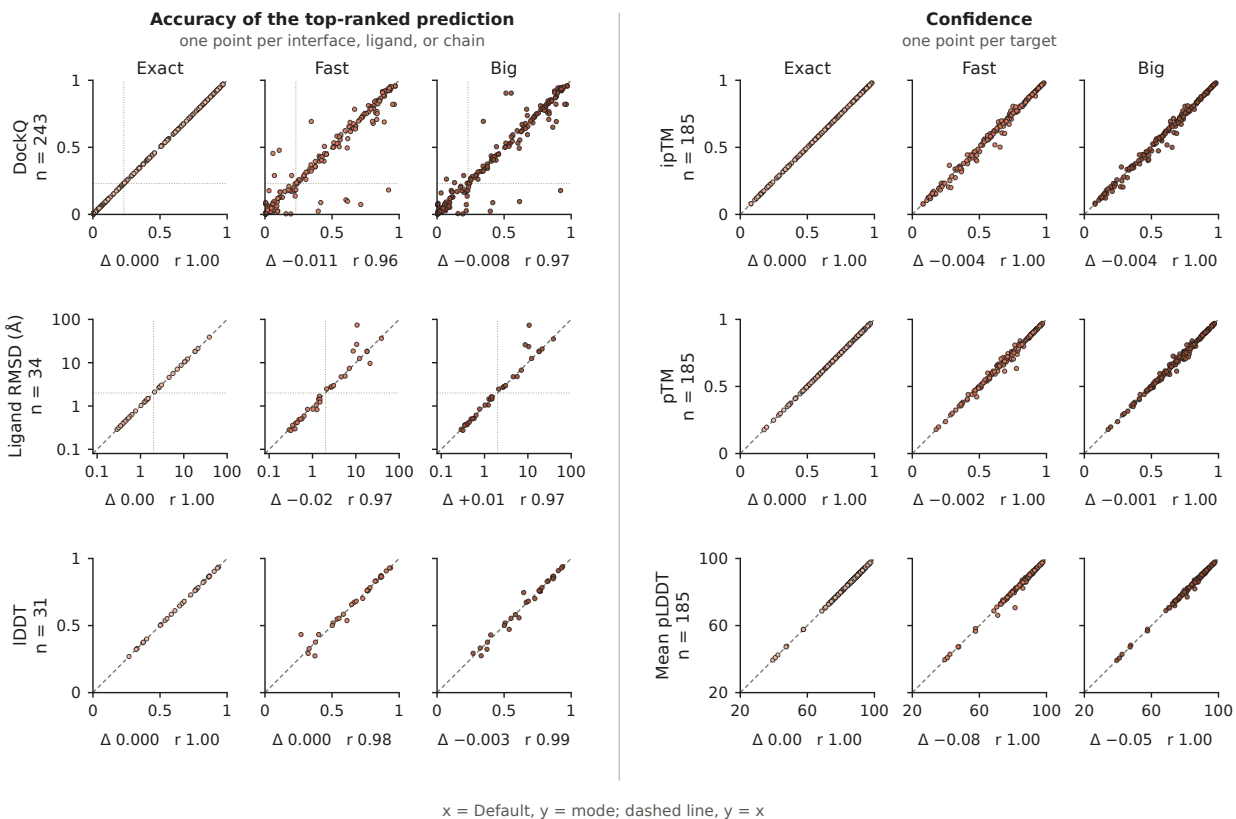


Figure S30. AlphaFold3 (PyTorch): accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S29), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.7 Protenix v2

Protenix v2 is ByteDance’s cofolding model for proteins, nucleic acids, ligands, and ions, an AlphaFold3-class diffusion-based structure predictor; the benchmark measured upstream release v2.0.0 (checkpoint `protenix-v2`), run through its own command-line interface [40].

Benchmark setup. Each call folds one protein complex through the model’s own command-line interface; there is no batching axis, so one item equals one call. Speed and memory were measured on protein complexes at 200, 400, 600, 800, 1,000, 1,200, and 1,400 tokens. A separate set of homomeric assemblies, in 1,000-token steps from 2,000 to 13,000 tokens, probed the largest input that completes. Timed passes discarded warm-up copies and reported the median of the remaining repeats, on a single H100 80 GB GPU, except the Big-on-two-GPUs reach measurements, which used two NVLink-connected H100s. The headline speed-up uses the model’s own forward (prediction) call clock – the trunk, diffusion sampler, and confidence head, excluding input featurization and output writing. Whole-pass wall time is reported alongside as a companion clock. Peak GPU memory is the NVML device-memory high-water mark above an idle baseline for a dedicated single pass per input size and mode; the allocator’s own peak reads lower, since it excludes reserved memory pools and the CUDA context. For this model, default runs the package through its documented command line at its own defaults, plus two additions applied to every mode alike: the checkpoint is set explicitly to the v2 model, and the number of trunk cycles is raised from the upstream setting of 10 to 11 (10 recycles, 11 trunk passes). Otherwise, default matches the unmodified base package, which already enables its documented speed settings, so default is also the fastest correct way to run it.

Sampling settings. Default and every mode used the same sampling settings: 11 trunk passes (`--cycle 11`; upstream default 10), 200 diffusion steps, and 5 diffusion samples per call, with the `protenix-v2` checkpoint selected explicitly (`--model_name protenix-v2`) because the command-line default is the v1 checkpoint. Inputs are precomputed unpaired MSAs from `colabfold_search`, with no templates; each call predicts one complex.

Modes. Exact, Fast, and Big all exist for Protenix v2, and Big was also run on two GPUs to reach larger inputs. Exact fuses core trunk computations into hand-written kernels, captures the diffusion sampler (and, for inputs at or below 448 tokens, the whole block stack) in CUDA graphs – recording GPU operations once and replaying them without Python overhead – and substitutes a fused triangle-computation kernel only when it has been proven bitwise equal to the default kernel on the running GPU, otherwise falling back to default. It was verified to give bit-identical outputs to default. Fast keeps most of Exact’s optimizations and adds reduced-precision replacements – fused triangle attention and float16-with-float32-accumulation attention (lower-precision multiplication with higher-precision accumulation) in the sampler and atom transformers, plus fused pair-averaging and pair-bias operations. These replacements are checked against an accuracy guard requiring their deviation from default to stay within default’s own seed-to-seed spread, and Fast passed this check. Big keeps Fast’s approximations but drops the CUDA graph capture and retained memory pool, adding memory-chunking and cache-freeing optimizations instead. In exchange, it trades some speed relative to Fast, especially at smaller input sizes, for substantially larger reachable inputs and lower peak memory.

Optimizations. The following optimizations account for most of the measured behavior; labels mark those used only by some modes.

- **Fused trunk kernels.** In each Pairformer block, layer norm and the projections share one fused kernel and the gating and output steps another, and triangle multiplication, triangle attention, and the transitions use dedicated kernels, cutting launch overhead across the 48 blocks, which run 11 times per prediction.
- **CUDA graph capture of the diffusion sampler (Exact, Fast).** The repeated denoising step (200 steps over 5 samples per complex) is recorded once and replayed, removing Python and kernel-launch overhead; for inputs at or below 448 tokens the whole 48-block stack is captured as well.
- **Reduced-precision fused attention (Fast, Big).** float16 arithmetic with float32 accumulation in the sampler and atom transformers, plus fused pair-averaging, trade exact bit-reproducibility for extra speed within default’s own seed-to-seed spread.
- **Memory-freeing and chunking (Big).** Pair and MSA caches are freed as soon as possible, the conditioning cache and attention pair biases are computed in chunks above 1,023 tokens, and the full relative-position tensor is never built, lowering peak memory so larger inputs fit. Big also lifts the unmodified package’s refusal of inputs above 2,560 tokens.
- **Multi-GPU row-sharding (Big on two GPUs).** The pair representation is split by rows across GPUs in the trunk, MSA module, template embedder, and confidence head, extending the largest input that fits beyond one GPU’s memory.

Results. Fast gives the largest speed-ups, with a geometric mean of 4.08× across input sizes: 6.3× at 200 tokens and 3.6–4.1× above that. Exact is 3.7× faster at 200 tokens and 2.1–2.4× above that, and Big’s speed-up grows with input size, from 1.5× at

200 tokens to 4.0 $\times$ at 1,400 tokens. Whole-pass geometric means are 2.39 $\times$ (Exact), 3.19 $\times$ (Fast), and 3.08 $\times$ (Big). Peak memory shows a clear trade-off. Big’s peak memory is below default at every tested size, falling from 0.78 $\times$ at 200 tokens to 0.42–0.47 $\times$ from 800 tokens up. Exact and Fast, which retain the CUDA-graph memory pools, run above default at the largest sizes tested. On one GPU, default completes 2,000 tokens (74 GB peak), and the unmodified package refuses the next size, 3,000 tokens, because it rejects inputs above 2,560 tokens. Big, which lifts that limit, completes 3,000 tokens on one GPU and 8,000 tokens on two GPUs before running out of memory. The two-GPU value is among the least certain (p. 16): it comes from shortened probe runs whose peak memory read 6% below the peak memory of a full-settings run at 4,000 tokens (45.3 vs 48.0 GB).

Accuracy on FoldBench-Lite. On the 251 interfaces of 152 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable (DockQ ≥ 0.23) for 66.1% of interfaces under default and for 64.5%, 64.5%, and 64.5% under Exact, Fast, and Big (Figure S34). No paired change from default, for any interface class, for ligands or for monomers, has a 95% confidence interval that excludes zero. Confidence scores track default closely (Figure S35): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are at most 0.004 for ipTM, at most 0.002 for pTM, and at most 0.05 points for mean pLDDT.

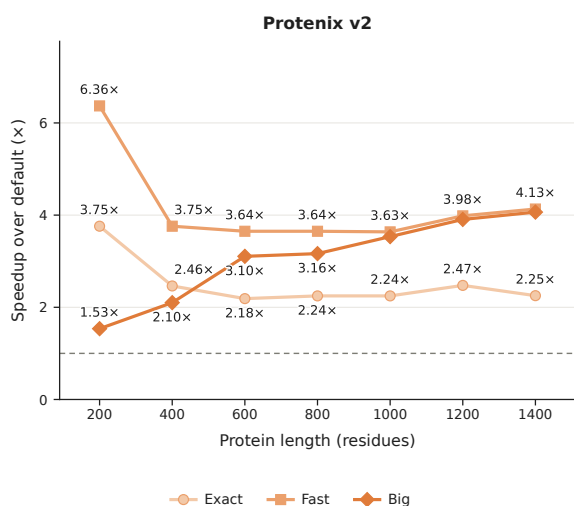


Figure S31. Protenix v2: forward-call speed-up over default vs. input size. Each line is one mode (Exact, Fast, Big) at input sizes from 200 to 1,400 tokens, measured on the model’s own forward (prediction) call clock; the dashed line marks default (1 $\times$).

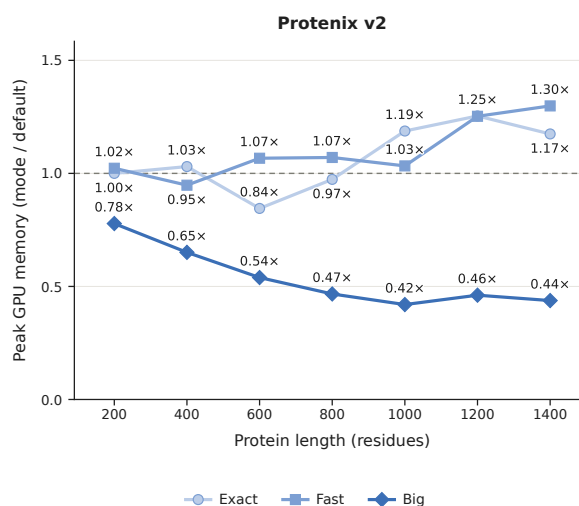


Figure S32. Protenix v2: peak GPU memory relative to default vs. input size. Each line is one mode (Exact, Fast, Big) showing the device’s peak memory above an idle baseline for a dedicated single pass at each input size from 200 to 1,400 tokens; the dashed line marks default (1 $\times$). Exact is relative to a separate default run; default’s peaks differ between the two runs by up to 22%.

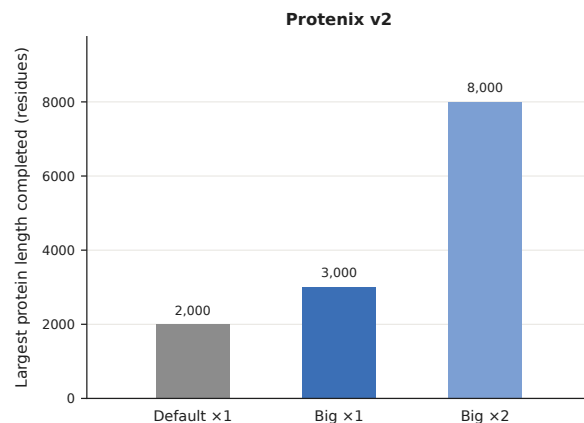


Figure S33. Protenix v2: largest input completed. One bar per configuration (default, Big, or Big on two GPUs) showing the largest homomeric-assembly input size, in tokens, that completed. Default's search ended because the package refuses inputs above 2,560 tokens, not because it ran out of memory.

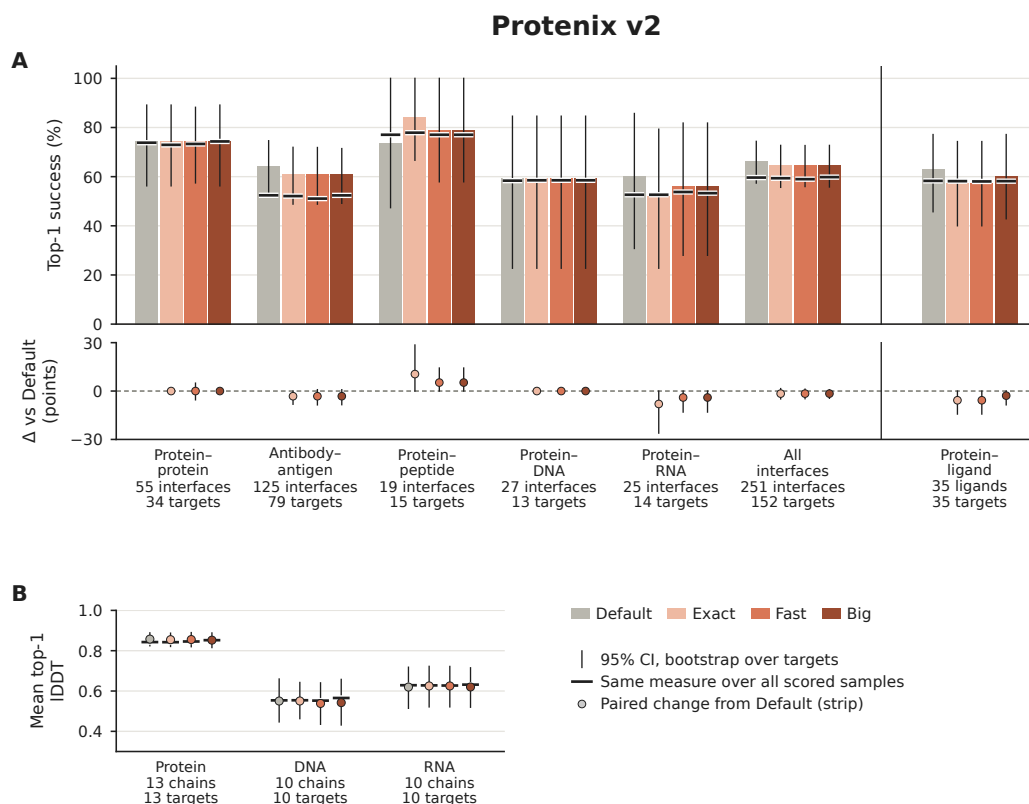
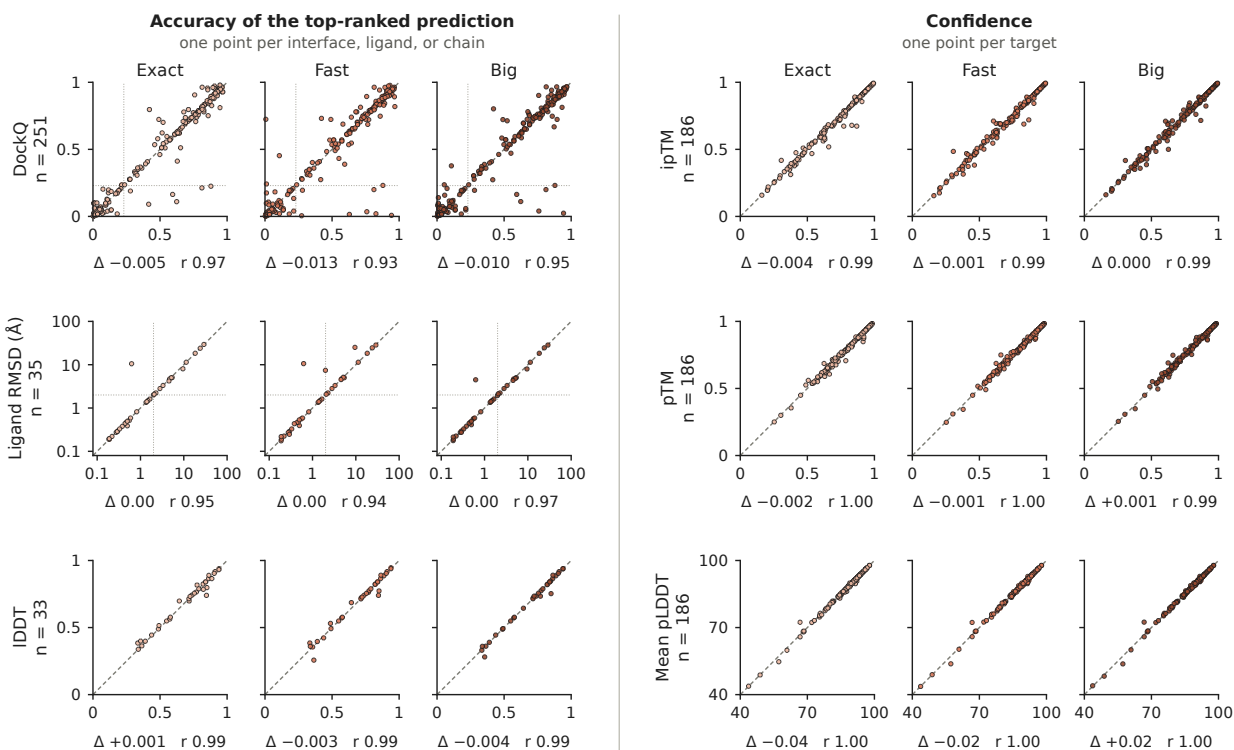


Figure S34. Protenix v2: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

Protenix v2



x = Default, y = mode; dashed line, y = x

Figure S35. Protenix v2: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, y = x. Left, accuracy of the top-ranked prediction (the predictions of Figure S34), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.8 Protenix v1

Protenix is an AlphaFold3-family structure predictor for protein complexes (input embedder, MSA module, template embedder, a recycled 48-block Pairformer trunk, and a diffusion-based structure sampler). The benchmark used the 1.1.0 release as distributed on PyPI (`protenix==1.1.0`) with its default checkpoint [38, 39].

Benchmark setup. Complexes were drawn from a fixed token ladder at exactly 200, 400, 600, 800, 1,000, 1,200 and 1,400 tokens. Items ran one at a time in a single process, with no batch dimension, timing 12 items per pass at 200–400 tokens and 6 at larger sizes after untimed warm-up items. This was repeated over 3 replicate runs on one NVIDIA H100 80 GB GPU (two GPUs for the reach measurements of Big on two GPUs). The headline clock is the time spent inside the model’s prediction call (trunk recycles, diffusion sampling, confidence head), excluding the small output-file dump. Upstream’s own forward-time readout, which includes the dump, gives speed-ups lower by less than 7%, and a whole-pass process wall clock is reported as a further companion. Peak GPU memory is the NVML device high-water mark for a dedicated pass, measured above the GPU’s idle baseline. Default is the upstream package run through its documented command-line interface, whose defaults already enable upstream’s documented speed settings. It differs from base only in running 11 trunk passes instead of the shipped 10, a fixed floor applied to every mode. Base was not measured separately. Each mode’s speed-up is computed against the default passes run before and after it on the same machine, to cancel host-to-host variance.

Sampling settings. Default and every mode used the same sampling settings: 11 trunk passes (upstream default 10), 200 diffusion steps, and 5 diffusion samples per call, with the `protenix_base_default_v1.0.0` checkpoint. Inputs are precomputed unpaired MSAs from `colabfold_search` (UniRef30 2302 and ColabFold environmental database 202108), one for each protein chain, with templates disabled; each call predicts one complex.

Modes. Exact, Fast, and Big are all present, plus Big on two GPUs. Exact reproduces default’s arithmetic bit-for-bit using fused, order-preserving kernels and was verified bit-identical to default under a deterministic recipe (320 of 320 compared outputs). Fast uses the same fusions with lower-precision kernels and was checked to stay within default’s own seed-to-seed variation on structural accuracy. Big keeps Fast-class numerics but is retuned for the lowest peak memory, extending the largest input size that fits on one or more GPUs. Big passed the same seed-spread checks. Big on two GPUs also passed them, on the standard and templated identity inputs.

Optimizations. The optimized package applies run-time kernel replacements to the unmodified upstream model; its weights and featurization pipeline are unchanged.

- **CUDA-graph replay of the diffusion sampler.** The repeated 200-step denoising loop is captured once as a CUDA graph (recording the GPU operations once and replaying them without Python overhead), with step-invariant projections hoisted out of the loop. This change is the largest single contributor to Exact’s speed-up, and Fast uses it too (Big does not).
- **Fused triangle multiplication.** A core Pairformer operation is rewritten as one fused kernel that preserves upstream’s summation order.
- **Fused triangle attention.** Triangle attention runs as one fused block around a kernel that reproduces upstream’s arithmetic bit-for-bit on H100 (Exact).
- **Lower-precision fused kernels (Fast mode).** Triangle and diffusion attention run as bfloat16/float16 flash-style kernels, and the diffusion-transformer and atom blocks as fused kernels with lower-precision matrix multiplications, trading exact reproducibility for extra speed within an accuracy guard.
- **Memory-directed changes (Big mode).** Upstream’s own chunked pair path replaces the CUDA-graph memory pools. The initial pair tensor waits in pinned host memory between trunk passes, and caches are freed after last use. Several stages (above 2,048 tokens also triangle multiplication) run in row blocks, lowering peak memory at some cost in speed.

Results. Fast is the fastest mode at every size from 200 to 1,400 tokens, 2.8–3.6× faster than default. The 200-token Fast and Big values are the least certain (default’s timing drifted by more than 5% in one of three replicates). Big trades some of that speed for lower memory; its speed-up grows with input size, from 1.2× at the two smallest sizes to 2.4× at 1,400 tokens. Exact, with bit-identical outputs, is 1.5–1.7× faster, depending on input size. Big uses less peak GPU memory than default at every input size, whereas Exact uses more at every size and Fast at every size from 400 tokens. Exact and Fast use more because both keep the diffusion sampler’s CUDA-graph memory pools and fused-kernel workspaces in memory, the trade-off that Big is designed to avoid. Big modestly raises the largest input that completes on one GPU. Big on two GPUs raises it several-fold, splitting the pair representation across the GPUs so that no single GPU holds a full input-sized tensor. This two-GPU value is among the least certain (p. 16): it rests on shortened probe runs (two trunk passes; diffusion samples unchanged) whose peak memory read 16% below the

peak memory of a full-settings run at 4,000 tokens, and the next size failed with a communication timeout that is counted as running out of memory.

Accuracy on FoldBench-Lite. On the 251 interfaces of 152 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 55.0% of interfaces under default and for 55.0%, 55.4%, and 55.8% under Exact, Fast, and Big (Figure S39). The only paired change from default whose 95% confidence interval excludes zero is a change of -0.040 in mean RNA-monomer lDDT under Big. Confidence scores track default closely (Figure S40): across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r = 1.00$ (to two decimals), and the mean paired differences are at most 0.001 for ipTM, below 0.0005 for pTM, and at most 0.03 points for mean pLDDT.

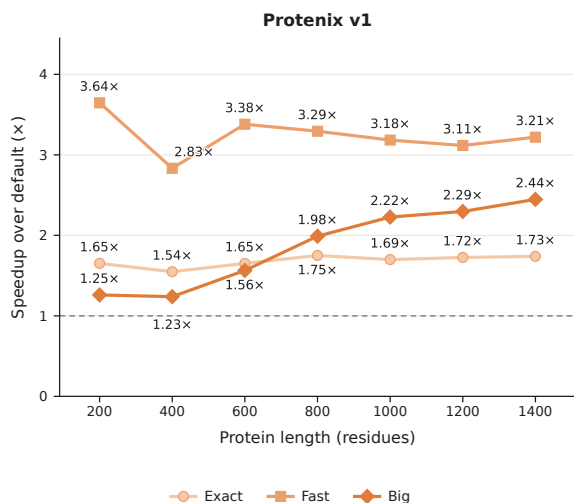


Figure S36. Protenix v1: speed-up over default vs. input size on H100. Each line is one mode (Exact, Fast, Big); the clock is the time per item inside the model's prediction call. The dashed line marks default (1x).

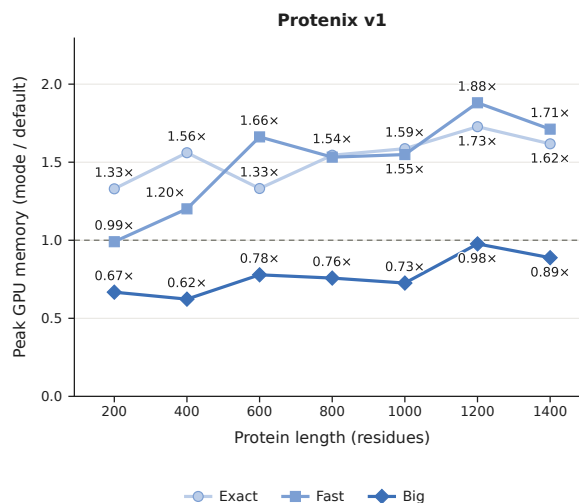


Figure S37. Protenix v1: peak GPU memory relative to default vs. input size on H100. Each line is one mode (Exact, Fast, Big); the meter is the NVML device peak above idle baseline from dedicated memory passes. The dashed line marks default (1x).

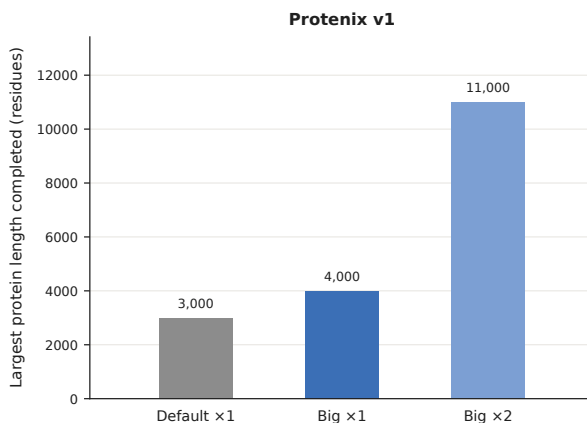


Figure S38. Protenix v1: largest input completed. One bar per configuration (Default x1, Big x1, Big on two GPUs), showing the largest homooligomer input size that completed; the next size ran out of GPU memory, except for Big on two GPUs, whose next size failed with a communication timeout.

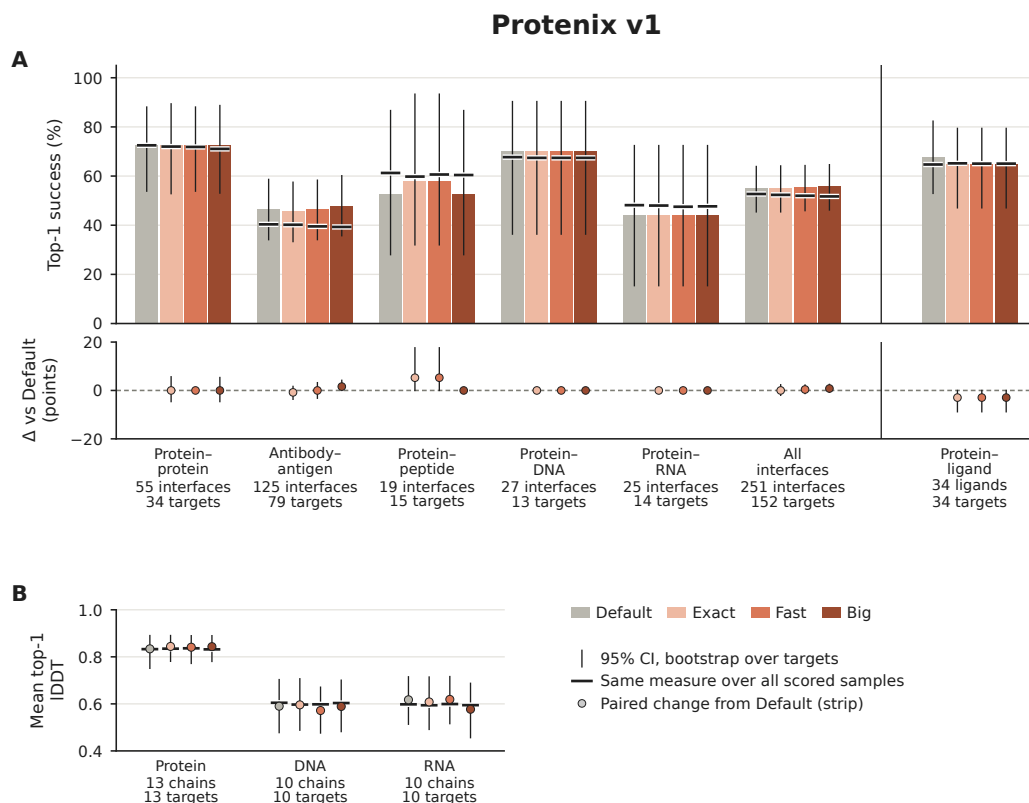
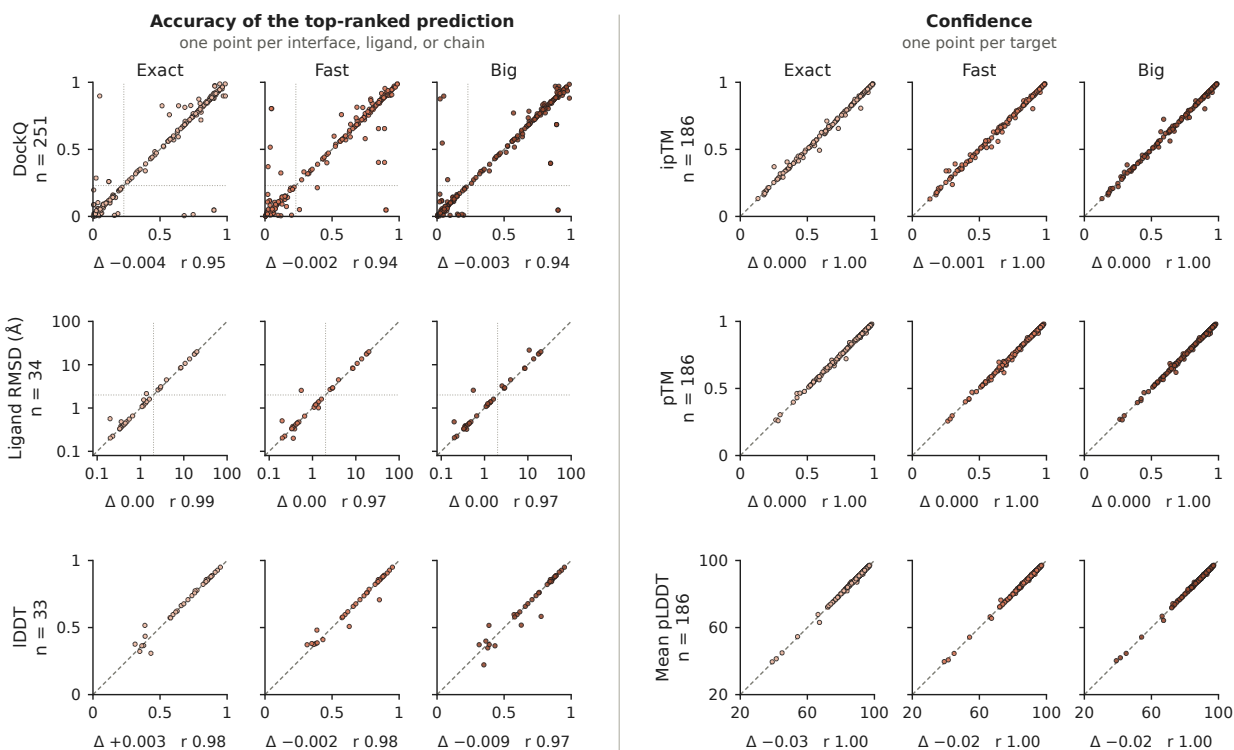


Figure S39. Protenix v1: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

Protenix v1



x = Default, y = mode; dashed line, y = x

Figure S40. Protenix v1: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, y = x. Left, accuracy of the top-ranked prediction (the predictions of Figure S39), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.9 OpenDDE

OpenDDE is an AlphaFold3-class biomolecular cofolding model of the Protenix family, combining an input featurizer, a Pairformer trunk, a diffusion sampler, and a confidence head to predict complex structures as mmCIF output. We benchmarked upstream tag v1.1.1 (checkpoint `opendde.pt`) [41].

Benchmark setup. Speed and memory were measured on a fixed set of complexes at 200, 400, 600, 800, and 1,000 tokens, plus a high-range addendum at 1,200 and 1,400 tokens. Reach was measured on a separate set of inputs stepped in 1,000-token increments. Each call processes one complex on one GPU, with no batch axis. Speed runs used three replicates per input size on an H100 80 GB GPU, each an untimed warm-up pass followed by a default-mode-default timing sequence, every pass in a fresh process. Big at 1,200 and 1,400 tokens was measured in six replicates of three complexes each. The headline clock is upstream’s own per-item forward-call timer – trunk recycling, all diffusion samples, the confidence head, and output writing – excluding start-up, checkpoint loading, featurization, and reading the precomputed alignment. A companion whole-pass wall clock (start-up to finish) is reported alongside it. Peak memory is the device’s NVML high-water mark above idle, from dedicated single-pass memory runs. Default here is base plus its two documented one-line speed settings, `bfloat16` precision and a fused LayerNorm extension, chosen as the fastest correct configuration so that speed-ups are not measured against a needlessly slow baseline. Base (`float32`, unfused LayerNorm) is markedly slower than default from 400 tokens up, needs substantially more memory, and runs out of memory at 1,200 and 1,400 tokens where default still completes.

Sampling settings. Default and every mode used the same sampling settings: 11 trunk passes (`--cycle 11`; upstream default 10) and 5 diffusion samples of 200 steps each per call, with the `opendde_v1` checkpoint. Inputs are precomputed MSAs, with no templates; each call predicts one complex.

Modes. Exact, Fast, and Big are all available. Exact was verified to give bit-identical outputs to default under deterministic settings, confirmed by dedicated identity checks. The FoldBench-Lite comparison below ran at production settings, where Exact can differ from default (p. 16). Fast uses faster reduced-precision numerics, guarded by requiring that its structural deviation from default stay within the spread default itself shows across different seeds. Big recomposes Fast’s kernels to minimize peak memory and can split the computation by rows across several GPUs of one host. Fast, Big, and Big on two GPUs passed every seed-spread check.

Optimizations. The optimized package layers several changes on top of the unmodified model:

- **Trunk kernels (Fast, Big).** Triangle attention, triangle multiplication, pair transition, and pair-row LayerNorm in the Pairformer and confidence stacks run through fused, flash-style `bfloat16` kernels instead of upstream’s operations, speeding up the recycling passes. Not applied below 300 tokens.
- **Diffusion sampler graph capture and hoisting.** One denoiser step is captured once as a CUDA graph (recording the GPU operations once and replaying them without Python overhead) and reused for the remaining steps. Tensors that do not change between steps are computed once per call instead of at every step. Fast and Big also use pair-bias attention kernels and fused token and atom transformers at reduced precision.
- **Bit-exact kernel bindings (Exact).** Exact uses a bit-exact version of a fused kernel where one exists for the running hardware and software stack, and otherwise falls back to the unmodified operation, preserving bitwise identity with default.
- **Host-side overlap.** The next complex is featurized on the host while the current one is still computing on the GPU, and each prediction’s confidence JSON is encoded only once.
- **Memory-saving offload (Big).** Above a size threshold, pair tensors from the trunk, structure, and confidence stacks stream through pinned host memory in blocks instead of staying on the GPU, diffusion samples run one at a time, and hoisted sampler buffers are not kept resident, trading time for lower memory.

Results. Fast gives the largest forward-call speed-up over default from 400 tokens up, reaching $2.87\times$ at 1,400 tokens. Exact gives positive speed-ups at every size and edges out Fast at 200 tokens ($1.16\times$ versus $1.15\times$), where the trunk kernels are off. Big’s speed-up stays below Fast’s throughout, and falls further, to $0.63\times$, once its pair-offload path engages at 1,400 tokens, trading forward-call time for a large cut in peak memory. This time depends on the host (about 480–785 s per complex on different H100 hosts, against 316 s for default). Peak memory for Exact and Fast stays close to default at most sizes, whereas Big saves most at the largest inputs (52% at 1,400 tokens, against 2–7% at 200–400 tokens) and completes inputs up to 4,000 tokens on one GPU, well beyond default’s limit, and up to 5,000 tokens on two. Both reach values come from shortened probe runs (two trunk passes; diffusion samples unchanged), checked against full-settings runs at 4,000 tokens on one GPU and 3,000 tokens on two whose peak memory matched, so they show what fits in memory, not a run time.

Accuracy on FoldBench-Lite. On the 222 interfaces of 152 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 60.8% of interfaces under default and for 59.5%, 59.5%, and 60.4% under Exact, Fast, and Big (Figure S44). The only paired changes from default whose 95% confidence intervals exclude zero are a change of -0.003 in mean protein-monomer IDDT under Big and a change of $+0.030$ in mean DNA-monomer IDDT under Exact. Confidence scores track default closely (Figure S45): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are below 0.0005 for ipTM, at most 0.001 for pTM, and at most 0.05 points for mean pLDDT.

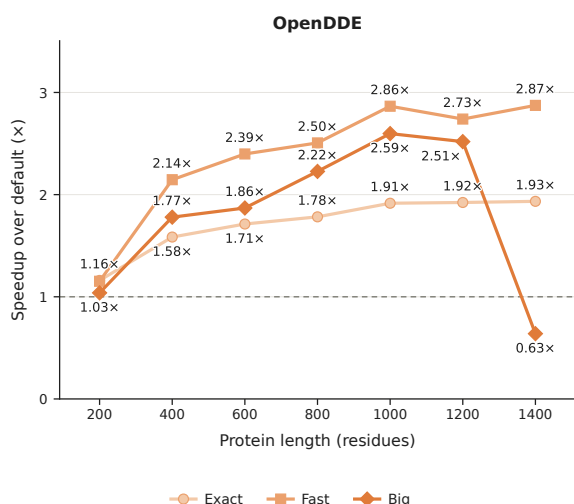


Figure S41. OpenDDE: forward-call speed-up over default vs. input size. Lines show the per-item forward-call speed-up (default time divided by mode time) for Exact, Fast, and Big across input sizes in tokens; the dashed line marks default (1 $\times$).

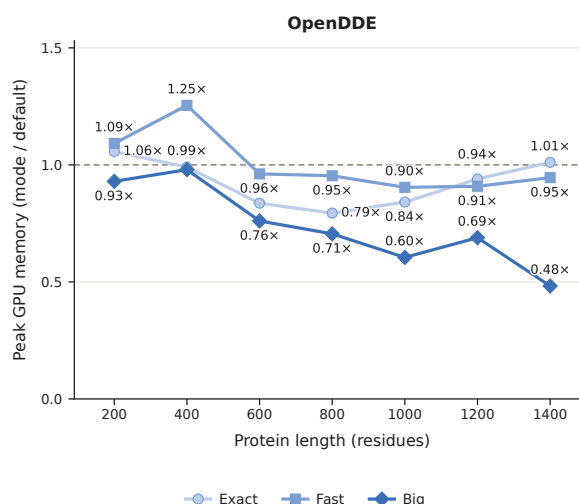


Figure S42. OpenDDE: peak GPU memory relative to default vs. input size. Lines show each mode’s peak device memory (NVML high-water mark above idle, from dedicated single-pass memory runs) relative to default, across input sizes; the dashed line marks default (1 $\times$).

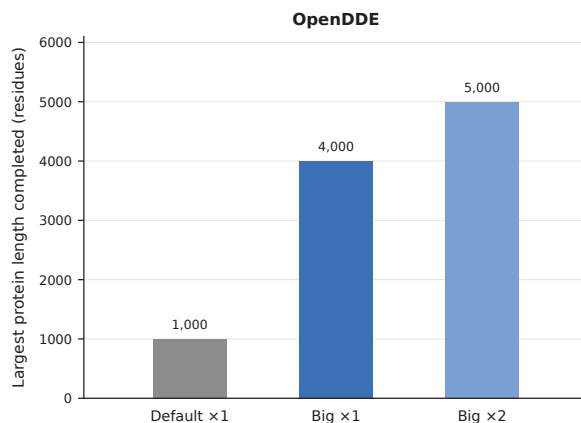


Figure S43. OpenDDE: largest completed input size by configuration. Bars show the largest complex size (tokens) that completes for default on one GPU, Big on one GPU, and Big on two GPUs. Default’s bar is limited by the 1,000-token steps of the reach search: default completed 1,000 tokens and ran out of memory at 2,000, and it also completes the 1,200- and 1,400-token inputs of the speed benchmark.

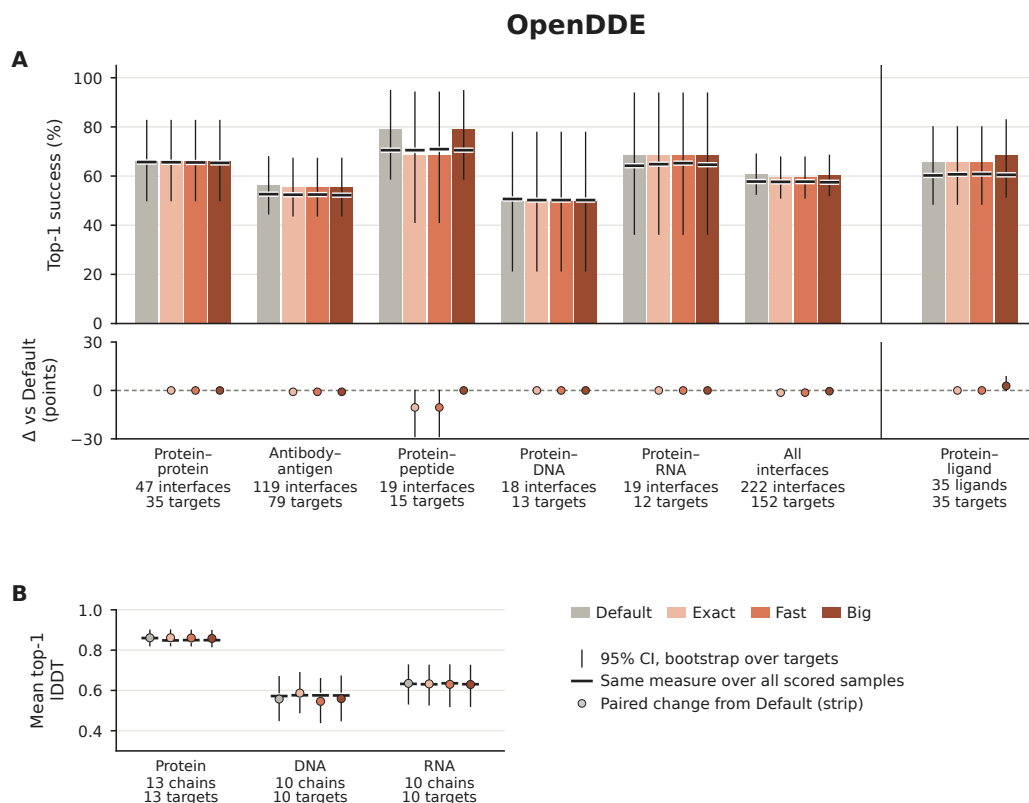


Figure S44. OpenDDE: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

OpenDDE

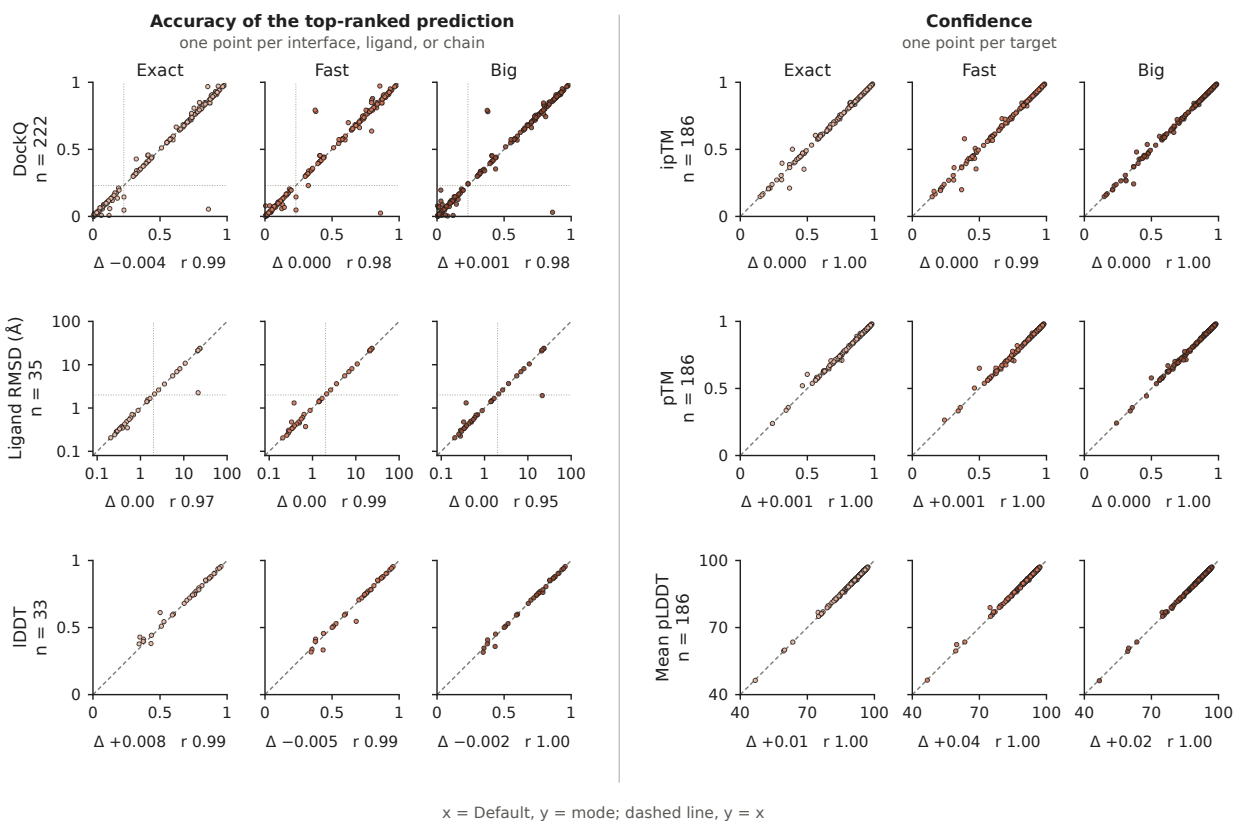


Figure S45. OpenDDE: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, y = x. Left, accuracy of the top-ranked prediction (the predictions of Figure S44), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.10 Boltz-2

Boltz-2 is an all-atom biomolecular structure predictor of the AlphaFold-3 class, combining an MSA module and a Pairformer trunk with a diffusion sampler and a confidence head to predict complex structures from sequence input. The benchmarked release is the upstream `boltz 2.2.1` package (tag `v2.2.1`) with the published `boltz2_conf.ckpt` checkpoint [10].

Benchmark setup. Each call predicts one biomolecular complex (batch = 1), drawn from a fixed speed/memory ladder of input sizes from 200 to 1,400 tokens. Several complexes were measured per size, with the first items of each pass excluded as warm-up. All speed and memory runs used one NVIDIA H100 80 GB GPU and the sampling settings below; the two-GPU reach search used shortened runs (two trunk passes, two sampling steps) whose peak memory matched the peak memory of a full-settings run at 4,000 tokens. The headline clock times only the model’s forward/prediction call (embedder, MSA module, trunk passes, diffusion sampling, confidence head), excluding input featurization and structure-file writing. A companion whole-pass wall-clock measurement covers worker start-up, model loading, and all items, and gives smaller speed-ups (1.41–1.86× for Exact, 2.14–3.48× for Fast, and 2.10–3.98× for Big). Peak GPU memory is the device high-water mark above an idle baseline, measured in dedicated passes separate from the timing runs. For this model, default is the upstream `boltz predict` command with the sampling settings below, which every mode shares. The command already enables its fastest kernels and mixed-precision execution, so no other change was needed and no separate base measurement is reported.

Sampling settings. Default and every mode used the same sampling settings: 10 recycling steps (11 trunk passes; upstream default 3), 200 sampling steps, and 5 diffusion samples per call, denoised as one chunk (upstream default 1). The model is Boltz-2 (`boltz 2.2.1`, checkpoint `boltz2_conf.ckpt`) without the affinity head. Inputs are precomputed MSAs used at full depth, without subsampling, and no templates; each call predicts one complex.

Modes. Exact, Fast, and Big are all implemented for Boltz-2, plus a Big on two GPUs configuration. Exact was verified to produce bit-identical output to default in the three identity checks run at the benchmarked settings. Fast uses reduced-precision (bfloat16) arithmetic and fused kernels, with outputs verified to stay within default’s own seed-to-seed variation. Big trades some of Fast’s speed for lower peak memory by processing wide intermediate tensors in row chunks and dropping the largest memory buffers. Big on two GPUs additionally shards the pairwise representation across the two cards. Both passed the same seed-to-seed check as Fast.

Optimizations. The optimized package wraps the unmodified upstream model in a persistent worker and applies the following runtime changes, ordered roughly by their contribution to the speed-up:

- **Fused triangle-attention and triangle-multiplication kernels (all modes; bit-identical versions in Exact).** The Pairformer trunk’s pairwise attention and multiplication operations are fused into single, larger kernels instead of many small ones. In Fast, the bfloat16 versions of these kernels provide most of the speed-up, and their share grows with input size.
- **CUDA-graph diffusion sampler (Exact, Fast).** The 200-step, 5-sample diffusion sampler is captured once as a replayable CUDA graph (recording the GPU operations once and replaying them without repeated Python overhead) instead of reissuing each step individually. This is the largest single contributor to Exact’s speed-up; Big leaves CUDA graphs off to save memory.
- **Resident bfloat16 pair tensor (Fast, Big).** The pairwise representation is kept resident in lower-precision (bfloat16) format across the trunk, MSA module, and confidence head, reducing data movement and compute time.
- **Step-invariant conditioning computed once per sample (Exact, Fast).** Quantities that do not change across diffusion steps, such as normalization terms and expanded pairwise biases, are computed once per sample instead of being recomputed at every sampling step.
- **Overlapped input/output with GPU work.** Preparing the next input and writing outputs are overlapped with ongoing GPU computation rather than pausing it.

Results. Across the input-size ladder, Fast gives the largest forward-call speed-up, a geometric mean of 5.56× over default. The benefit of the fused trunk kernels grows with input size, and the CUDA-graph sampler helps at every size tested. Big and Exact are slower than Fast but remain faster than default at every size tested. Peak GPU memory for Exact and Fast exceeds default’s at every size from 400 tokens up – reaching up to 1.90× (Exact) and 2.13× (Fast) at 400–1,000 tokens and still 1.28–1.30× and 1.04–1.09× at 1,200–1,400 tokens. Both modes hold additional sampler state that scales roughly with the square of the token count. Above 1,024 tokens this state is released before the confidence pass, and a memory-headroom check can hand a prediction to the upstream sampler. Big instead reduces memory throughout by dropping this sampler state and chunking wide intermediate tensors. On the reach ladder, Big on two GPUs completes inputs up to 9,000 tokens, well beyond what default or single-GPU Big can process before running out of memory.

Accuracy on FoldBench-Lite. On the 251 interfaces of 152 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 53.0% of interfaces under default and for 53.0%, 51.4%, and 52.2% under Exact, Fast, and Big (Figure S49). No paired change from default, for any interface class, for ligands or for monomers, has a 95% confidence interval that excludes zero. Confidence scores track default closely (Figure S50): across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are at most 0.005 for ipTM, at most 0.004 for pTM, and at most 0.05 points for mean pLDDT.

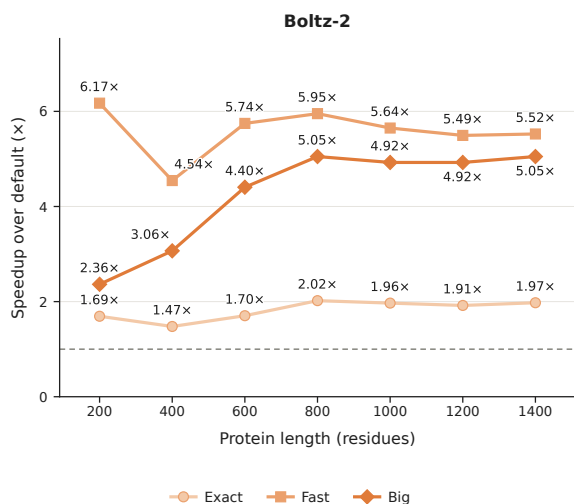


Figure S46. Boltz-2: forward-call speed-up over default vs. input size. Each line is one mode (Exact, Fast, Big) measured on the H100 across the 200–1,400 token ladder, using the timer around the model's forward/prediction call; the dashed line marks default (1x).

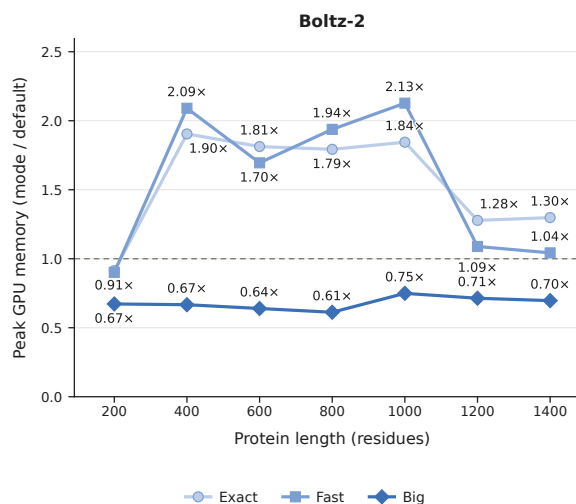


Figure S47. Boltz-2: peak GPU memory relative to default vs. input size. Peak GPU memory (NVML device high-water mark above idle baseline) vs. input size on the H100, one line per mode (Exact, Fast, Big) from dedicated passes separate from the timing runs; the dashed line marks default (1x). Exact and Fast share one label at 200 tokens (0.91x, 0.90x).

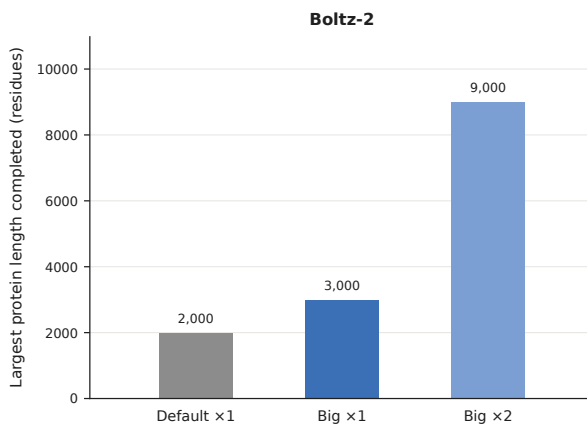


Figure S48. Boltz-2: largest completing input size by mode. Bars show the largest input complex (in tokens) that completes on H100 GPUs for default on one GPU, Big on one GPU, and Big on two GPUs.

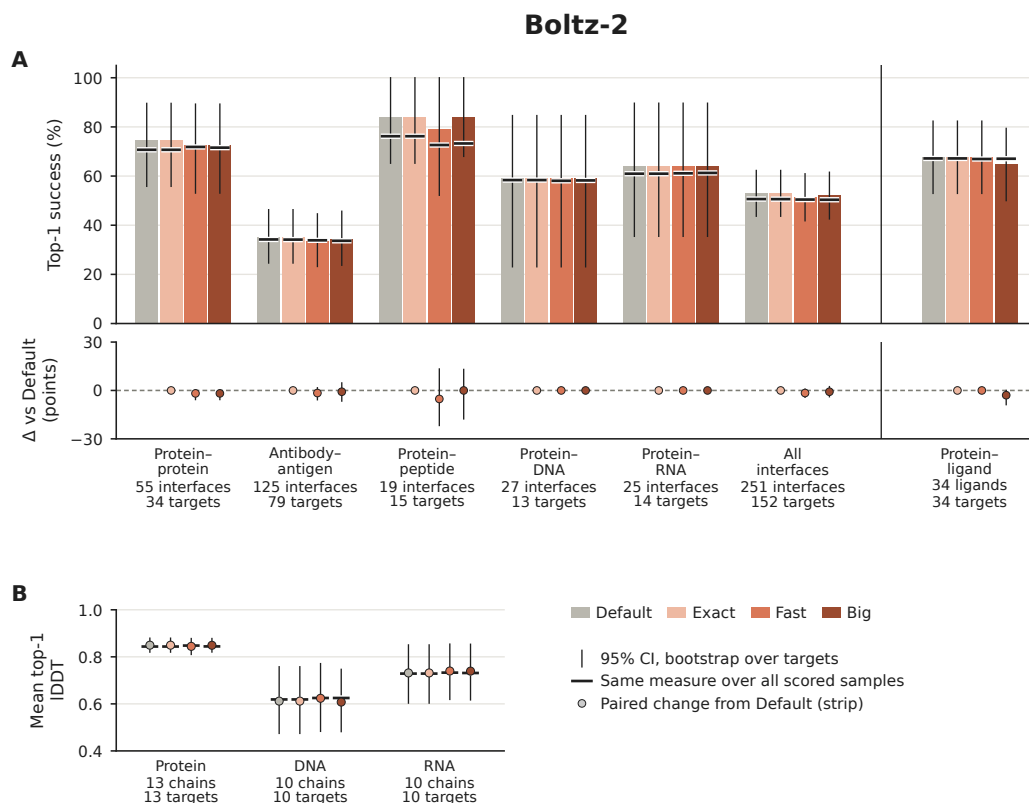


Figure S49. Boltz-2: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

Boltz-2

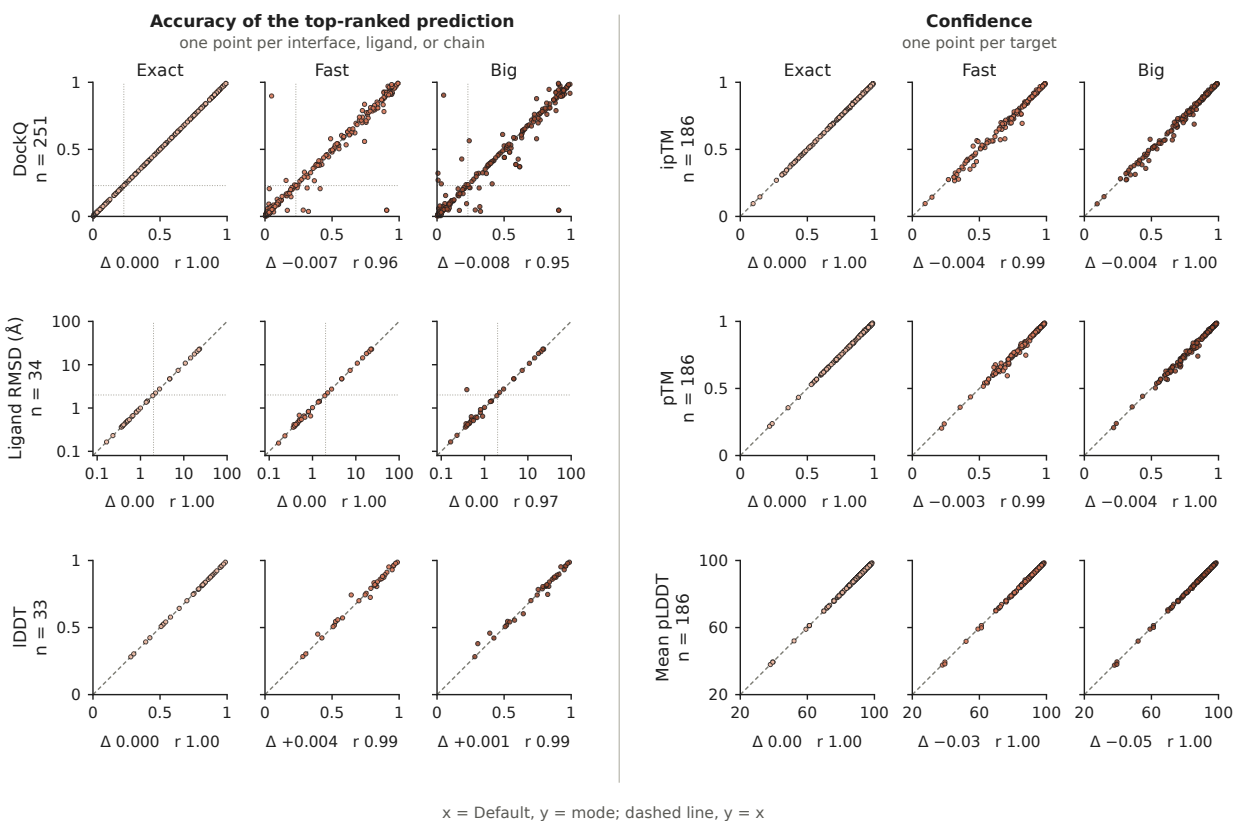


Figure S50. Boltz-2: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S49), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.11 Chai-1

Chai-1 is an AlphaFold-3-class cofolding model that takes a FASTA input (proteins, nucleic acids, ligands, modified residues) and outputs one mmCIF structure plus one score file per diffusion sample, built around a pairformer trunk with an MSA module and template embedder, a 200-step diffusion sampler, and a confidence head. The benchmarked release is `chai_lab 0.6.1` [42].

Benchmark setup. Folding tasks were drawn from a fixed series of seven input sizes, from 200 to 1,400 tokens, using two homo-oligomeric complexes per size. Each complex is folded in its own call, since cofolding calls are not batched (batch size of one throughout). Each input size was measured in three replicate runs, each timing default, Exact, Fast, Big, and default again in fresh processes (warm-up items excluded) after an untimed warm-up pass. Each mode is compared with the mean of the two default passes, whose agreement checks for drift. All measurements used a single H100 80 GB GPU dedicated to one job at a time. The headline clock is the time of the network itself per complex – the trunk passes, the diffusion sampler, and the confidence module – synchronized with the GPU; FASTA parsing, MSA loading and featurization run before the prediction call in every configuration and are in no per-item clock, only in the whole-pass clock. The time of the whole prediction call, kept in the data as context, adds to the network time the per-call loading of the model’s exported components (default reloads them for every complex, about 20 s per fold, while the optimized modes keep them loaded), candidate ranking, and output writing. A secondary whole-pass clock, covering process launch through the last item’s output, gives speed-ups of 2.40–3.34× for Exact, 3.96–5.54× for Fast, and 4.09–5.72× for Big. These are smaller than the headline speed-ups at every size for Fast, from 400 tokens up for Big, and from 1,000 tokens up for Exact, but larger for Big at 200 tokens and for Exact at 200–800 tokens. The differences arise because the whole pass also includes work outside the network that default repeats for every complex, such as reloading the model components, which the optimized modes avoid by keeping them loaded. Peak GPU memory is the device’s high-water NVML reading above an idle baseline, taken from a dedicated warm-up-then-timed memory run per input size. This measurement includes allocator reservations and CUDA context overhead, not tensor storage alone. For this model, default keeps the package’s own inference entry point but adds three settings on top of the shipped defaults: 11 trunk recycles rather than the shipped three, the full-memory (non-low-memory) code path, and ESM2 embeddings switched off in favor of precomputed multiple-sequence alignments used as evolutionary input on every configuration. The recycle count and the alignment input are benchmark settings shared by every configuration (the shipped 3-recycle, ESM2-on settings were not measured); the full-memory code path is what makes default the fastest correct way of running the unmodified package. The shipped low-memory code path, run with the benchmark’s other settings unchanged, was measured separately and found to run at 0.95–0.97× default’s speed per prediction call (timed over the whole call rather than the network alone) across the input sizes tested, but it used 0.81–0.96× of default’s peak memory, so default is the faster baseline but not the leaner one.

Sampling settings. Default and every mode used the same sampling settings: 11 trunk passes (`num_trunk_recycles=11`; upstream default 3), 1 trunk sample, and 5 diffusion samples of 200 steps each per call. Inputs are precomputed MSAs, with no templates and with ESM2 embeddings switched off. The model is Chai-1 (`chai_lab 0.6.1`), and each call folds one complex.

Modes. Exact, Fast, and Big are all available for this model; Big runs on a single GPU only, since the optimized package refuses to run it across more than one. Exact reproduces default’s output bit for bit under a deterministic setting. This was confirmed on every case of the identity check except 3 cases, which were set aside because the upstream package itself is not reproducible on them. These inputs receive an unseeded random rotation of reference conformers for non-standard residues during feature preparation. A separate bit-identity check of Exact on the inputs of the accuracy benchmark did not pass; this does not affect the speed, memory or reach results reported below. Fast enables TF32 arithmetic and runs the trunk at each input’s token length rounded up to a multiple of 64 rather than at the padded size to which every input is normally rounded up. It was checked by requiring predicted coordinates to stay within the spread that default’s own predictions show across random seeds, and they do. Big applies the same numerics as Fast in a more memory-lean form and also stays within that spread.

Optimizations. Changes are ordered by their estimated contribution to the measured speed-up.

- **Graph-replayed denoiser step (Exact, Fast; Big without the CUDA graph).** The six shipped modules are re-expressed as eager PyTorch (ordinary Python execution rather than a pre-compiled graph). The diffusion step’s work that does not depend on the current noise level is computed once, and the remaining per-step work is replayed from a CUDA graph (recording the GPU operations once and replaying them without Python overhead). Together, these changes remove most of the launch overhead across the 200-step sampler.
- **Fused trunk kernels and true-length trunk (Fast, Big).** Fused triangle-multiplication and triangle-attention kernels in the pairformer and MSA module, together with running the trunk at the input’s token length rounded up to a multiple of 64 instead of its padded size, cut both kernel-launch overhead and wasted computation on padding.
- **Compiled sampler step and fused diffusion attention (Fast, Big).** The denoiser step is further compiled (and, in Fast, still replayed inside the CUDA graph), alongside a fused attention kernel in the diffusion transformer, shrinking per-step cost further.

- **Memory-lean kernels for Big.** Row-blocked triangle-multiplication/attention and blocked outer-product-mean/MSA averaging, together with dropping the CUDA-graph private memory pool, trade a small amount of speed for lower peak memory at most sizes (65.6 against 71.1 GB for default at 2,000 tokens). They do not extend reach, which the model’s 2,048-token cap limits.
- **Persistent process state (outside the network clock).** Default reloads its six shipped modules inside every prediction call (about 20 s per complex). The optimized package instead loads the weights and the conformer library once per process and runs feature preparation for the next item and output writing on a helper thread. These changes shorten the whole prediction call and the whole pass, not the network time on which the speed-ups in this section are measured.

Results. Fast and Big remain the fastest configurations across the whole set of input sizes, well ahead of Exact, with Fast reaching a geometric-mean speed-up of 6.43× over default across the seven measured input sizes. Peak memory shows an unusual pattern for this model: Exact and Fast use more memory than default at every input size – reaching about 1.74× default’s peak at some sizes – most likely because they hold sampler tensors for all five samples, a CUDA-graph memory pool, and cached allocator reservations (inferred from the design, not measured), even as they save time. Big, in contrast, trims memory below default at most input sizes by dropping that graph pool and blocking large intermediate tensors. Default and Big both complete 2,000 tokens, the largest size tested below the model’s 2,048-token cap (larger inputs were refused by the model, not by memory limits); Exact and Fast were not searched to the same limit.

Accuracy on FoldBench-Lite. On the 234 interfaces of 150 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 39.7% of interfaces under default and for 41.0%, 39.7%, and 39.7% under Exact, Fast, and Big (Figure S54). The only paired change from default whose 95% confidence interval excludes zero is a change of +0.012 in mean protein-monomer lDDT under Exact. Confidence scores track default closely (Figure S55): across targets, the model’s ipTM, pTM, and mean pLDDT under each mode correlate with default at $r = 1.00$ (to two decimals), and the mean paired differences are at most 0.002 for ipTM and pTM and at most 0.04 points for mean pLDDT.

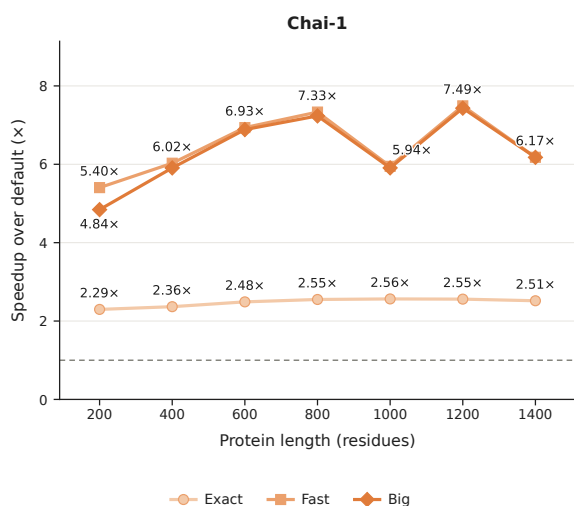


Figure S51. Chai-1: speed-up over default vs. input size. Each line shows one mode’s (Exact, Fast, Big) speed-up on the network clock – per-complex time of the trunk, diffusion sampler, and confidence module, one complex per call on a single H100 GPU – relative to default (dashed line, 1×) at the same input size. Big’s values at 400–1,400 tokens (5.90×, 6.88×, 7.23×, 5.90×, 7.43×, and 6.17×) are hidden under Fast’s labels.

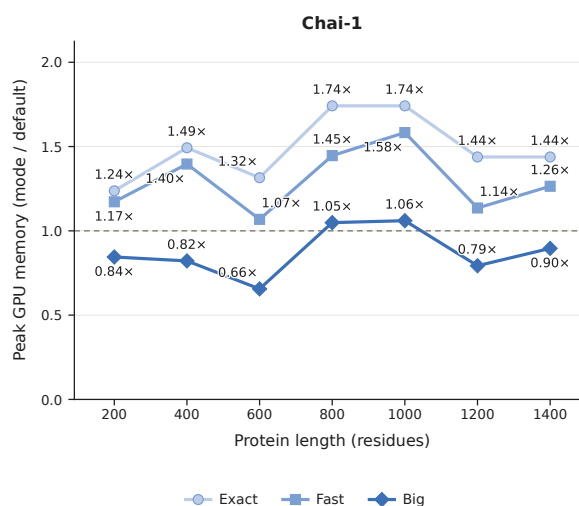


Figure S52. Chai-1: peak GPU memory relative to default. Each line shows one mode’s (Exact, Fast, Big) peak device memory (NVML high-water mark above an idle baseline, from dedicated memory runs) as a ratio to default’s peak (dashed line, 1×) at the same input size.

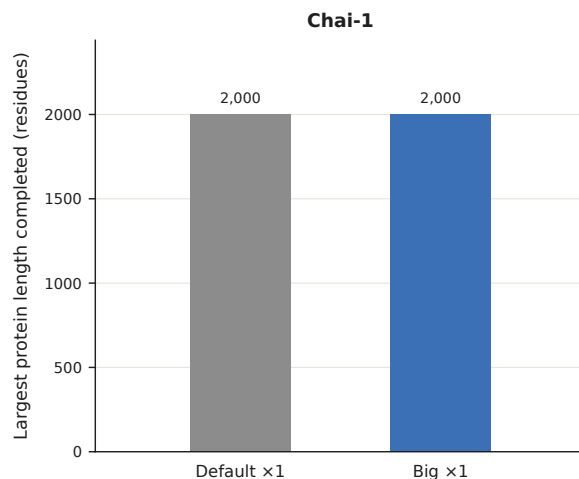


Figure S53. Chai-1: largest input completed. Bars show the largest input size, in tokens, that default and Big complete on a single H100 GPU; inputs above the model's 2,048-token cap are refused by the model itself rather than failing from memory exhaustion.

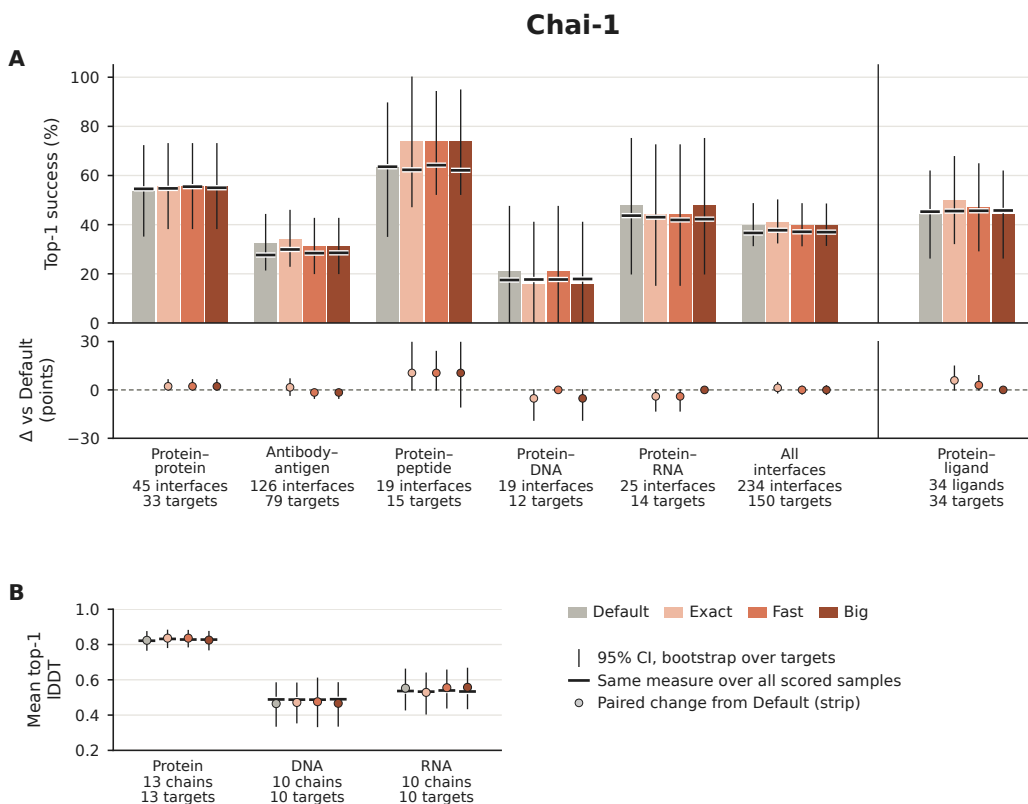
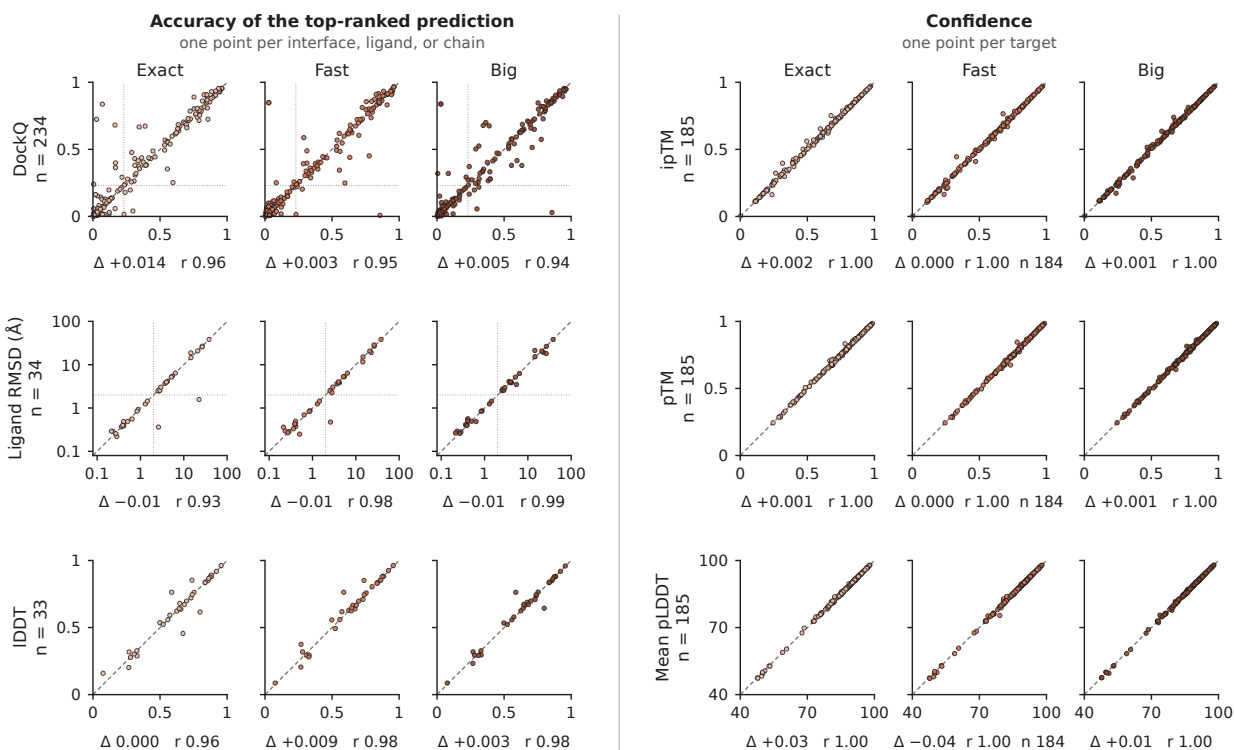


Figure S54. Chai-1: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

Chai-1



x = Default, y = mode; dashed line, y = x

Figure S55. Chai-1: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, y = x. Left, accuracy of the top-ranked prediction (the predictions of Figure S54), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right). One target has no scored Fast predictions, so Fast's n is 184.

S1.12 RoseTTAFold3

RoseTTAFold3 is an all-atom biomolecular structure predictor that folds proteins, nucleic acids, ligands, and modified residues. Benchmarks used the upstream package’s command-line interface `rf3 fold` (RosettaCommons/foundry, package `rc-foundry`), with the released `rf3_foundry_01_24_latest_remapped` checkpoint [43].

Benchmark setup. The benchmark folds single complexes at seven fixed input sizes (200 to 1,400 tokens in steps of 200), one complex per forward call. Each timed pass folds several copies of the complex at a given size (14 copies at 200/400 tokens, 8 at larger sizes). A few early copies serve as warm-up and are excluded from the timed statistic, leaving 12 or 6 steady items per pass. Each input size was measured in three replicate jobs that timed default before and after the modes (Exact in separate jobs). Each mode is compared with the mean of the two default passes. All runs used H100 80 GB GPUs, one per run except for Big on two GPUs. The headline speed-up is measured on the forward (prediction) call itself, timed with GPU synchronization. This call covers all recycling steps, the diffusion sampler, and the confidence head. A per-item wall clock (which also includes input parsing and output writing) and a whole-pass wall clock were also recorded. The whole-pass speed-ups of Exact (1.51–1.68×) and Fast (1.35–2.62×) are smaller than their forward-call speed-ups. Big’s whole-pass speed-up (1.35–1.75×) exceeds its forward-call speed-up at 200 and 400 tokens and is close to it from 600 tokens up. Peak GPU memory is the device’s peak allocation above its idle baseline. This reading varied by as much as 63% between otherwise identical default runs at one input size because it also includes the memory allocator’s own reserved pool. Memory ratios are therefore always computed within the same run. For this model, default (the benchmark baseline) is the unmodified upstream package with two settings applied to every mode: 11 recycling passes instead of the shipped 10, and the data-dependent early-stopping exit turned off. Every mode therefore does the same fixed amount of work regardless of model confidence. Base (the package run exactly as shipped) instead keeps early stopping active, so its runtime depends on each input’s own confidence and is not directly comparable. No separate measurement of base is reported.

Sampling settings. Default and every mode used the same sampling settings: 11 trunk passes (upstream default 10), 5 diffusion samples of 50 steps each (`diffusion_batch_size=5`, `num_steps=50`), and early stopping disabled (`early_stopping_plddt_threshold=0`; upstream default 0.5), so that every call does the same work. Inputs are precomputed MSAs for each chain, with no templates; each call predicts one complex.

Modes. Exact, Fast, and Big all exist for this model, and Big is also run on two GPUs. Exact was verified to give bit-identical outputs to default under fixed, deterministic run settings, confirmed on a 32-case set of representative inputs. Fast allows reduced-precision or approximate kernels within a stated accuracy budget, so it is checked against default’s own run-to-run (seed-to-seed) variation instead of exact identity. Fast passed this check, as did Big on one GPU and on two GPUs.

Optimizations. The optimized package leaves the model’s outputs unchanged in Exact and adds the changes below, the first three in decreasing order of estimated contribution to Exact’s speed-up:

- **CUDA-graph capture of the diffusion roll-out.** Each set of denoising steps is recorded once as a CUDA graph, a sequence of GPU operations that is captured once and replayed without re-issuing each operation from Python. This graph capture removes per-step launch overhead in the sampler and is the largest single contributor to Exact’s speed-up.
- **Step-invariant conditioning computed once.** Pair and atom-encoder conditioning tensors that do not change across denoising steps are computed once per roll-out and reused instead of being recomputed at every step. This reuse and the graph capture above together account for most of Exact’s gain.
- **Trunk capture per input size.** The 48-block pairformer trunk is captured once for a given input size and replayed across recycling steps rather than re-launched each time. The benefit is moderate and matters most on smaller inputs.
- **Fused kernels (Fast, Big).** Triangle multiplication, triangle attention, and the diffusion token transformer are each replaced by a single fused kernel that combines several GPU operations into one launch. The fused kernels cut the number of separate launches in the trunk and sampler and give most of Fast’s gain over Exact.
- **Memory-shaped kernels (Big).** Atom-pair conditioning is built directly in a local-window form, several operations are processed in row blocks, and some intermediate data are staged in host memory. These changes trade some speed, with CUDA graphs turned off, for a much higher input-size ceiling.

Results. Fast is the fastest mode at every input size on the forward-call clock, reaching 3.89× default at 1,400 tokens. Exact gives smaller gains: 2.6× at 200 tokens and 1.7–1.8× at larger sizes. Big is faster than default at every input size tested except the smallest, where it is slower because its CUDA-graph-free kernels run comparatively slowly. Its main benefit, however, is memory: peak GPU memory falls to 0.46× default at 1,400 tokens. This memory saving lets Big complete a 4,000-token complex on a single H100, twice default’s largest completed input (2,000 tokens). Big’s reach extends further still on two GPUs. Exact and Fast were not put

through a dedicated reach search and are only confirmed to work up to the largest input size used for speed testing. Exact's 0.72× memory readings at 1,000–1,400 tokens are not a saving. Default peaked higher in Exact's separate run than in the main run, and at these sizes Exact allocates 1.14–1.16× as much tensor memory as default does in the main run.

Accuracy on FoldBench-Lite. On the 226 interfaces of 138 FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 53.1% of interfaces under default and for 50.9%, 53.5%, and 49.6% under Exact, Fast, and Big (Figure S59). The only paired change from default whose 95% confidence interval excludes zero is a change of +0.008 in mean RNA-monomer IDDT under Exact. Exact is bit-identical to default under deterministic settings (Modes), but these predictions come from a separate production-settings run (p. 16) whose samples differ from default's for all 166 targets with an interface or ligand, so this change and the other Exact differences in Figure S59 most likely reflect run-to-run variation rather than the Exact optimizations. Confidence scores track default closely (Figure S60): across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r = 1.00$ (to two decimals), and the mean paired differences are below 0.0005 for ipTM and pTM and below 0.005 points for mean pLDDT.

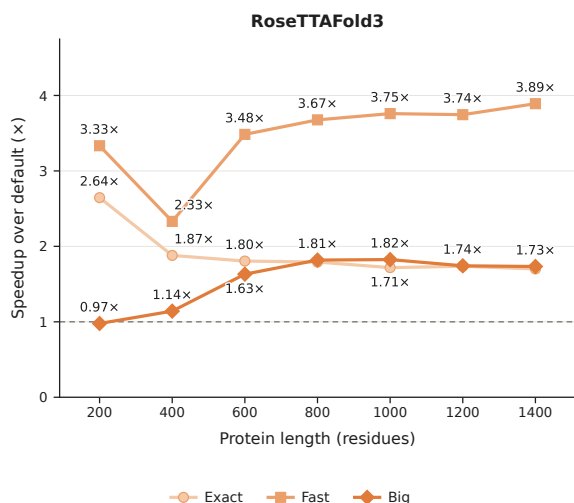


Figure S56. RoseTTAFold3: forward-call speed-up over default. Speed-up (default's forward-call time divided by each mode's) versus input size, one line per mode (Exact, Fast, Big); the dashed line marks default (1×). Exact's values at 800, 1,200, and 1,400 tokens (1.79×, 1.73×, 1.70×) are hidden under Big's labels.

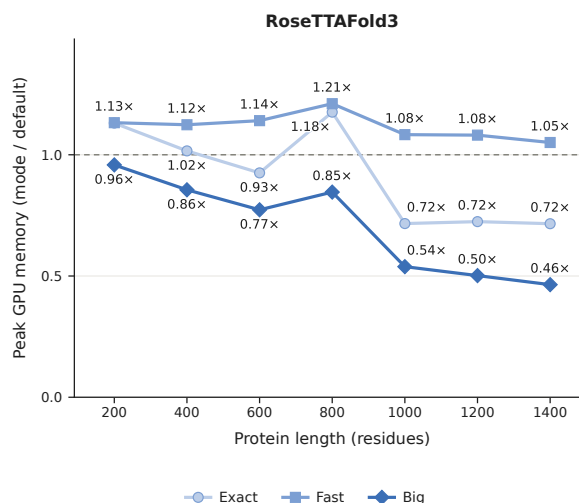


Figure S57. RoseTTAFold3: peak GPU memory relative to default. Peak device memory (NVML high-water mark above idle) from dedicated memory runs, each mode divided by default measured in the same run, versus input size, one line per mode (Exact, Fast, Big); the dashed line marks default (1×). Exact's readings below 1× reflect higher default peaks in its separate run, not a saving (Results).

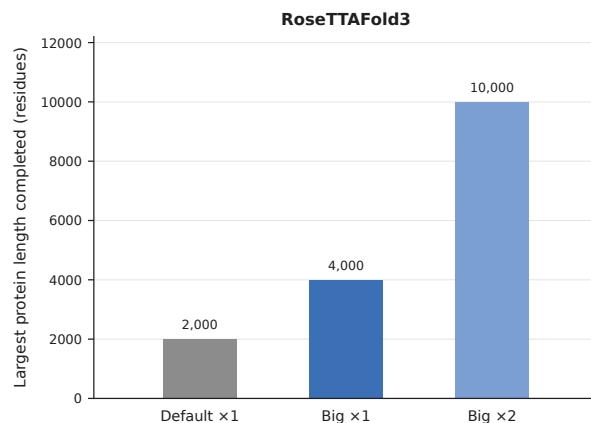


Figure S58. RoseTTAFold3: largest complex completed. Largest input size (tokens) that completes without running out of GPU memory, one bar per configuration: default on one GPU, Big on one GPU, and Big on two GPUs. The two-GPU value comes from shortened runs with two recycling steps (p. 16).

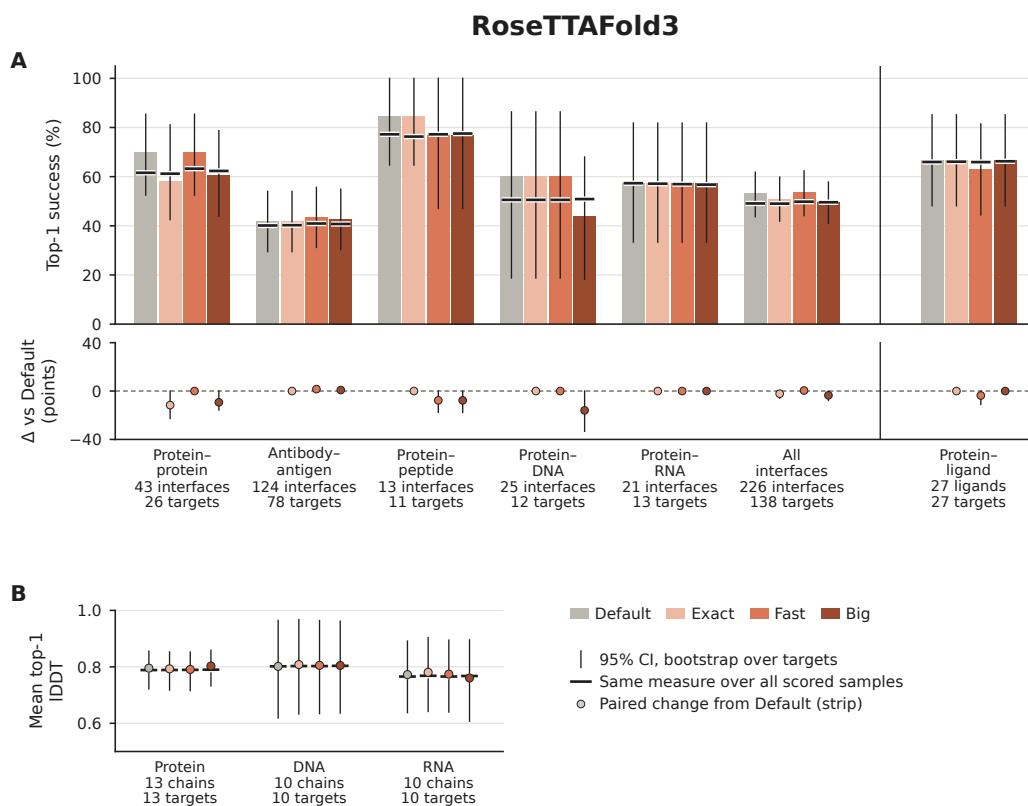
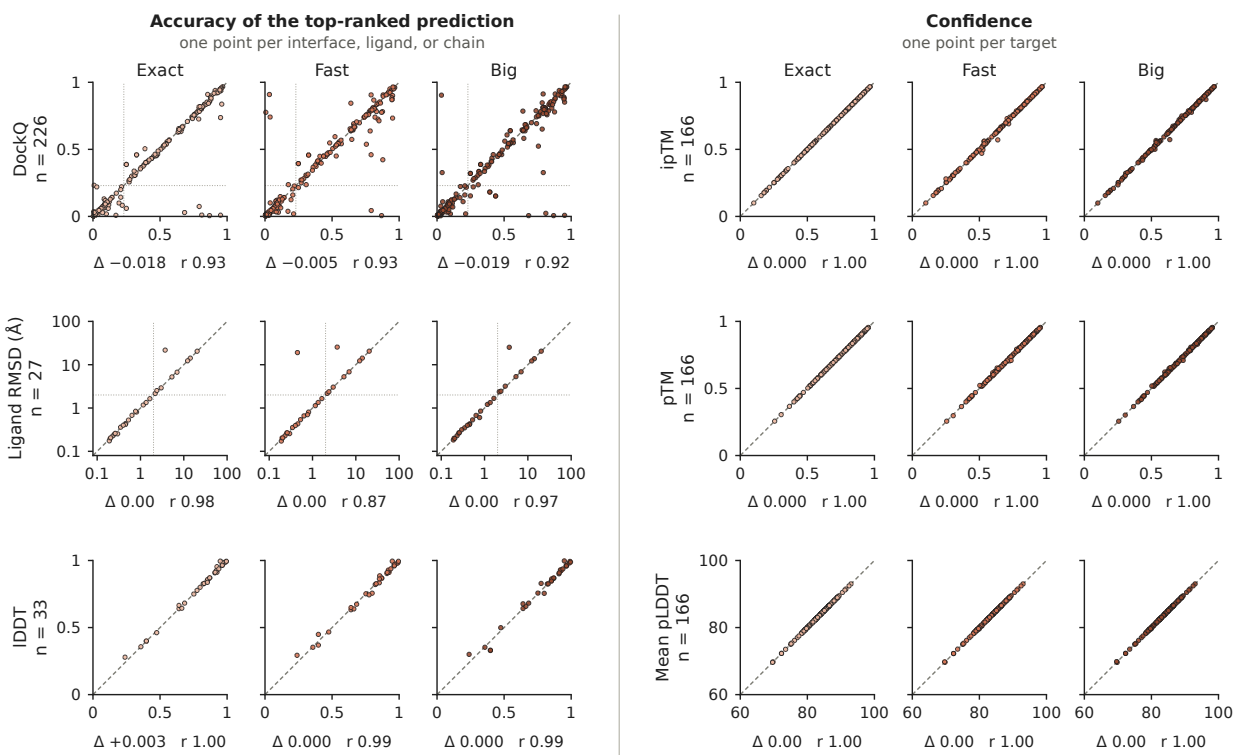


Figure S59. RoseTTAFold3: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23 ; for ligands, ligand RMSD below 2 Å), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein, DNA, and RNA monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces, ligands, or chains scored under default and every mode, and their targets (p. 16).

RoseTTAFold3



x = Default, y = mode; dashed line, y = x

Figure S60. RoseTTAFold3: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S59), one point per interface (DockQ), ligand (ligand RMSD, logarithmic axes), or protein, DNA, or RNA chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance thresholds (DockQ 0.23; ligand RMSD 2 Å). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default; for ligand RMSD, the median, in Å); r , Pearson correlation (for ligand RMSD, of the logarithms); n , number of interfaces, ligands, or chains (left) or targets (right).

S1.13 AtlasFold

AtlasFold is a single-sequence protein co-folding model that combines a protein language model (AtlasLM-3B) with a 48-block AlphaFold-3-style Pairformer trunk and diffusion module, producing atomic structures and confidence scores from a single FASTA record with no MSA or template input. The benchmarked release is AtlasFold v1.0.0 [44].

Benchmark setup. The task folds homomeric protein crops at seven input sizes from 200 to 1,400 tokens, padded to upstream’s own length buckets. Upstream’s own packing combines same-bucket targets into one forward call (four targets at 200 tokens, two at 400, one from 600 tokens up) identically on every configuration. Each input size was measured over three replicate passes on a single H100 80 GB GPU, excluding the first (warm-up) target from all timings. The headline speed-up is the CUDA-event time of the model’s forward call (language model, trunk, diffusion sampler, and confidence head) per target. Peak GPU memory is the device-level NVML high-water mark over a dedicated memory-only pass, above an idle baseline. For this model default equals base. The upstream package runs through its documented command line at its own sampling settings (below), with bucketing and packing left untouched on every configuration to keep the comparison fair to upstream’s own batching behavior.

Sampling settings. Default and every mode used the same sampling settings, which are upstream’s defaults: 10 recycles (11 trunk passes) and 5 diffusion samples of 200 steps each per target, with the confidence head evaluated on every sample. Predictions use the multimer head (AtlasFold-M with the AtlasLM-3B protein language model) on single-sequence input; the model takes no MSAs or templates. Targets are packed by upstream’s token budget (`--max-tokens-per-batch 1024`): 4 targets per call at 200 tokens, 2 at 400 tokens, and 1 from 600 tokens up.

Modes. Exact, Fast, and Big are all available. Exact keeps default’s numerical path while fusing kernels and removing redundant per-step work. It was verified bit-identical to default under a deterministic setting (4 of 4 identity checks passed). Fast and Big trade some exactness for further kernel fusion and reduced-precision arithmetic; both were checked against default’s own seed-to-seed variation and stayed within it (2 of 2 checks passed for each). Big reuses Fast’s kernel set but omits the CUDA-graph capture, trading some speed for lower peak memory at 200–1,000 tokens; it does not reach larger inputs than Fast.

Optimizations. The optimized package leaves recycling steps, diffusion steps, sample count, bucketing and packing untouched on every mode, and instead changes:

- **Sampler work computed once per roll-out and fewer host synchronizations.** Tensors constant across a diffusion roll-out (pair bias, atom-encoder, and single conditioning) are computed once per sampling call rather than at every denoising step, and the loop’s per-step host synchronizations are collapsed to once per roll-out. This change is the largest single contributor to Exact’s speed-up.
- **CUDA-graph denoiser capture/replay (Fast only).** The diffusion denoising loop is recorded once as a CUDA graph (the GPU operations captured once and replayed with less launch overhead) for inputs of up to 1,024 tokens, making Fast 1.06–1.50× as fast as Big, which does not use this optimization, at 200–1,000 tokens.
- **Fused triangle-multiplication and triangle-attention block kernels (Fast, Big).** Most of the gain on large inputs, where the trunk’s cost dominates, comes from triangle multiplication running through fused kernels chosen per card and input size and, from 512 tokens, each triangle-attention block running through three fused kernels. Exact uses bit-identical fused kernels for triangle attention and SwiGLU transitions and de-duplicates atom-attention keys.
- **Reduced-precision diffusion and attention kernels (Fast, Big).** The diffusion transformer runs under bfloat16 autocast, and the atom encoder-decoder uses fused SDPA and TF32 kernels, trading exactness for throughput.
- **Memory-only optimizations (all optimized modes).** Streamed PAE output, chunked pair transitions, lazy relative-position recompute, offloaded distogram logits, and an expandable allocator cut peak memory without changing forward-call timing, letting Fast and Big reach larger inputs than default on one GPU.

Results. Forward-call speed-ups for Fast and Big grow with input size, since the trunk’s triangle operations dominate at large sizes, and level off from 800 tokens for Fast (2.98×, then 2.94–2.95×) and from 1,200 tokens for Big (2.94×, after a dip to 1.79× at 600 tokens). Gains are smaller at short inputs, where a fixed language-model/confidence cost limits the benefit. Exact, which keeps default’s exact numerical path, is consistently the slowest of the three modes. Default’s peak memory grows sharply, reaching 38.8 GB at 1,400 tokens, while the optimized modes stay comparatively flat, letting Fast and Big fold inputs of up to 4,000 tokens on one GPU, twice default’s limit. The one reading above default, Exact at 200 tokens (1.47×), comes from memory growing across successive four-target calls in one process, which two-target packing (`--max-tokens-per-batch 512`) avoids.

Accuracy on FoldBench-Lite. On the 197 interfaces of 124 protein-only FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable ($\text{DockQ} \geq 0.23$) for 52.3% of interfaces under default and for 52.3%, 52.8%, and 52.8% under Exact, Fast, and Big (Figure S64). No paired change from default, for any interface class or for protein monomers, has a 95% confidence interval that excludes zero. Confidence scores track default closely (Figure S65); across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.99$, and the mean paired differences are at most 0.003 for ipTM, at most 0.002 for pTM, and at most 0.05 points for mean pLDDT.

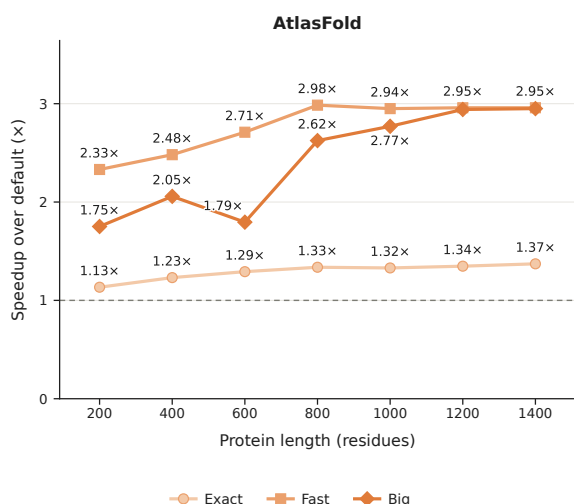


Figure S61. AtlasFold: forward-call speed-up over default. Speed-up on the forward-call clock versus input size on H100, one line per mode (Exact, Fast, Big); the dashed line marks default (1 $\times$). At 1,200 and 1,400 tokens Big (2.94 $\times$) lies under Fast's labels.

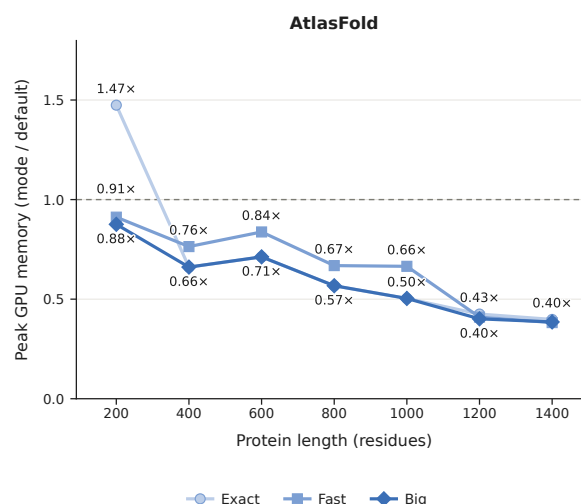


Figure S62. AtlasFold: peak GPU memory relative to default. Peak device-level NVML memory (above idle baseline) for each mode relative to default, one line per mode (Exact, Fast, Big), versus input size on H100, from dedicated memory-only passes; the dashed line marks default (1 $\times$). Exact's line is hidden under Big's at 400–1,000 tokens (equal to two decimals); unlabeled are Fast at 1,200 and 1,400 tokens (0.41 $\times$, 0.38 $\times$) and Big at 1,400 tokens (0.39 $\times$).

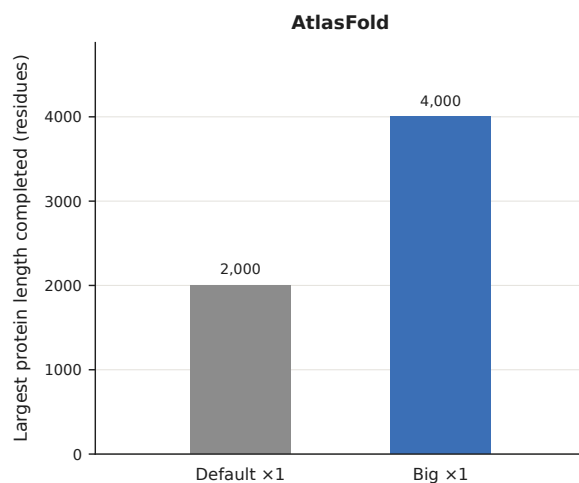


Figure S63. AtlasFold: largest input completed. Largest input size that completes on one H100 GPU, one bar per configuration (Default $\times 1$, Big $\times 1$). Fast, not shown, completes the same 4,000 tokens as Big.

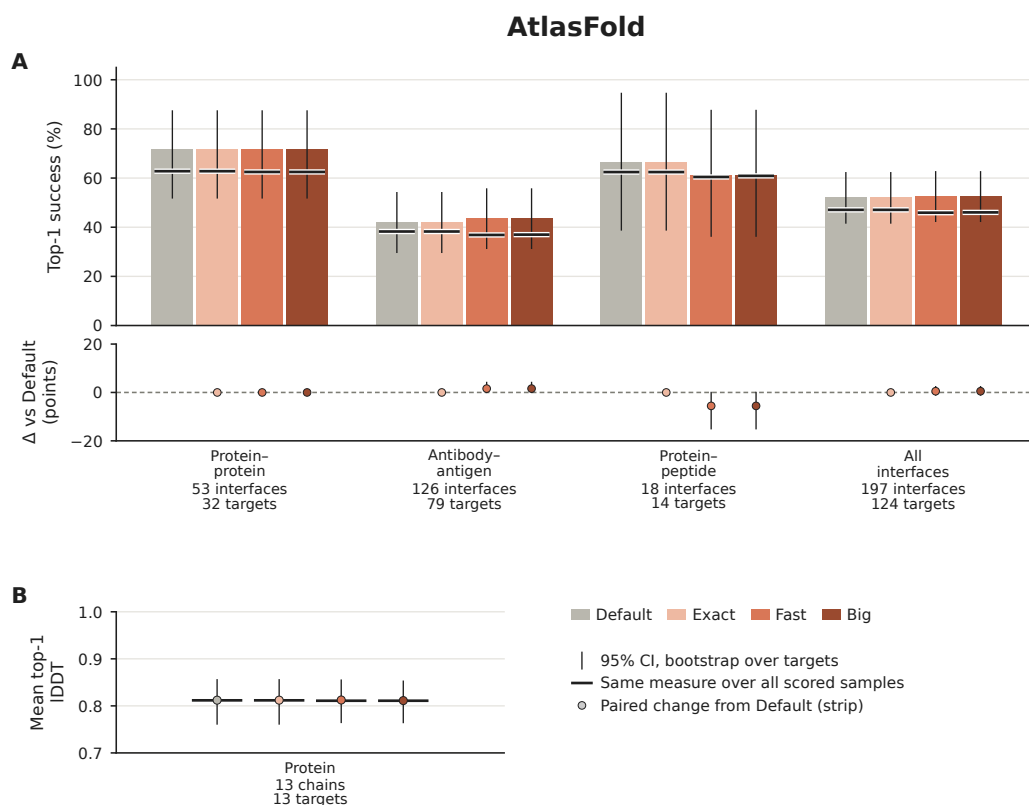


Figure S64. AtlasFold: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces or chains scored under default and every mode, and their targets (p. 16). AtlasFold models proteins only and was run on protein-only inputs.

AtlasFold

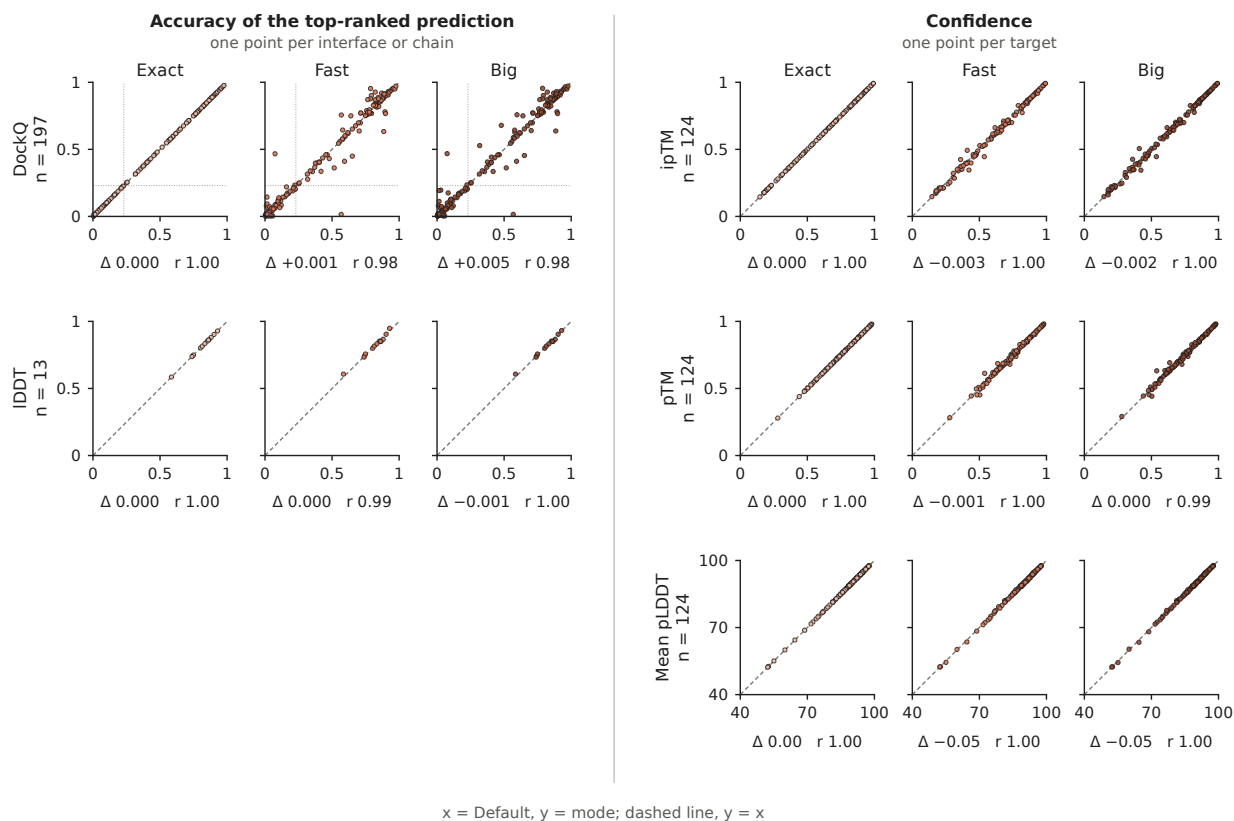


Figure S65. AtlasFold: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S64), one point per interface (DockQ) or protein chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance threshold (DockQ 0.23). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default); r , Pearson correlation; n , number of interfaces or chains (left) or targets (right).

S1.14 ColabFold (AF2-Multimer)

ColabFold’s batch predictor `colabfold_batch` runs AlphaFold2-Multimer v3 to predict the structure of a multi-chain complex from a multiple-sequence alignment. The benchmark used the unmodified upstream wheels `colabfold` 1.6.1 and `alphafold-colabfold` 2.3.13 together with the 2022-12-06 AlphaFold-Multimer v3 parameters [45, 59]. Later ColabFold releases add optional fused kernels of their own. Version 1.6.2 (July 2026) introduced optional fused Pallas/Triton Evoformer kernels, reported to accelerate predictions by about 2.5 times [33], and version 1.6.3 (September 2026) renamed the option [34]. Speed-ups against a current release with fused kernels enabled are likely to be smaller than those reported below, because default here does not use these kernels and we have not benchmarked either release.

Benchmark setup. The task is complex structure prediction on real multi-chain complexes cropped to exactly 200, 400, 600, 800, 1000, 1200, and 1400 tokens, using precomputed a3m alignments from ColabFold’s standard MSA search as input. `colabfold_batch` has no batch axis, so each pass predicts complexes one at a time: 14 items per pass at the two smallest sizes and 8 at the larger sizes. Every configuration uses one model instead of upstream’s five and a fixed 20 recycles (21 passes through the network) with early stopping disabled, so all configurations do the same amount of work and ratios reflect optimization rather than differing work. Values are medians over items and over three replicate jobs per input size on one H100 80 GB GPU, with a warm-up copy of each shape excluded from every statistic. The headline speed-up uses the forward-call clock, the time `colabfold_batch` itself reports for the model call of one complex, excluding alignment reading, output writing, start-up, and compilation. Peak GPU memory is the device’s high-water mark above an idle baseline, with the memory allocator in growth mode. Because that allocator grows in doubling steps, the measurement reflects pool-size steps rather than bytes actually in use. Default here is the package as shipped, but with the memory-environment settings changed (unified memory off, allocation fraction 0.95) in place of the shipped, heavily over-subscribed unified-memory pool, because the shipped setting does not make forward progress on an 80 GB H100. Apart from this and the shared one-model, fixed-recycle settings above, no change is applied, and for the same reason base (the package run exactly as shipped) could not be measured.

Sampling settings. Default and every mode used the same sampling settings: one AlphaFold2-Multimer v3 model per call (`--num-models 1`; upstream default 5) and 20 recycles (21 trunk passes) with early stopping disabled (upstream stops at a tolerance of 0.5). Inputs are precomputed MSAs read from A3M files, without an MSA server; templates and relaxation are disabled, and each call predicts one complex.

Modes. The optimized package provides three modes: Exact, Fast, and Big (Big is also run on two GPUs, for reach). Exact changes only where data is kept: it holds parameters, input features, and the recycling state on the GPU throughout the 21-pass recycle loop and fetches outputs once at the end instead of after every pass. This was verified to give bit-identical outputs to default under the deterministic settings that upstream itself needs for run-to-run reproducibility. Fast is the optimized package’s own standard configuration and adds the fused-kernel changes described below. At one GPU it runs the identical program as Big, so the speed, memory, and reach figures show Big only, and the accuracy and confidence figures show Fast only. Fast’s kernel fusions trade some numerical precision for speed and are checked against default’s own seed-to-seed variation rather than required to match bit-for-bit; Big’s outputs, at one GPU, likewise stay within that same variation. Big is the package’s memory-optimized configuration. On two GPUs it additionally splits the pair representation across the two cards, which lowers per-GPU memory and switches off three of the changes below (fused triangle multiplication and transitions, template de-duplication). Its outputs also stay within default’s seed-to-seed variation.

Optimizations. The optimizations mainly target the recycle loop’s compute-heavy attention and matrix operations:

- **Fused pair-biased attention.** Fusing triangle attention and MSA row attention into a single kernel that never materializes the full token-by-token attention matrix is the largest single contributor to the speed-up, avoiding the cost of storing and moving that matrix.
- **Fused triangle multiplication.** Several small matrix operations (normalization, gated projections, contraction) are combined around one larger batched matrix multiply.
- **Larger attention/transition sub-batch.** The number of rows processed together is raised from 4 to 128 whenever a memory estimate shows the run’s largest input fits the card, improving GPU utilization.
- **Fused transition kernel.** The LayerNorm→Linear→ReLU→Linear sequence of each transition block is merged into one kernel call.
- **Fused MSA attention and template de-duplication.** MSA column and extra-MSA row attention run through fused kernels, and a template-free run embeds its four identical placeholder template rows once instead of four times.

- **Device-resident placement.** Parameters and the recycling state stay on the GPU across all 21 passes instead of being copied to and from the host every pass, with outputs fetched once at the end. This is Exact's only change and is also used inside Fast and Big.

Results. Big gives the largest speed-up, roughly flat with input size, with a geometric mean of 4.23 $\times$ over the input sizes, while Exact's placement-only change gives a smaller speed-up of 1.11 $\times$ from reduced host-device traffic in the recycle loop. Peak GPU memory is essentially unchanged across configurations at a given input size, consistent with the coarse allocator-step resolution of the memory measurement. Default and Big both complete 4,000 tokens on one H100 and run out of memory at 5,000; splitting the pair representation for Big across two GPUs lets it complete 5,000 tokens, a value found with shortened runs (Figure S68).

Accuracy on FoldBench-Lite. On the 196 interfaces of 124 protein-only FoldBench-Lite targets scored under default and every mode (p. 16), the top-ranked prediction was acceptable (DockQ ≥ 0.23) for 38.3% of interfaces under default and for 39.8% and 39.3% under Exact and Fast (Figure S69; Fast runs the same program as Big on one GPU). No paired change from default, for any interface class or for protein monomers, has a 95% confidence interval that excludes zero. Exact is bit-identical to default only under deterministic settings (Modes); its sampled predictions here differ from default's for 116 of the 124 targets with an interface, so its 1.5-point gain reflects run-to-run variation, not the Exact optimizations. Confidence scores track default closely (Figure S70): across targets, the model's ipTM, pTM, and mean pLDDT under each mode correlate with default at $r \geq 0.98$, and the mean paired differences are at most 0.002 for ipTM and pTM and at most 0.12 points for mean pLDDT.

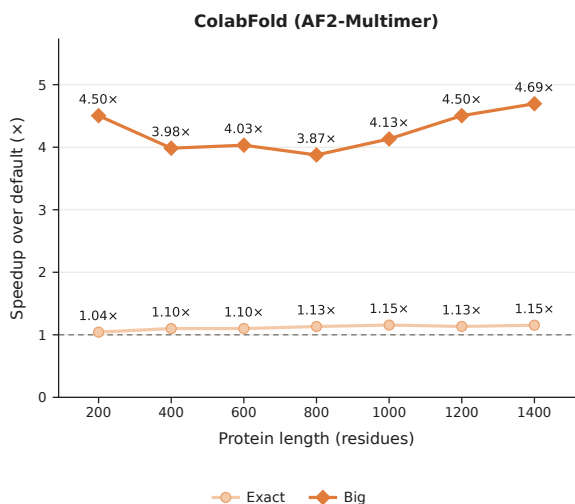


Figure S66. ColabFold: forward-call speed-up over default. Speed-up on the model's own forward-call clock (default time divided by mode time) vs input size (tokens) on H100, one line per mode (Exact, Big); the dashed line marks default (1 $\times$).

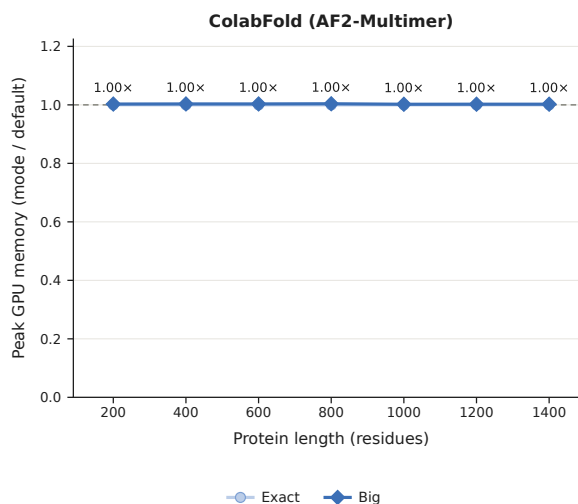


Figure S67. ColabFold: peak GPU memory relative to default. Peak device memory (high-water mark above idle) for each mode relative to default vs input size (tokens) on H100; the dashed line marks default (1 $\times$). Because the memory allocator grows in doubling steps, near-identical ratios can mask real differences in memory actually used. Exact and Big coincide at 1.00 $\times$, so the Exact line is hidden under Big.

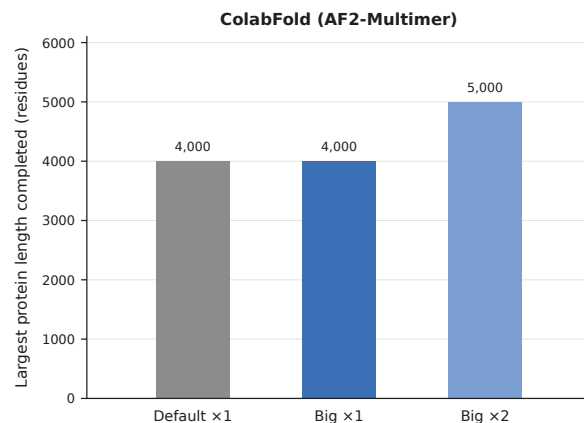


Figure S68. ColabFold: largest complex completed. Largest input size (tokens) that completed, one bar per configuration (Default $\times 1$, Big $\times 1$, Big on two GPU(s)). The one-GPU searches used the benchmark's fixed settings; the two-GPU search used shortened runs with a single recycling step (two passes), whose peak memory at 4,000 tokens equaled that of a full-settings run.

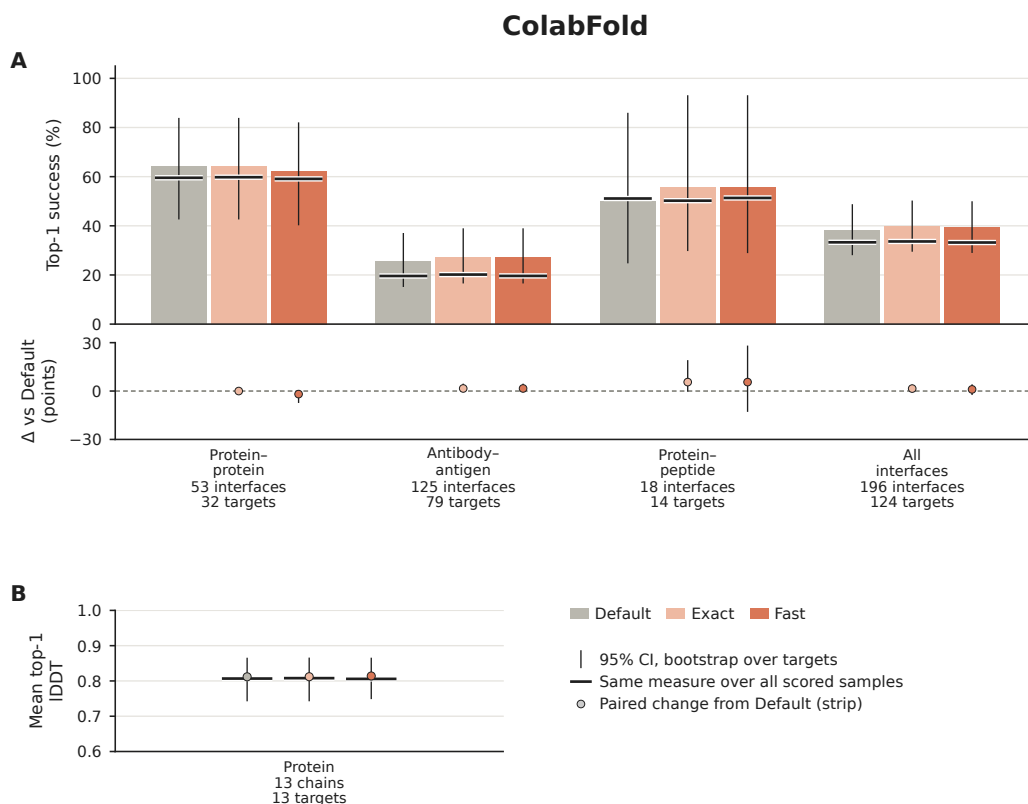


Figure S69. ColabFold: accuracy on FoldBench-Lite under each mode. **A**, Share of interfaces whose top-ranked prediction is acceptable (bars; DockQ ≥ 0.23), with 95% bootstrap intervals (vertical lines); short horizontal lines, the share of all scored samples that are acceptable, averaged over the same units. The strip below shows each mode's paired change from default in percentage points, with its 95% interval. **B**, Mean IDDT of the top-ranked prediction for protein monomers (points), with 95% intervals; short horizontal lines, the median IDDT of all scored samples, averaged over chains. Counts are the interfaces or chains scored under default and every mode, and their targets (p. 16). ColabFold models proteins only and was run on protein-only inputs. Fast runs the same program as Big on one GPU (Modes).

ColabFold

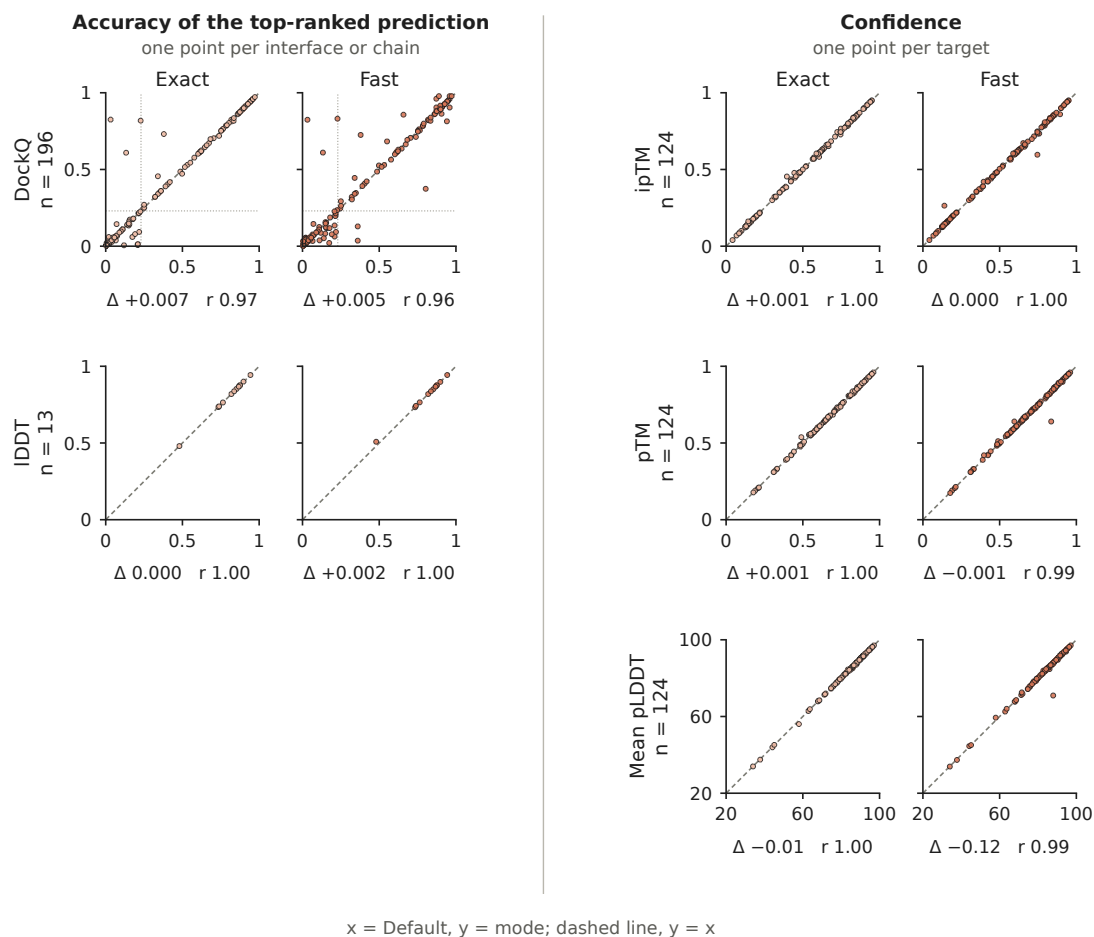


Figure S70. ColabFold: accuracy and confidence of each mode compared with default. Each point pairs default (x) with the mode (y); dashed line, $y = x$. Left, accuracy of the top-ranked prediction (the predictions of Figure S69), one point per interface (DockQ) or protein chain (IDDT), for the units scored under default and every mode; dotted lines mark the acceptance threshold (DockQ 0.23). Right, the model's confidence scores, one point per target with at least one interface: ipTM, pTM, and mean pLDDT, each taken as the median over the samples of a seed and then averaged over seeds. Δ , mean paired difference (mode minus default); r , Pearson correlation; n , number of interfaces or chains (left) or targets (right). ColabFold models proteins only and was run on protein-only inputs. Fast runs the same program as Big on one GPU (Modes).

S1.15 AF2 initial guess

AF2 initial guess re-predicts a designed binder–target protein complex using AlphaFold 2 (model_1_ptm weights, the JAX/Haiku code vendored inside the upstream package). It takes the design itself as the structural initial guess and scores the resulting interface. We benchmarked the `af2_initial_guess` step of the `dl_binder_design` repository, pinned at the commit dated 2024-01-10 [46].

Benchmark setup. Each call takes one binder–target complex PDB and predicts one complex, run sequentially at a batch size of one design. Inputs are drawn from a fixed size ladder of public complexes cropped to 200–1,400 tokens (the 1,400-token size pools a 1,308-token and a 1,400-token complex). Each pass times 12 predictions at 200 and 400 tokens, 6 at 600–1,000 tokens, and 3 at 1,200 and 1,400 tokens, after one untimed warm-up copy of each source complex. Each replicate run starts with an untimed warm-up default pass and then times default, Exact, Fast, Big, and default again on the same GPU (one NVIDIA H100 80 GB, the only hardware used since the package has no multi-GPU support). Each mode is compared with the mean of the two default passes, which controls for host drift. The headline clock is the steady-state, device-synchronized time of the single forward (prediction) call per design, excluding warm-up and compilation. A companion whole-pass wall clock (process start to exit, including input processing and output writing) is reported alongside it. Peak GPU memory is the NVML device high-water mark above idle over the whole pass. Measured peaks can plateau or jump rather than rise smoothly with input size because the JAX allocator grows its memory pool in discrete steps. For this model, default adds a small set of settings applied identically to every mode: a recycle count raised from upstream’s 3 recycles to 10 (11 network evaluations per call instead of 4), a PyRosetta-free PDB front end that leaves the model arithmetic unchanged, a current JAX/cuDNN software stack confirmed to give bitwise-identical outputs to the older stack it replaces, and a clean environment with the optimized package’s code excluded from the import path. Base (the package run exactly as shipped, with 3 recycles, PyRosetta input handling, and the older software stack) was not measured, so all ratios are relative to this configured default. Bracketing the modes with default passes, and reporting only steady-state times, keeps host variance or compilation from inflating the reported speed-ups.

Sampling settings. Default and every mode used the same sampling settings. AlphaFold2 (model_1_ptm, a single model) runs 10 recycles, that is, 11 network evaluations per design (upstream default 3 recycles), and predicts one structure per design, with one design per call. The input is the binder–target complex as a single-sequence alignment, with the target chain supplied as a template; no MSA search is run.

Modes. Exact, Fast, and Big all exist for this model. Exact changes only host, data-path, and compile-path handling, leaves model arithmetic untouched, and was verified bit-identical to default under deterministic settings. Fast is the mode the package runs when no mode is specified. It adds kernel and batching changes with numerics inside a declared tolerance. Fast’s maximum per-case $C\alpha$ -RMSD to default is 0.897 Å, within a stated 1.0 Å gate. Big uses the same numerics tolerance class as Fast. Big leaves out Fast’s 8,192-row sub-batch for template point-wise attention and raises the JAX memory-pool limit from 0.75 to 0.95 of GPU memory, staying within 0.678 Å of default.

Optimizations. The optimized package layers data-path and kernel changes on top of the unmodified model:

- **Wider internal attention sub-batch.** Raises the sub-batch size inside every attention and transition layer from 4 to 128, which improves GPU utilization.
- **Fused triangle-attention kernel.** Merges the normalization, projections, attention core, and gating steps of triangle attention into one fused GPU operation instead of several separate ones.
- **Fused triangle multiplication.** Replaces many small operations with fused pre-processing, one large matrix multiplication, and fused post-processing.
- **Flash attention and outer-product fusion.** Applies a memory-efficient attention algorithm to the remaining pair-biased attention, chiefly the model’s MSA row attention, and re-associates the outer-product-mean step into one matrix multiplication.
- **Data-path and compile-path handling (Exact and above).** Keeps model parameters resident on the GPU across calls instead of resending them, and moves each design’s output to the host in one transfer. Runs input processing and output writing on background threads concurrently with GPU computation, and precompiles every distinct input length ahead of the timed loop.

Results. Fast and Big are both faster than default on the forward-call clock across the input-size ladder, with geometric-mean speed-ups of 4.96× for Fast and 4.43× for Big. Exact shows essentially no change on this clock because its changes affect data movement and compilation rather than the forward call itself. Exact instead speeds up by up to about 1.44× over default on the whole-pass clock, which also counts input processing, output writing, and compilation. The forward-call clock cannot capture this benefit. Peak GPU memory matches default for Exact at every input size except 1,200 tokens, for Fast up to 600 tokens, and for

Big up to 800 tokens (Figure S72). The device peak moves in steps at larger sizes because the memory allocator grows its pool in doubling regions that it never returns. The readings above default (Exact at 1,200 tokens, 1.27 $\times$; Fast at 800 and 1,200 tokens, 1.30 $\times$ and 1.27 $\times$) reflect an allocation crossing such a step rather than genuinely higher memory use. Fast reads 0.68 $\times$ at 1,000 tokens and 0.67 $\times$ at 1,400 tokens, and Big shows the lowest peaks, 0.43–0.81 $\times$ at 1,000–1,400 tokens. These low readings also partly reflect pool steps: the allocator’s own in-use peak for Fast and Big at these sizes is 0.75–0.86 $\times$ default’s. Big does not extend the reach at the resolution of the reach measurement, which tested only 3,000 and 4,000 tokens. Default and Big both completed 3,000 tokens and ran out of memory at 4,000.

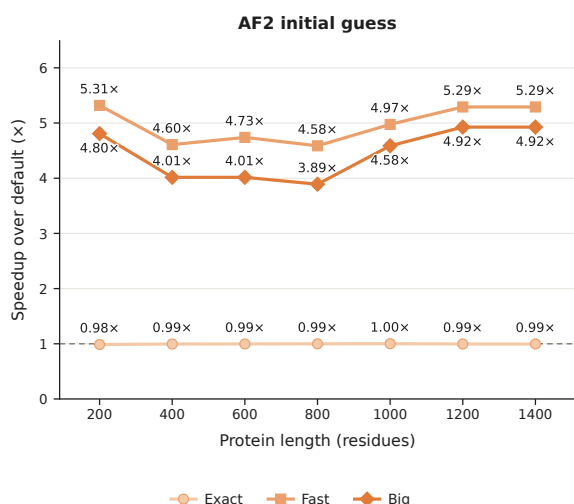


Figure S71. AF2 initial guess: forward-call speed-up over default vs. input size. Each line is one mode (Exact, Fast, Big); the dashed line marks default (1 $\times$). The clock is the steady-state, device-synchronized time of the single forward (prediction) call per complex, at a batch size of one design per call.

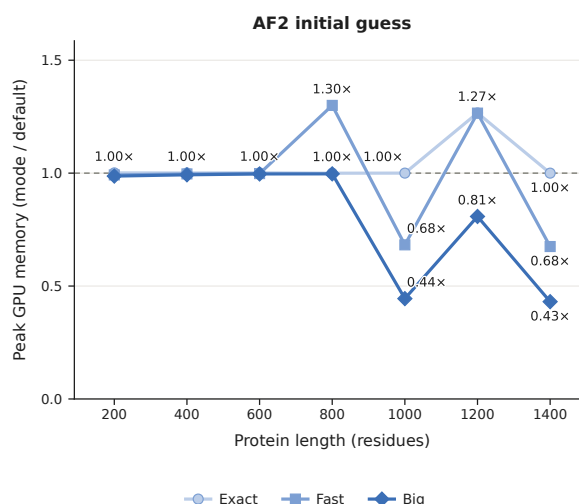


Figure S72. AF2 initial guess: peak GPU memory relative to default vs. input size. Each line is one mode (Exact, Fast, Big); the dashed line marks default (1 $\times$). The measurement is the NVML device high-water mark above idle over the whole pass, from dedicated memory passes that measure one mode at a time at each input size. Where points coincide, one label is printed; the hidden values that differ from it are Fast and Big at 200 tokens and Big at 400 tokens (0.99 $\times$ each).

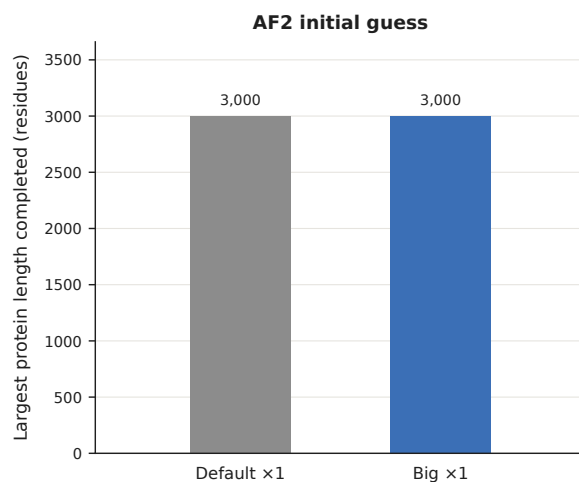


Figure S73. AF2 initial guess: largest input size completed. Bars show the largest complex, in tokens, that default and Big each complete on one GPU; both fail with an out-of-memory error at the next size tested.

S2 Hallucination

These workflows design a protein by repeatedly running a structure predictor and adjusting the sequence by gradient descent so that the predicted structure scores well. Speed-ups are for the time per design step. Section S7 compares the quality of designs made under each mode.

S2.1 ColabDesign

BindCraft’s binder-hallucination design step is built on ColabDesign and optimizes a binder sequence by gradient descent through AlphaFold-Multimer. We benchmarked this step on ColabDesign 1.1.3 (the October 2025 repository version, five commits after the v1.1.3 tag) with the five AlphaFold-Multimer v3 parameter files [15, 47].

Benchmark setup. The task is a single binder-design trajectory: gradient-based optimization of a binder sequence against a target structure. The benchmark covers seven input sizes from 200 to 800 combined target-plus-binder tokens, with two design cases per size. Each GPU call runs one trajectory because the tool cannot batch trajectories together. Each input size was measured on an H100 80 GB GPU over three replicate allocations. Each allocation ran an untimed warm-up pass followed by default, Exact, Fast, and default again at a fixed seed. The headline clock is seconds per optimization step (one forward-plus-backward pass through AlphaFold-Multimer at a fixed recycle count). It is read from the one-recycle soft-stage steps that every trajectory runs before a rule that can later raise the recycle count unevenly across configurations. A companion whole-pass wall-clock time, covering both trajectories of a replicate including compilation and output writing, is reported alongside it. Peak GPU memory is the device high-water mark above an idle baseline, read through NVML during a dedicated pass per configuration with the allocator set to grow on demand rather than pre-allocate. For this model, default is simply the upstream package invoked through its documented interface exactly as shipped, so default equals base and no separate base configuration was measured. Restricting the headline comparison to matching-recycle steps, and running one trajectory per fresh process on every configuration, keeps the comparison to equal amounts of work. BindCraft’s own step-50 rule can otherwise raise the recycle count on one configuration and not another.

Sampling settings. Default and every mode used the same sampling settings, which are BindCraft’s shipped design step: each trajectory runs 75 soft (logit), 45 temperature-annealed (softmax), and 5 hard (one-hot) gradient steps, then 15 greedy refinement rounds, for an 80-residue binder, with one design per call. AlphaFold-Multimer uses 1 recycle, which BindCraft’s own β -sheet check raises to 3 partway through some trajectories, and samples one of the five AlphaFold-Multimer v3 parameter sets at each step (`sample_models`). Inputs are single sequences, with the target structure supplied as a template.

Modes. Exact and Fast are both offered; there is no Big mode, whether memory-lean or on two GPUs. Exact changes only start-up and host-to-device transfer costs and leaves the per-step arithmetic untouched. Exact’s outputs were verified bit-identical to default at the same seed. Fast additionally serves attention, triangle-multiplication, the Transition multilayer perceptron, and other operations through fused kernels. Fast’s outputs are deterministic run to run but not bit-identical to default. They passed a per-design-step check, staying within the spread that default itself shows across seeds.

Optimizations. Fast makes the following changes, ordered by their estimated contribution to its per-step speed-up:

- **Removing the above-384-token sub-batch limit.** The upstream code splits the forward and backward passes into small chunks above 384 tokens to save memory. The optimized package traces both design programs without this chunking, which the attention kernel below makes affordable in memory. This change accounts for most of the gain at larger input sizes.
- **Fused, flash-style attention kernel.** A custom kernel computes attention with pair bias and key masking read inside the kernel, without ever materializing the full attention-score matrix, which saves time and memory. This kernel accounts for most of the gain at input sizes of 384 tokens or below.
- **Fused triangle-multiplication kernel.** AlphaFold’s triangle-multiplication block is fused into a single kernel together with its backward pass, instead of running as several separate operations.
- **Fused Transition multilayer perceptron.** The wide intermediate layer of the per-residue feed-forward block is kept on-chip rather than materialized in memory.

Both modes also keep the recycling features on the GPU between steps, avoiding host-device round trips. This GPU-resident recycling is the only change that speeds up Exact’s steps (by a few percent). A persistent compile cache in both modes spares a later process at the same input size about two minutes of recompilation without affecting the per-step clock.

Results. Both optimized modes speed up every input size measured, but they diverge sharply. Exact gives a small, roughly constant per-step gain (1.02–1.08 $\times$) from the GPU-resident recycling features, and its start-up savings show only in the whole-pass

time (1.07–1.47 $\times$). Fast’s per-step gain rises sharply once the sub-batch limit is crossed at 384 tokens and stays far above Exact’s at larger sizes, reaching a geometric mean of 4.91 $\times$ across the seven input sizes. Fast’s companion whole-pass wall-clock time improves less, at a geometric mean of 3.49 $\times$, because start-up compilation, the greedy rounds, and other host-side work are only partly accelerated. Exact matches default’s peak memory (1.00 $\times$ at every size). Fast needs as little as 0.35 $\times$ of default’s peak at the 700-token input size because its fused attention kernel never materializes the large attention-score matrix that dominates default’s memory use at larger sizes. All configurations complete every benchmarked input size (up to 800 tokens) on a single GPU; larger inputs were not tested.

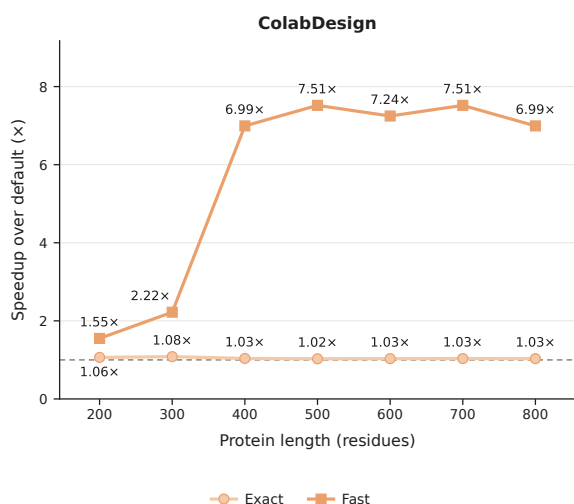


Figure S74. ColabDesign: per-design-step speed-up over default. Speed-up over default in time per optimization step, plotted against input size (200–800 combined target-plus-binder tokens) for Exact and Fast; the dashed line marks default (1 $\times$).

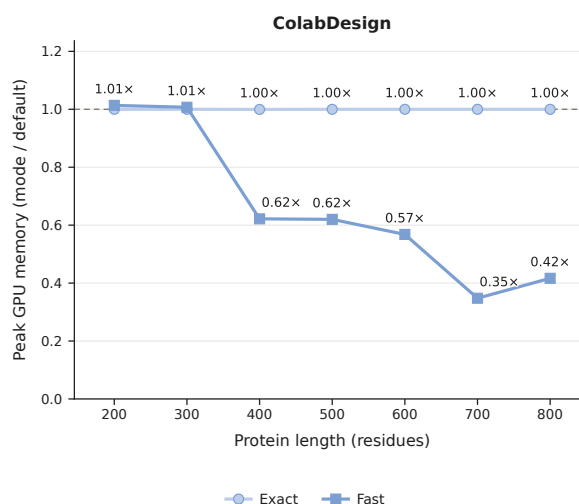


Figure S75. ColabDesign: peak GPU memory relative to default. Peak device memory above idle, measured by NVML in dedicated passes, for Exact and Fast relative to default, plotted against input size; the dashed line marks default (1 $\times$). Exact’s points (1.00 $\times$) lie under Fast’s at 200 and 300 tokens.

S2.2 ESMFold2-inverse hallucination

ESMFold2-inverse hallucination designs a protein binder by backpropagating structural and language-model losses through a frozen ESMFold2 structure predictor. It optimizes soft residue identities over a 150-step design trajectory per binder. The benchmark targets the ESM cookbook’s binder-design tutorial from `esm` 3.4.0, run over a Biohub `transformers` fork (version 4.57.6) that supplies the ESMFold2 and ESM C model classes [35, 57].

Benchmark setup. Each design folds a target-binder complex through the pair trunk and an ESM C 6B pseudo-perplexity term over 150 optimization steps. Runs cover seven input sizes from 200 to 800 tokens (target plus an 80-residue binder), with one design per call and one fresh process per design in every configuration; batch size is 1 throughout: the cookbook’s trajectory-batching option was not used, and the optimized modes support only one design per call. Each input size is measured over three replicate runs of two designs, after an untimed warm-up pass, on one exclusive NVIDIA H100 80 GB GPU per run; reported speed-ups are the geometric mean over the seven input sizes. The headline clock is the steady-state seconds per design step read from the software’s own run summary, excluding first-call, compilation, and graph-capture time; a companion whole-pass wall-clock time, including model loading, compilation, graph capture, and the final scoring folds, is reported alongside it. Peak GPU memory is the device-level high-water mark read externally through NVML, above the idle baseline, with in-process allocator counters recorded beside it rather than substituted for it. For this model, default is the cookbook loop with two documented one-line settings applied at load time: an unchunked pair stack and the NVIDIA cuEquivariance triangle-kernel backend, which – unlike the upstream library’s own fused kernels – remain usable under gradients. This is the fastest correct configuration reachable through the library’s own settings, chosen so that speed-ups are not measured against an artificially slow baseline. Base, default and all modes carry the same small upstream bug fixes (transition-block data type; ESM C rotary embedding under gradients); base needs the data-type fix to run at these numerics. Base, the upstream library invoked as shipped apart from these fixes (pair stack chunked at 64, reference einsum path under gradients), is markedly slower than default and increasingly so with size (indicative only).

Sampling settings. Default and every mode used the same sampling settings: each design is a 150-step optimization with a cosine temperature schedule from 1 to 0.01 (one loop per step) for a binder of fixed length 80 (upstream draws the length at random), and each call produces one design. At every step, the target–binder complex is folded, as a single sequence with soft residue types, one trunk loop and no templates, through the two inversion models, ESMFold2-Experimental-Fast and ESMFold2-Experimental-Fast-Cutoff2025, and the structure losses are back-propagated through them into the binder sequence together with the pseudo-perplexity of the 6B-parameter ESM C model. The confidence head enters the objective only on the final low-temperature steps (temperature below 0.05). After the trajectory, four critic folds (the two inversion models plus ESMFold2-Experimental and ESMFold2-Experimental-Cutoff2025) score the finished design once; they are in the whole-pass clock, not the per-step clock.

Modes. All three modes – Exact, Fast, and Big – are implemented for this model. Exact preserves default’s arithmetic and was verified to give bit-identical outputs to default per optimizer step under a deterministic recipe (maximum absolute deviation 0.0). Fast and Big use reordered or reduced-precision kernels and are not bitwise. Their accuracy guard is a per-optimizer-step comparison of loss terms, gradients, and post-update outputs against default’s own dumped design states; it must fall inside default’s own run-to-run (seed-to-seed) variation band. Both passed this check on their primary test set, but Fast did not pass a separate check on an EGFR design case. Big uses Fast’s kernels but recomputes every pair-trunk block as default does and drops the memory-holding CUDA graphs and caches, trading speed for peak memory at or below default’s.

Optimizations. The optimized package composes several changes on top of the unmodified cookbook loop, without editing it:

- **Resident pair-block activations.** Spare GPU memory is used to cache as many pair-trunk block intermediate values as fit, instead of discarding and recomputing them during the backward (gradient) pass. We estimate that this is the largest single contributor to Exact’s speed-up.
- **Fused triangle-multiplication kernel.** The pair trunk’s core triangle-multiplication step is replaced, under gradients, with a single fused, lower-precision (bfloat16) forward and backward computation, contributing most of Fast’s additional gain over Exact.
- **Skipped confidence head and deferred structure sampling.** Steps that do not use the confidence head’s output or the one-step structure sample skip computing them.
- **CUDA-graph capture of the trunk for smaller complexes.** For complexes at or below 256 tokens, the trunk’s gradient-enabled forward and backward passes are recorded once as a CUDA graph (a captured sequence of GPU operations replayed without repeated launch overhead) and reused each step.
- **CUDA graphs and reuse for the language-model term.** The 80-layer ESM C forward pass and the pseudo-perplexity computation are likewise captured as graphs. After the first full pass, the target side of the complex cannot change within a design, so it is skipped and only the binder rows are recomputed.

Results. Fast gives the largest per-design-step speed-up at every input size. Exact and Fast are both fastest at the smallest input sizes and narrow toward 800 tokens; Big’s speed-up instead grows with input size and overtakes Exact’s at the larger sizes. Geometric-mean per-step speed-ups over the seven input sizes are 2.08× for Exact, 3.04× for Fast, and 1.86× for Big. Whole-pass wall-time speed-ups include one-time costs such as model loading and compilation. They are consistently smaller than the per-step figures (geometric means of 1.62×, 1.94×, and 1.50× in the same order), because those fixed costs weigh more when individual steps are cheap, i.e. at the smallest input sizes. Big’s peak memory stays at or below default’s at every size, whereas Exact and Fast use most of the card’s free memory by design.

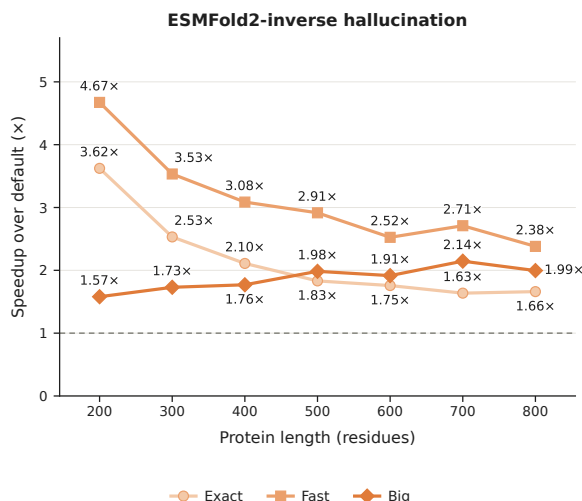


Figure S76. ESMFold2-inverse hallucination: per-design-step speed-up over default on H100. Speed-up over default (steady-state seconds per optimization step) versus input size in tokens, one line per mode (Exact, Fast, Big); the dashed line marks default (1×).

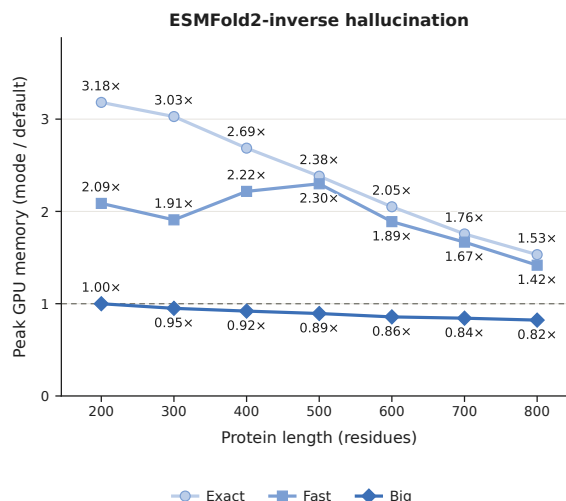


Figure S77. ESMFold2-inverse hallucination: peak GPU memory relative to default on H100. Peak device memory (NVML high-water mark above idle, from dedicated memory runs) relative to default versus input size in tokens, one line per mode (Exact, Fast, Big); the dashed line marks default (1×).

S2.3 Mosaic

Mosaic designs protein binders by gradient descent on a relaxed sequence representation – a 20-way probability vector per binder position – scored at each step by Boltz-2 through a JAX translation of its structure-prediction module, together with ProteinMPNN as an auxiliary sequence-recovery term. The benchmarked release is the upstream `mosaic` design package together with its pinned Boltz-2 and ProteinMPNN dependencies [48].

Benchmark setup. Each item is one protein-binder design task: a public target complexed with an 80-residue binder. It is run at seven input sizes (200–800 tokens, target residues plus binder length) with two public targets per size, one design per process on one GPU (upstream supports no batching, and none was added). Three replicates were run per input size, each bracketing every mode’s pass between two default passes required to agree within 5% (observed drift 0.55%), after an untimed default warm-up. All measurements used one NVIDIA H100 80 GB GPU. The headline clock is steady-state seconds per optimization step (one forward-and-backward pass of the design loss plus the sequence update), excluding the first two iterations (compilation and first call). A companion clock records the wall time of a whole pass (two designs), which additionally includes one-time costs such as weight loading, featurization, compilation, and the final refold. Peak GPU memory is the NVML high-water mark above an idle baseline, measured in a dedicated pass per input size with the allocator in growth mode; because the allocator extends in roughly 8.6 GB chunks, small ratios near 1× can reflect allocator granularity rather than genuinely equal usage. For this model, default coincides with base: the unmodified upstream package run through its own documented interface in its fastest supported configuration, with no added compilation cache or batching. As a result, the comparison isolates what the optimized package changes rather than comparing against an artificially slow baseline.

Sampling settings. Default and every mode used the same sampling settings. Each call designs one 80-residue binder for one target: the design loss is optimized with `simplex_APGM` for 75 steps at step size 0.1, then for 50 steps at step size 0.5 with `scale=1.5`, followed by one refold. The structure predictor is Boltz-2 (`boltz2_conf.ckpt`) with 1 recycling step and 25 deterministic diffusion steps, using a precomputed MSA and no templates, and the sequence-recovery loss draws 16 ProteinMPNN samples (`v_48_020`) per step.

Modes. All three modes are present. Exact changes only one-time start-up costs – a persistent compilation cache with pinned autotuning results, faster weight loading, and reuse of precomputed input features – so the timed design step itself executes the same compiled code as default, unaltered. It was verified bit-identical to default (checked across hosts under deterministic settings and a fixed seed). Fast and Big instead change per-step arithmetic and were checked against a declared per-step tolerance, and both stayed within it. Big additionally applies a memory reschedule that lowers its peak memory below Fast’s from 300 tokens up, at a per-step time cost of 15–20% relative to Fast.

Optimizations. The optimized package leaves the design recipe, loss terms, and step counts untouched and instead speeds up the per-step arithmetic and start-up path.

- **Fused triangle-attention kernels.** A single fused GPU routine computes triangle attention’s forward and backward passes together instead of building large intermediate attention-score arrays first, cutting memory traffic and time on every step.
- **Lower-precision pair track.** The pairwise trunk representation runs in `bfloat16` (a lower-precision 16-bit number format) with `float32` accumulation, trading a little numerical precision for faster arithmetic.
- **Fused triangle-multiplication kernels with a fixed layout.** Triangle-multiplication is combined into one kernel that keeps data in a single, consistent memory layout, avoiding repeated data-reformatting.
- **Faster structure-sampler math and template skip.** The diffusion structure sampler’s matrix multiplications run in `TF32` (a faster, lower-precision matrix-multiply mode), and the template module is skipped entirely when no template is supplied, since its contribution would be exactly zero.
- **Memory rescheduling (Big only).** Big recomputes parts of the pairwise trunk in groups rather than storing every intermediate result and processes triangle attention in row-by-row chunks. This reduces peak memory (about 35 GB against Fast’s 53 GB at 800 tokens), at some added recompute cost per step.

Results. Fast’s per-step speed-up over default grows with input size, reaching 4.03× at 700 tokens with a geometric mean of 3.06× over the 200–700 token range. Big tracks somewhat lower speed-ups throughout, consistent with the extra recompute of its memory reschedule. This per-step gain is larger than the whole-pass gain, because a large one-time cost per process (weight loading, featurization, compilation, and the final refold) dominates at small input sizes and differs between modes. Fast and Big, like default, compile their own programs in every process, so their whole-pass speed-ups are only 0.99× and 0.94× at 200 tokens (2.67× and 2.34× at 700 tokens). Exact, which shortens this cost, has no per-step gain (1.00×) but whole-pass speed-ups of 1.34× at 200 tokens

and 1.04 $\times$ at 700 tokens. Default runs out of memory before reaching the largest input sizes, as does Exact at 700 and 800 tokens in the memory runs, whereas Fast and Big both complete designs at 800 tokens, with Big holding a peak of about 35 GB there. More broadly, Big's peak memory grows more slowly with input size than default's because its rematerialization schedule keeps peak usage tied to one group of layers rather than the whole network.

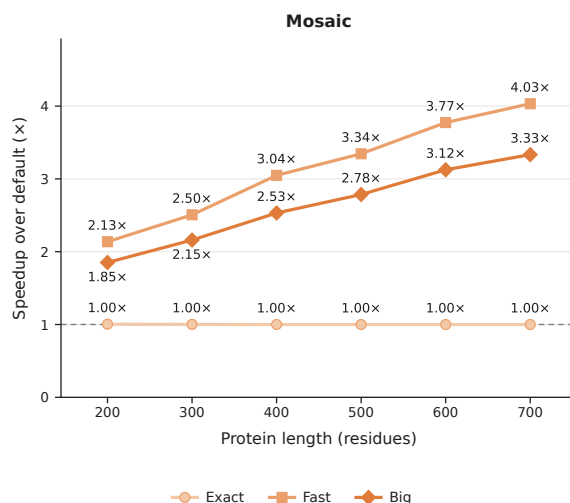


Figure S78. Mosaic: per-step speed-up over default on H100. Speed-up over default in steady-state seconds per optimization step, plotted against input size (target plus binder tokens), one line per mode (Exact, Fast, Big); the dashed line marks default (1 $\times$). No speed-up point is shown at 800 tokens because default runs out of memory there.

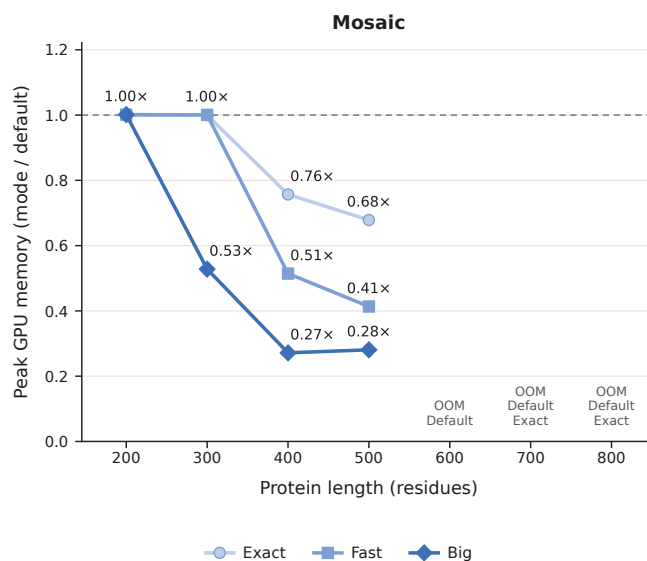


Figure S79. Mosaic: peak GPU memory relative to default on H100. Peak device memory above idle (NVML high-water mark, dedicated memory runs) for each mode relative to default, plotted against input size; the dashed line marks default (1 $\times$). Points are absent where default itself runs out of memory (600 tokens and above under this allocator setting).

S3 Structure generation

These generative models produce new protein backbones or all-atom designs, typically by diffusion or flow matching. Speed-ups are for design throughput, as stated per model. Section S7 compares the quality of designs made under each mode.

S3.1 Genie 3

Genie 3 (aqlaboratory/genie3) is a denoising-diffusion model that generates protein backbones: each design is one CA-only PDB produced by 100 DDIM steps, conditioned on either a target chain length or a target structure with hotspot residues plus a binder length. The benchmark pins an untagged upstream commit at package version 0.0.1, with the published v1 checkpoint [18].

Benchmark setup. Two input-size ladders were measured separately, never pooled: unconditional backbones of 100-500 residues, and PD-L1 binders (115-residue target, hotspot Tyr56) with total complex size 200-700 residues. Default and both modes ran at batch 8 on one NVIDIA H100 80GB GPU per job, with 72 designs per pass (nine calls) on the unconditional ladder and 48 (six calls) on the binder ladder; on both ladders the first call of every pass is a warm-up unit, excluded from the rate. Speed is the median of 3 independent jobs per input size, with an untimed warm-up pass excluded; peak memory comes from one job per input size. The headline clock is designs per GPU-second over the steady sampling loop, start-up excluded; a companion whole-pass wall clock (launch to exit, including compilation) is also reported. Peak GPU memory is the NVML device high-water mark over a whole pass, above the card’s idle baseline. Default means the upstream package run with its two documented speed settings (`torch.compile` enabled and `batch_size` set to 8) and a seed. The batch of 8 was declared once rather than tuned per size. At short lengths, default’s denoiser calls are limited mainly by kernel-launch overhead, which a larger batch could partly remove. The base configuration (upstream exactly as shipped, at batch 1, one job per input size) is markedly slower at short input sizes and is slightly (5%) faster than default at the longest binder length.

Sampling settings. Default and every mode used the same sampling settings: upstream’s DDIM sampler with 100 steps, η 1.0, and noise scale 1.0, and 8 designs per call (batch size 8; upstream default 1), using the single Genie 3 v1 release checkpoint.

Modes. Exact and Fast both exist; there is no separate Big mode, because the memory-saving changes are in both modes, although Fast’s peak memory exceeds default’s at the larger binder sizes. Exact keeps upstream’s float32 arithmetic and uncompiled kernels, replayed as CUDA graphs. Its design files matched, by sha256 hash, those of upstream run without `torch.compile` under deterministic settings at batch size 2 and the same seed, on the shortest and longest input of each ladder and across host machines. Fast additionally uses reduced-precision and approximate kernels. Its design quality is compared with default’s in Section S7.

Optimizations. The following changes speed up the sampling loop, in rough order of estimated contribution (no per-change ablation was run):

- **CUDA graph replay.** The denoiser’s pair-transform, sequence, and structure networks are captured once per batch shape and replayed at each of the 100 steps, removing per-step launch and Python overhead. A sync-free step (host synchronization replaced by device-side values) makes capture possible.
- **Featuriser hoist.** Step-invariant terms of the pair featuriser (relative-position encoding, masks, conditioning terms) are computed once per batch instead of being rebuilt at every step.
- **Reduced-precision and fused kernels (Fast only).** A fused triangle-multiplication kernel (unused below 101 residues, so not at 100 residues) and NVIDIA’s lower-precision TensorFloat-32 (TF32) matrix multiplication replace some of upstream’s IEEE float32 operations. An inductor-compiled captured core fuses elementwise operations between matrix multiplies, and each pair-transition block processes a whole design at once rather than in 4-row chunks.

Outside the sampling loop, one persistent process serves every batch of a request and reuses captured graphs for repeated input shapes, which shortens only start-up and graph capture, both excluded from the sampling clock.

Results. On the unconditional ladder, Fast’s speed-up on the steady sampling (forward) clock falls from about 14.9 $\times$ at the shortest input size to about 5.2 $\times$ at the longest, while its whole-pass speed-up (not plotted) rises from 3.1 $\times$ to 4.6 $\times$, because a largely fixed compile-and-capture cost is amortized over a longer loop. The PD-L1 binder ladder shows a similar pattern: Fast’s sampling-clock speed-up is 9.1 $\times$ at the shortest size and 5.2–6.0 $\times$ from 500 residues on, and its whole-pass speed-up rises from 4.2 $\times$ to 5.8 $\times$. Exact’s forward-clock speed-up is far more modest throughout, since it only removes per-step overhead rather than changing arithmetic. Peak GPU memory for Exact stays below default at every input size, while Fast’s approaches or slightly exceeds default’s, reaching about 1.14 $\times$ at the largest binder size.

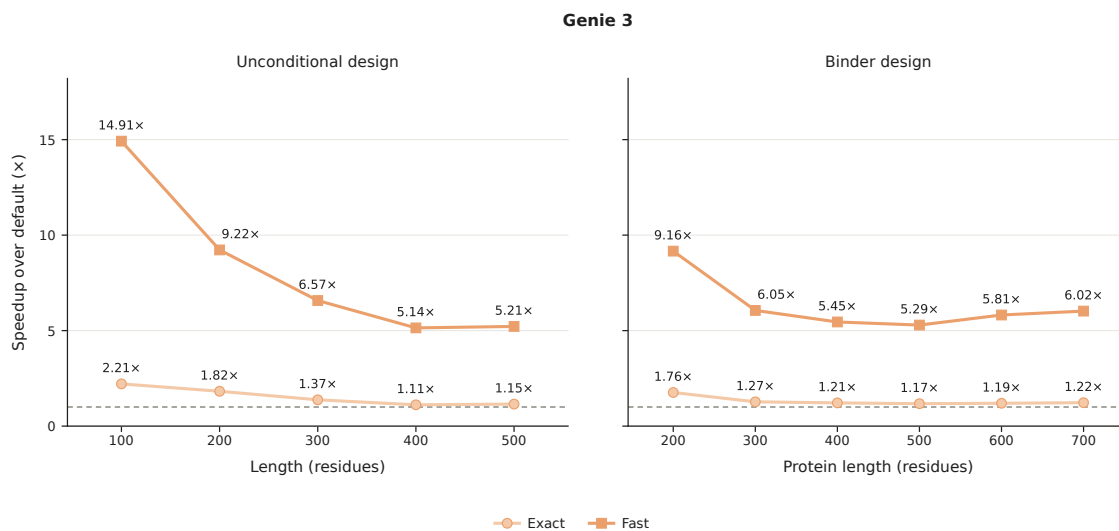


Figure S80. Genie 3: speed-up over default versus input size on an H100 GPU. Each panel shows one input set (unconditional design or binder design); lines show Exact and Fast on the sampling (forward) clock, with the dashed line marking default (1×).

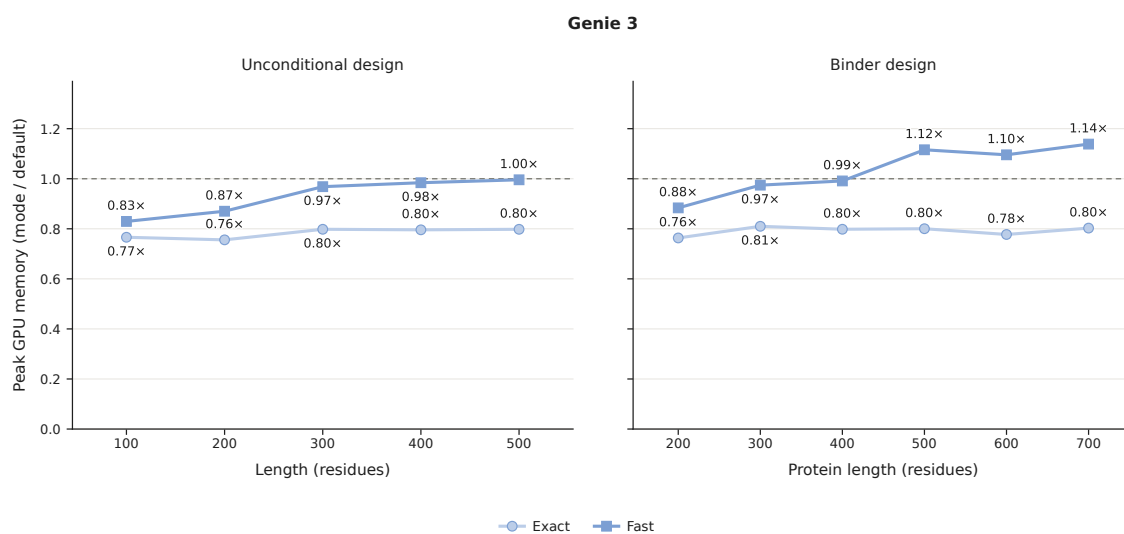


Figure S81. Genie 3: peak GPU memory relative to default. Each panel shows one input set (unconditional design or binder design); lines show Exact and Fast peak GPU memory (NVML device high-water mark above idle) as a ratio to default at the same input size, with the dashed line marking default (1×).

S3.2 PXDesign

PXDesign is a *de novo* protein-binder design suite whose diffusion generator, PXDesign-d, is built on Protenix. We benchmarked the `pxdesign infer` command from bytedance/PXDesign, package 0.1.0, with the released `pxdesign_v0.1.0.pt` checkpoint [19].

Benchmark setup. The task is binder design against one target, PD-L1 (PDB 5JDS chain A, residues 18–132, hotspot Tyr56), with binder length varied so that target and binder together give complex sizes of 200, 300, 400, 500, 600, and 700 tokens exactly. Each pass runs the same design task (same target, hotspot and binder length) under six different fixed seeds, the same six for every configuration; each run samples 40 designs in one batch (240 designs per pass). The first task of every pass is a warm-up unit, recorded but excluded from the rate, so the rate is over 200 timed designs. For each input size, three replicate jobs interleave a warm-up, default, and each mode, with default measured immediately before and after every mode, all on one exclusive GPU per process on an H100 80 GB GPU. The headline clock is designs per GPU-second over the per-task sampling calls (trunk embedding, the 400-step denoiser loop, and output writing), excluding start-up. A whole-pass wall clock (process launch to exit) is reported alongside as a companion measure. Peak memory is the device’s NVML peak above idle, measured in dedicated memory runs. For this model, default engages Protenix’s fused LayerNorm via an environment variable set before Python starts (base’s own flag takes effect too late to matter) and raises the per-task batch from base’s 5 to 40 designs, via upstream’s own switches. 40 designs sits on the throughput plateau for every mode, giving a fair, saturated comparison. The base configuration – the package run exactly as shipped, at batch 5 with plain LayerNorm – is markedly slower than default, reaching about 0.46–0.50× of default’s designs per GPU-second across the input-size range.

Sampling settings. Default and every mode used the same sampling settings: a 400-step diffusion sampler denoising 40 designs per task in one batch (`--N_sample 40 --sample_diffusion_chunk_size 40`; upstream default 5 designs), with the single released checkpoint `pxdesign_v0.1.0.pt`. Inputs are single-sequence features with no MSA and no templates (upstream’s MSA option is on by default).

Modes. Exact, Fast, and Big are all implemented for PXDesign. Exact was verified bit-identical to default under a deterministic recipe (a fixed cuBLAS workspace configuration and deterministic seeding), passing on a panel and a 200/400/700-token ladder check, both run at batch size 2. Fast, the mode the optimized package runs when none is specified, is checked against an accuracy guard requiring its outputs to fall within default’s own seed-to-seed spread rather than match default exactly. The design quality of Fast and Big is compared with default’s in Section S7. Big applies Fast’s numerics and is meant to lower peak memory further for inputs too large for Fast on one GPU, which were not measured here.

Optimizations. Estimated contributions to the speed-up, from largest to smallest, then memory changes:

- **Pair-conditioning hoist (all modes).** A large per-task conditioning tensor used at every diffusion step is computed once per task and reused instead of being recomputed at all 400 steps; this gives most of Exact’s gain.
- **TF32 matrix multiplication (Fast, Big).** Fast and Big run some of the sampler’s float32 matrix multiplies in NVIDIA’s faster, lower-precision TensorFloat-32 (TF32) mode, giving most of the additional speed-up beyond Exact and the main source of the resulting numerical drift.
- **Token-transformer pair-bias caching (all modes).** Attention bias terms used throughout the transformer, which do not change between denoising steps, are computed once and cached rather than recomputed every step.
- **Sample-row de-duplication (Fast, Big).** Some per-step computations are done once and broadcast across the batch instead of repeating for each design, because every design in a batch shares the same noise level at a given step.
- **Leaner per-task tensors (Fast, Big).** Once-per-task work runs on one row, broadcast over the 40 designs. The atom padding mask comes from index arithmetic rather than a dense atom-pair tensor. A training-only atom-pair feature stays off the GPU. These changes lower peak memory.
- **Row-slab evaluation (Big).** Big evaluates the once-per-task pair-tensor computations in slabs of at most 256 token rows, written into preallocated outputs, so that no full-size float32 intermediate is held in memory. This is meant to fit inputs too large for Fast on one GPU, and on the benchmarked inputs Big’s speed and peak memory closely match Fast’s.

Results. Across the 200–700 token range, the sampling-clock speed-up over default rises overall (not monotonically) with input size, reaching 2.11× for Exact and 3.79× for Fast at 700 tokens, with Big tracking Fast throughout. Peak GPU memory for Fast and Big falls to about 0.21× of default’s by 700 tokens, while Exact’s memory closely tracks default’s except at 200 tokens (1.27×) and at 400 tokens (1.55×, a flagged outlier). The whole-pass wall-time speed-up is consistently smaller than the sampling-clock speed-up,

reflecting a fixed per-process start-up cost that weighs more at small input sizes, where designs are quicker. The gap narrows with size for Fast and Big but stays roughly constant for Exact.

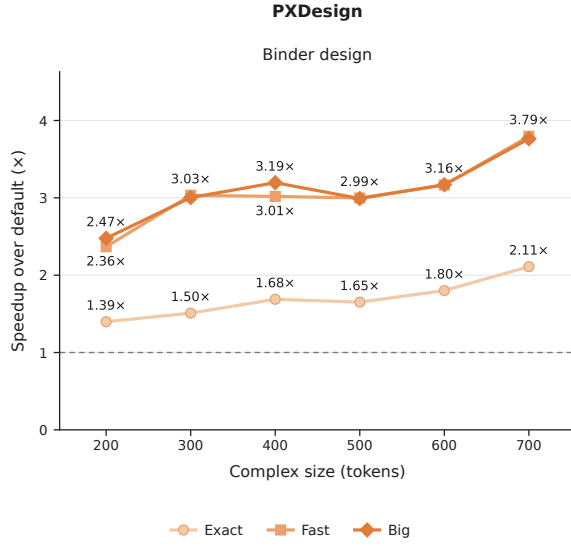


Figure S82. PXDesign: speed-up over default versus input size on H100. Each line is one mode (Exact, Fast, Big) on the binder-design task; the vertical axis is the speed-up on the designs-per-GPU-second sampling clock, and the dashed line marks default (1×). Fast and Big nearly coincide, so the Fast line is largely hidden under Big.

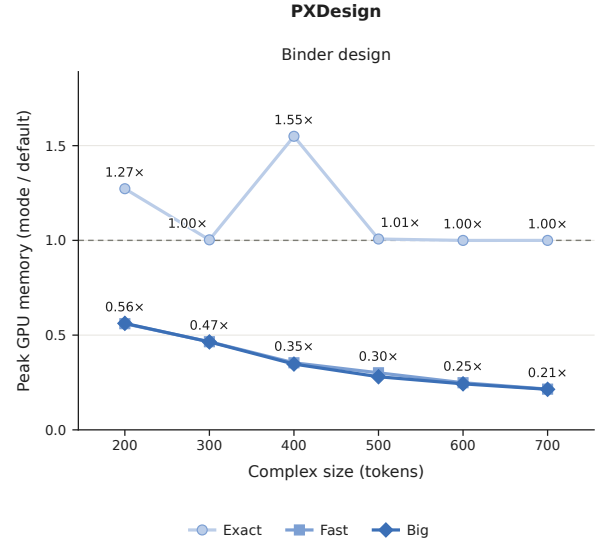


Figure S83. PXDesign: peak GPU memory relative to default versus input size on H100. Each line is one mode (Exact, Fast, Big) on the binder-design task, measured in dedicated memory runs; the vertical axis is peak NVML device memory above an idle baseline relative to default at the same input size, and the dashed line marks default (1×). Fast and Big nearly coincide, so the Fast line is largely hidden under Big.

S3.3 Proteina-Complexa

Proteina-Complexa is a 160M-parameter flow-matching transformer that generates an all-atom protein binder against a fixed target. Each design uses 400 integration steps over the binder’s $C\alpha$ coordinates and per-residue latents, with one network call per step. The output is then decoded into a PDB structure. The benchmark uses upstream Proteina-Complexa at package version 1.1.0 (an untagged commit) with the protein-target checkpoint `NV-Proteina-Complexa-Protein-Target-160M-v1` [20].

Benchmark setup. The task is binder design against one fixed target, PD-L1 (PDB 5JDS chain A, residues 18–132). Input sizes span $N = 200, 300, 400, 500, 600, 700$ tokens, where N is the target’s 115 residues plus a binder of length $N-115$ (85–585 residues). Speed runs process 96 designs per pass (six units of 16, the first excluded as warm-up), and memory runs process 32 designs per pass. Speed values are the median over three replicate jobs per input size. Memory values come from one job per input size, all on an H100 80 GB GPU. The headline clock times the model’s per-batch sampling call – the 400-step loop plus the decode of one batch, device-synchronized – excluding start-up, checkpoint loading, target featurization, and PDB writing. A companion whole-pass wall clock covers process start to exit. Peak memory is the device-level (NVML) peak over the whole pass, GB above idle, measured from outside the process. Allocator-level figures are diagnostic only. For this model, default is upstream’s generation command with an explicit seed and its documented settings for single-pass generation without a reward model. Base, the shipped pipeline with best-of- n search, an AlphaFold2-based reward and filtering, was not measured. No optimization is added on the default side, and every mode uses the same settings. Only the generation stage is timed, and only binder design is run, as the checkpoint has no unconditional route.

Sampling settings. Default and every mode used the same sampling settings, which are upstream’s: 400 integration steps with self-conditioning on, guidance off, and the stochastic sampling mode (`sc`) with noise scale 0.1, and 16 designs per call (the shipped batch size). The model is the 160M-parameter flow-matching denoiser `NV-Proteina-Complexa-Protein-Target-160M-v1` with its frozen autoencoder.

Modes. Exact, Fast, and Big are all implemented. Exact removes host-device synchronization stalls during sampling and avoids recomputing per-step quantities that do not change across steps, leaving the arithmetic unchanged. It is verified bit-identical to default (at equal seed and batch), checked over the full 400-step trajectory at $N200$, $N500$, and $N700$, the last two also across host machines. Fast additionally fuses the pair-representation normalization and every layer’s attention-bias projection across the transformer’s 14 layers into one computation per network call, and replaces the multi-step pair-biased attention computation with a single fused call. Both changes introduce floating-point re-association drift. Section S7 compares its design quality with default’s. Big keeps Exact’s numerics (also verified bit-identical, across host machines) and changes only the memory layout of the pair-bias computation. In BinderBench (Section S7.1), Big matched default byte for byte in 9 of 10 generation jobs. In the tenth (PD-L1, design seeds 8–15), 61 of 128 designs scored differently.

Optimizations. The optimized package swaps in PyTorch-level replacements for specific computations inside the unchanged generation process, with no custom GPU kernels and nothing compiled at run time.

- **Fused pair-bias projection (Fast).** The pair representation does not change across the transformer’s 14 layers, so its normalization statistics and every layer’s attention bias are computed once per network call instead of once per layer.
- **Fused attention (Fast).** Pair-biased attention is computed with a single built-in attention call that takes the bias and mask as one additive tensor, replacing a separate multiply-softmax-multiply sequence.
- **Removed per-step synchronization (Exact, Fast, Big).** Step timing comes from a host-side schedule instead of values read back from the GPU at every step, removing synchronizations that would otherwise stall the sampling loop.
- **Cached target features (Exact, Fast, Big).** The part of the pair features that depends only on the fixed target is computed once per batch and reused at all 400 steps instead of every step.
- **Direct pair assembly (Exact, Fast, Big).** With one binder length per batch, as here, the extended pair tensor is built by block copies without host synchronizations (skipped when binder lengths vary). One-hot features are built directly in float32.
- **Row-blocked pair-bias layout (Big).** The pair-bias projection is computed in row blocks written directly into the attention-bias tensor, so the full normalized pair tensor is never held in memory at once, without changing the numerics. On this benchmark it lowers the peak below Exact’s only at $N300$.

Results. The speed-up over default on the per-batch sampling clock is roughly flat across the input-size ladder, with geometric means over the six sizes of 1.42 $\times$ for Exact, 2.72 $\times$ for Fast, and 1.31 $\times$ for Big. This flat profile is consistent with most of the removed work scaling with the pair tensor that dominates default’s time at every size. The whole-pass wall-time speed-up is smaller than this at the shortest passes and converges toward it as passes lengthen, because each process carries a fixed cost that no mode reduces.

Peak GPU memory is unchanged among all modes at the smallest input size but drops under every mode from the next size upward, with Exact and Big showing the largest reductions relative to default.

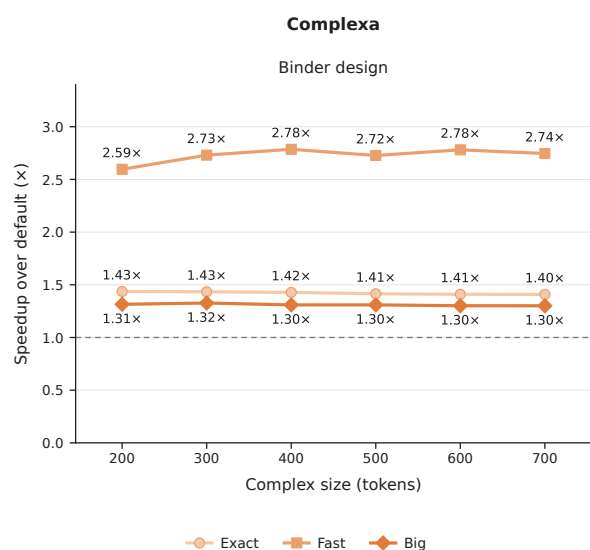


Figure S84. Proteina-Complexa: design-throughput speed-up over default. Speed-up over default (designs per GPU-second, timed around the per-batch sampling call) versus input size (tokens) for the PD-L1 binder-design task on H100, one line per mode (Exact, Fast, Big); the dashed line marks default (1×).

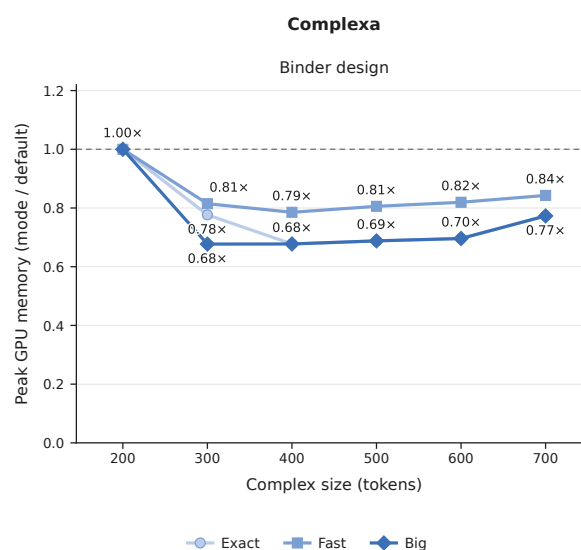


Figure S85. Proteina-Complexa: peak GPU memory relative to default. Peak device memory (NVML, above idle) for each mode (Exact, Fast, Big) relative to default versus input size (tokens), from dedicated memory runs at 32 designs per pass on H100; the dashed line marks default (1×). At 200 tokens all modes equal default; from 400 tokens up Exact's peak equals Big's, hiding the Exact line.

S3.4 RFdiffusion3

RFdiffusion3 is the all-atom protein-design diffusion model (rfd3) of the RosettaCommons *foundry* project, run through its `rfd3` design command line [16].

Benchmark setup. The task is unconditional, fixed-length backbone generation (design step only; no sequence design, re-folding, or filtering) at five input sizes from 100 to 500 residues, run one GPU per job on an H100 80 GB card. The batch size (`diffusion_batch_size=8`, upstream’s shipped value) is fixed across every mode and input size. The first roll-out (compile / graph capture) is excluded from the timed statistic, leaving 11 timed roll-outs per pass. The speed comparison uses 3 replicate jobs per input size. The headline clock is designs per GPU-second, timed on the model-forward call producing each batch of eight designs and averaged over the 11 steady roll-outs. A companion whole-pass wall clock (process start to exit, including model load, warm-up, featurization, file I/O, and teardown) is also reported. Peak memory is the device’s peak `nvidia-smi` memory above an idle baseline, measured in dedicated one-job-per-input-size runs, and includes the CUDA context, allocator pool, and captured graphs. Default adds upstream’s own `compile_model=true` switch plus the NVIDIA apex fused RMSNorm kernel, which upstream binds automatically whenever apex is importable. This is the fastest configuration reachable through upstream’s own flags, so the optimized package is compared against the strongest correct baseline rather than a naive install. Base (run exactly as shipped: eager execution, no apex) is markedly slower, and the gap widens with length. It reaches only 0.75× to 0.49× of default’s throughput from the shortest to the longest input size, with whole-pass wall time 1.07× to 1.86× longer.

Sampling settings. Default and every mode used the same sampling settings, which are upstream’s shipped sampler settings: a 200-timestep noise schedule (199 denoiser calls per roll-out) with step scale 1.5, γ_0 0.6, and noise scale 1.003, 2 recycles in each denoiser call, and 8 designs per roll-out (`diffusion_batch_size=8`), with the `rfd3_latest` checkpoint.

Modes. Only Exact and Fast exist for RFdiffusion3; there is no Big mode or multi-GPU mode. Design-CP, developed separately, runs RFdiffusion3 on multiple GPUs [60]. It uses context parallelism to split the model’s pair representations across GPUs, which makes large symmetric protein assemblies tractable to design. Exact is constrained to produce outputs identical to default under upstream’s deterministic `--det 1` switch, and was verified bit-identical under that switch (144 of 144 output files matched, checked at batch size 2). Because default includes upstream’s compile switch, and compiled execution cannot serve as a bitwise reference, the check compares Exact with the unmodified model run without compilation, which is otherwise default’s configuration, including its fused RMSNorm. Fast allows reduced-precision, approximate kernels; no test with an acceptance threshold is reported for it, and Section S7 compares its designs with default’s.

Optimizations. The optimized package adds the following changes to upstream’s unchanged sampling loop. They are not ranked by contribution.

- **CUDA-graph step replay.** One denoiser step is captured per (batch, length) shape using CUDA graphs (recording the GPU operations once and replaying them with much lower launch overhead) and reused for every later step and roll-out of that shape.
- **Roll-out hoist.** Tensors that stay constant within a roll-out – pair-bias projections, a conditioning down-cast, attention masks – are computed once per roll-out instead of being recomputed at every denoising step.
- **Fused feed-forward kernel.** A single kernel serves the SwiGLU feed-forward blocks, keeping upstream’s numerical rounding while avoiding materializing the widened intermediate activation.
- **Compilation and attention fusion (Fast only).** Compiling upstream’s denoiser sub-modules and replacing dense atom-level attention with neighbor-gathered attention plus a fused attention kernel for the token transformer cut per-call cost further. This comes at the cost of exact bit-reproducibility.
- **Row-blocked initializer embedding (memory saving).** The token initializer’s pairwise distance embedding is computed in bounded-size blocks rather than all at once, capping a transient memory peak without changing outputs.

Results. On the designs-per-GPU-second clock, the speed-up over default is largest at the shortest input size, where the sampling step is launch-bound rather than compute-bound, and shrinks as length grows. Fast is faster than Exact at every input size. Default’s throughput at 100 and 200 residues differed between two runs of the same protocol on different hosts, so gains there are ranges: 3.1–3.9× for Fast and 2.5–3.1× for Exact at 100 residues, and 2.0–2.2× and 1.5–1.7× at 200 residues. At 500 residues, Fast reaches 1.62× and Exact 1.08×. Relative to default, peak memory falls with length for Fast, to 0.52× at 500 residues. Because the row-blocked initializer removes the transient that sets default’s peak, Exact is 1.09× at 100 residues and 0.73–0.83× from 200 residues up. Whole-pass gains are smaller than the per-call gains for Fast at every input size and for Exact at 100 and 200 residues. Model loading and file I/O are unchanged, and Fast’s first roll-out (compilation and graph capture) is longer than default’s compilation. From 300 residues up, Exact’s whole-pass gain slightly exceeds its per-call gain (1.12× against 1.08× at 500 residues), because its

graph capture is shorter than default's compilation. On the whole-pass clock, Exact leads Fast at 100 and 200 residues (2.61× and 1.70× against 1.83× and 1.53×).

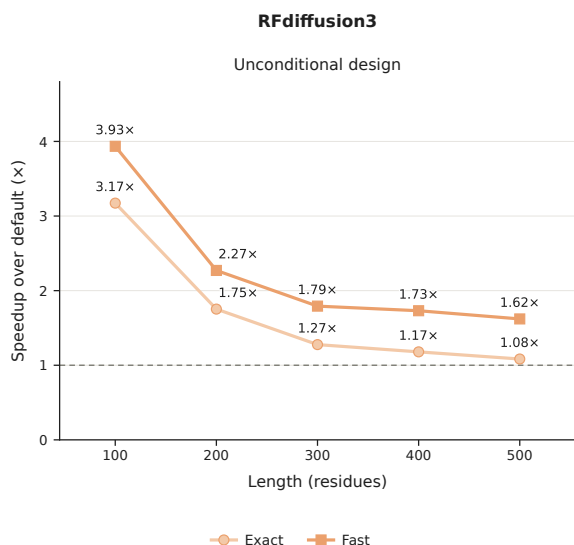


Figure S86. RFdiffusion3: speed-up over default vs input size. Designs per GPU-second, mode over default of the same run, for unconditional backbone design at input sizes from 100 to 500 residues; one line per mode (Exact, Fast), dashed line = default (1×). At 100 and 200 residues, default's throughput varied between hosts, so these points are uncertain (see Results).

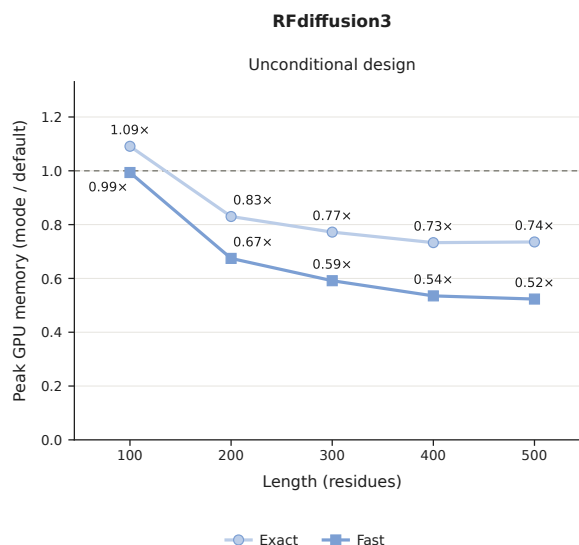


Figure S87. RFdiffusion3: peak GPU memory relative to default. Peak device memory (NVML peak above idle) for each mode as a ratio to default, measured in dedicated memory runs at batch 8, vs input size; one line per mode (Exact, Fast), dashed line = default (1×).

S3.5 RFdiffusion

RFdiffusion designs protein backbones by denoising residue frames through an SE(3)-equivariant network derived from RoseTTAFold, producing a full backbone trajectory over 50 reverse-diffusion steps. We benchmarked the upstream main branch of July 2026 (package version 1.1.0, 96 commits after the v1.1.0 release) [3].

Benchmark setup. Two design tasks were timed on an H100 80 GB GPU: unconditional backbone generation at lengths 100, 200, 300, 400, and 500 residues, and binder design against a fixed 115-residue PD-L1 target (PDB 5JDS) at total complex sizes 200–700 residues. RFdiffusion has no in-network batch dimension, so four concurrent seeded processes were packed on each GPU under CUDA MPS (a service that lets several processes share one GPU) to saturate throughput. This packing gave 24, 24, 16, 16, and 16 designs per pass across the unconditional lengths (24, 24, 16, 16, 16, and 16 for the binder task). Each process’s first design was excluded from timing as a warm-up. Each input size is measured over three replicates. Each replicate runs default passes before and after the Exact and Fast passes. The headline clock is designs per GPU-second over each design’s 50-step sampling span. A whole-pass wall clock (process launch to the last file closed, across all four processes) is reported alongside it. Peak GPU memory is the NVML device-level peak above idle for the whole card with all four processes resident. It is measured for each mode and for default in the same dedicated memory run at each input size and reported as the mode-to-default ratio. Default adds this four-way seeded process packing under CUDA MPS on top of base (a single unseeded, unpacked upstream process). This packing is the fastest correct way to keep the GPU busy given the model’s one-design-at-a-time execution, so that speed-ups reflect per-design gains rather than idle GPU capacity. A single base process reaches only 0.25× of default’s throughput at length 100, 0.62× at length 300, and 0.76× at length 500.

Sampling settings. Default and every mode used the same sampling settings: each design is one 50-step denoising trajectory producing one backbone (RFdiffusion has no batch dimension). The network is RFdiffusion 1.1.0 with upstream’s own checkpoint choice: `Base_ckpt.pt` for unconditional generation and `Complex_base_ckpt.pt` when hotspot residues are given for binder design.

Modes. Exact and Fast are the two available modes; there is no Big mode for this model. Exact was verified bit-identical to default on every written file (final backbone PDB, trajectory frames, and the `.trb` record apart from its timing field) under the same seeded, deterministic settings. The check covered full 50-step trajectories at length 100 and complex size 200, and the first 5 of 50 steps at lengths 300 and 500 and complex sizes 400 and 700. Exact’s outputs matched default’s on hosts of the same CPU type. Fast is checked with a per-step drift band against default on at least eight identical seeds, plus a refold quality check (ProteinMPNN and ESMFold pLDDT, CA-RMSD, scTM, designable fraction, and ipTM for binders). No run-to-run determinism claim is made for Fast. Section S7 detects no target- or length-averaged quality change under Fast. Fast’s median ipSAE_{min} was lower (0.120 against 0.206) on PD-L1, also this section’s binder target. This difference and a decrease of 0.005 on EGFR domain III (both medians below 0.02) are two of the four single-target differences in that section with an interval excluding zero, about as many as expected by chance.

Optimizations. The optimized package leaves upstream’s sampler, seeding, and noise schedule untouched and accelerates the surrounding computation and I/O.

- **Dense SE(3)-Transformer layer (Fast).** Replaces the graph-library evaluation of the structure module with a statically shaped, fused implementation across all 40 structure-module calls per step. This dense layer is the single largest contributor to the Fast speed-up.
- **Whole-forward CUDA graph capture/replay.** Records the embeddings, templates, and all 36 iteration blocks of a denoising step as one reusable CUDA graph. Replaying the graph removes thousands of small kernel launches per step, the dominant cost at short designs.
- **Resident driver.** Keeps one process alive per request, so the model and checkpoint are loaded once per request rather than once per upstream command invocation; the other changes run inside this process.
- **Per-design constants and host work.** Builds tensors that stay constant over a trajectory (masks, graph topology, lookup tables) once instead of at every step, cutting host-device round trips. Skips a CPU template featurization whose result the network never reads. Writes PDB files from arrays (byte-identical text), saving seconds of CPU time per design.
- **TF32 math and fused LayerNorm (Fast).** Uses TensorFloat-32 (TF32) math for float32 matrix multiplications and convolutions, and a Triton kernel for the network’s many small LayerNorm operations.

Results. Both modes give their largest speed-ups at the shortest designs on the sampling-span clock (the headline clock), and the gains shrink as designs lengthen. Fast reaches 2.56× default’s throughput at length 100 and falls to 1.63× at length 500. Exact’s bit-identical speed-up is much smaller and concentrated almost entirely at the shortest length. This pattern suggests that the network’s

own arithmetic, which Exact leaves unchanged, dominates the steps of longer designs. On the binder-design set, both modes gain most at the smallest complex size (Fast 1.94 $\times$, Exact 1.19 $\times$ at 200 residues) and stay roughly flat from 300 residues up (1.64–1.70 $\times$ and 1.02–1.06 $\times$). Fast’s peak GPU memory on the unconditional set drops to about 0.37 $\times$ of default at length 400 because the dense SE(3) layer avoids the graph library’s edge tensors. Exact’s memory stays slightly above default’s throughout. Peak memory in both default and Exact configurations reaches the card’s memory ceiling at the two longest binder sizes, so the ratios at these sizes are saturated. Exact’s saturated ratios (1.00 $\times$ and 0.99 $\times$) are uninformative, and Fast’s (0.43 $\times$ and 0.55 $\times$) understate its saving.

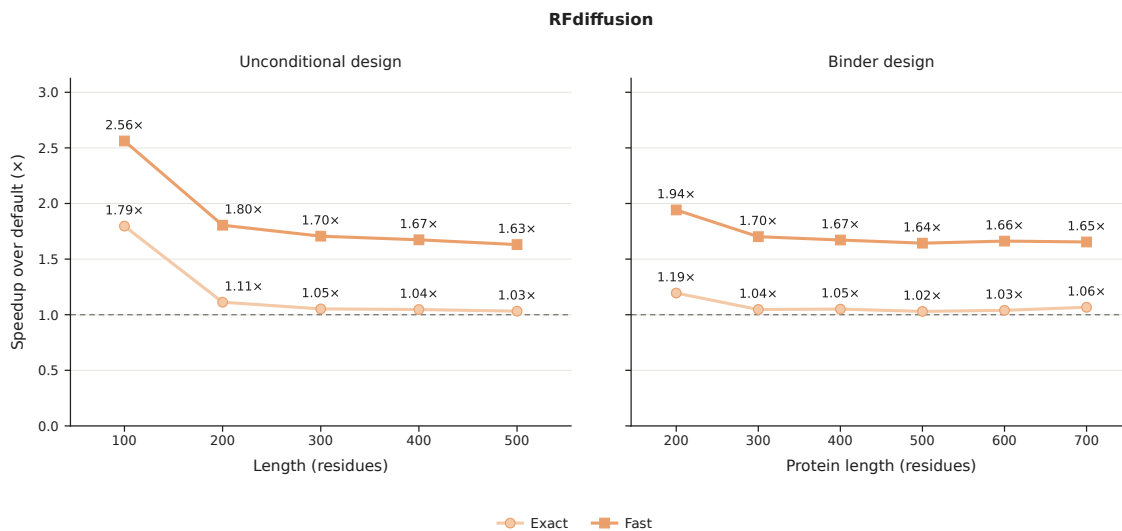


Figure S88. RFdiffusion: speed-up over default on H100. Speed-up on the sampling-span clock (the headline clock) versus input size, shown separately for the unconditional-design and binder-design sets, with one line per mode (Exact, Fast); the dashed line marks default (1 $\times$).

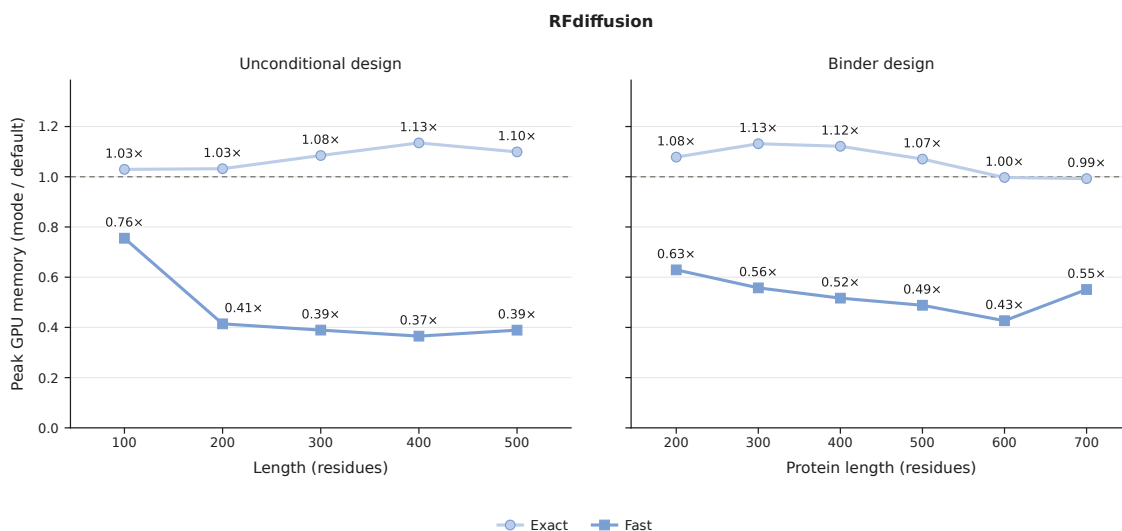


Figure S89. RFdiffusion: peak GPU memory relative to default. NVML device-level peak memory above idle for each mode, expressed as a ratio to default, versus input size, shown separately for the unconditional-design and binder-design sets; the dashed line marks default (1 $\times$).

S3.6 BoltzGen

BoltzGen is an all-atom generative diffusion model for protein and peptide binder design, in which a Boltz-style trunk conditions a 500-step diffusion sampler that draws a batch of designs per trunk pass [17]. We benchmarked BoltzGen 0.3.2 (the release wheel of tag v0.3.2, unmodified), using the published `boltzgen1_diverse` / `boltzgen1_adherence` design checkpoints. We timed only the design step of the pipeline.

Benchmark setup. The task is unconditional single-chain design at fixed length, at five input sizes from 100 to 500 tokens in steps of 100; the diffusion batch size is fixed at 64 for every configuration, giving 320 designs per pass (five calls of 64). Each reported value is the median of three jobs per input size, run on an exclusive H100 80 GB GPU. The headline clock is design-phase throughput (designs per GPU-second) over the steady sampling calls, excluding the first call and the checkpoint-switch call. A companion whole-pass wall-clock time is also reported. Peak GPU memory is the NVML device high-water mark above idle, measured in dedicated memory runs. For this model, default adds two things beyond base, the upstream package run exactly as shipped: the diffusion batch size raised to 64 and upstream’s own compile switches enabled. These two changes are the documented speed options a careful user would set, so the comparison targets the fastest correct configuration of the unmodified package. Base draws only 10 designs per sampling call (upstream’s automatic choice) and runs eager. It is markedly slower than default at small inputs, 0.34× default’s design-phase rate at 100 tokens, narrowing to 0.85× at 500 tokens.

Sampling settings. Default and every mode used the same sampling settings: 3 recycling steps and 500 diffusion steps per design, with the shipped step-scale and noise-scale schedules, and 64 designs per call (`diffusion_batch_size`; upstream chooses 10 automatically). Each pass switches from the released `boltzgen1_diverse` checkpoint to `boltzgen1_adherence` partway through its calls.

Modes. Exact, Fast, and Big all exist for BoltzGen. Exact applies the optimizations described below. Under matched seeds, it was verified bit-identical (sha256-identical CIF and npz files) to the unmodified package run without its compile switches, at 100 and 200 tokens and batch size 2 (no change is switched on or off by batch size). Fast adds reduced-precision fused kernels and is intended to stay within upstream’s own seed-to-seed output variation. Big is the memory-optimized configuration, meant for inputs beyond default’s limit (no measurement of the largest input it completes is reported). It runs slower than default below 500 tokens. Section S7 detects no target- or length-averaged quality change under Fast or Big; at 500 residues, Fast’s mean scTM was 0.049 lower than default’s (95% interval 0.014 to 0.090), one of two single-length differences with an interval excluding zero, about as many as expected by chance.

Optimizations. The optimized package leaves the upstream design step unchanged and adds the following at runtime, ordered by estimated contribution to the speed-up:

- **CUDA-graph sampler (Exact, Fast).** The per-step denoiser call in the diffusion loop is captured once per input shape as a CUDA graph – a recording of GPU operations that can be replayed without Python overhead – removing launch and synchronization overhead from the loop that dominates the design-phase clock.
- **Mask hoist (Exact, Fast).** Attention masks that do not change across diffusion steps are built once per structure rather than rebuilt in every layer at every step; the saving grows with the square of the token count, so the benefit increases with input size.
- **Fused reduced-precision kernels (Fast only).** The sampler’s attention and conditioning computations are fused into single lower-precision calls, and conditioning terms upstream recomputes for every design in a batch are computed once per step instead, trading a small, guarded amount of precision for extra throughput on top of Exact’s optimizations.
- **Faster start-up (all modes).** Upstream restarts a fresh interpreter and rebuilds the model at each pipeline step, including a redundant weight initialization that the checkpoint load then overwrites. The optimized package instead runs the GPU steps in one seeded process and skips that initialization.
- **Asynchronous structure writer (all modes).** Output CIF and npz files for each call are written by background processes while the next call’s computation proceeds, overlapping file writing with GPU work without changing the written bytes.
- **Chunked distance features (Big only).** Big therefore runs eager: pairwise token-distance features, upstream’s first out-of-memory point, are built in 64-row chunks, and the GPU memory allocator uses expandable segments, with the graph sampler and mask hoist switched off so their buffers are not held.

Results. Fast gives the largest design-phase speed-up, from 1.30× default at 100 tokens to 2.19× at 500 tokens. Exact stays close to parity at small and mid-size inputs, only pulling ahead of default at the largest input size, where the mask hoist has the most work to remove. Big trails default below 500 tokens, since it runs eager and targets memory rather than speed. It nevertheless uses less peak GPU memory than default at every input size, for example 0.84× default’s peak at 500 tokens. Whole-pass speed-ups

exceed the design-phase ratios for every mode because default compiles its model in every process (about two minutes), starts up more slowly, and writes its output files synchronously. The optimized modes avoid or shorten all of these steps, and start-up and compilation shrink as a share of the total as more designs are drawn per process.

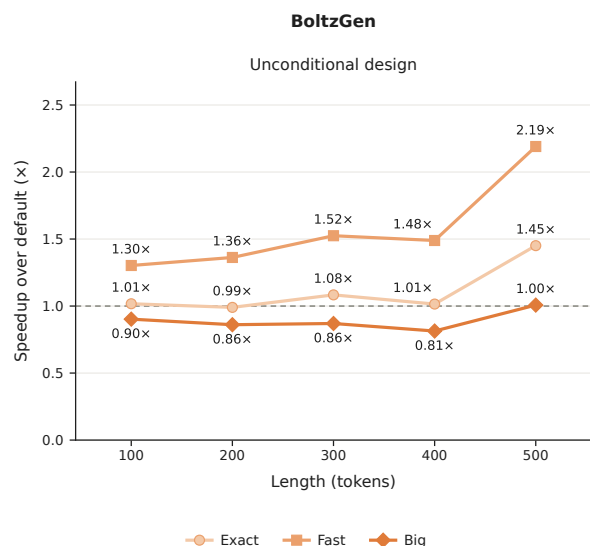


Figure S90. BoltzGen: design-phase speed-up over default on H100. Speed-up relative to default versus input size (tokens) for unconditional single-chain design, one line per mode (Exact, Fast, Big), all at diffusion batch size 64; the dashed line marks default (1×).

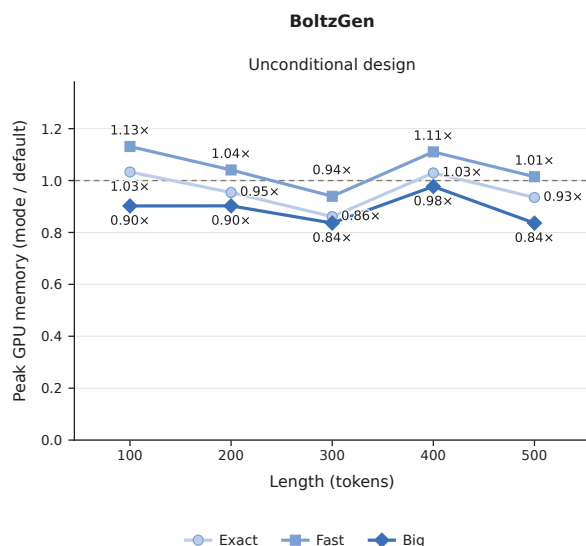


Figure S91. BoltzGen: peak GPU memory relative to default on H100. NVML device peak memory above idle baseline, measured in dedicated memory runs, relative to default versus input size (tokens), one line per mode (Exact, Fast, Big); the dashed line marks default (1×).

S4 Inverse folding

These models design amino-acid sequences for a given backbone structure. Speed-ups are for the end-to-end wall time of a design pass, including model loading and other one-time costs. Section S7 compares the quality of designs made under each mode.

S4.1 ProteinMPNN

ProteinMPNN is a 1.7-million-parameter message-passing network for fixed-backbone protein sequence design. It encodes each residue's local structural neighborhood once per backbone, then autoregressively decodes one designed residue per step [4]. The benchmark used the dauparas/ProteinMPNN repository (no release tag or package) with the soluble weight set (SolubleMPNN).

Benchmark setup. The task is fixed-backbone design on five input sizes of 100 RFdiffusion-generated single-chain backbones each, at lengths of 100, 200, 300, 400, and 500 residues. The number of backbones processed concurrently is fixed at 8 for both default and Exact, so that neither is compared at its own preferred concurrency. Speed uses 5 input sizes $\times$ 3 repeats (15 jobs), each preceded by an untimed warm-up pass. Peak memory uses one dedicated job per input size. All runs use a single H100 80 GB GPU, one job per GPU, exclusive. The headline speed-up is the end-to-end design wall clock – process launch to the last output file written for the full pass, including one-time costs such as weight loading. A companion sampling-span clock, timing only the steady design window after warm-up, is reported as context, not as the headline. Peak GPU memory is the device high-water mark measured externally (NVML) above an idle baseline. The framework allocator's own peak, recorded separately, is smaller because it excludes reserved memory pools and the CUDA context. For this model, default adds CUDA MPS (a service that lets several processes share one GPU concurrently) to pack eight unmodified upstream processes onto the card, each designing its share of the backbones one at a time, because the upstream script has no cross-backbone batching and a single process cannot saturate the GPU. Default also fixes an explicit random seed, shared with Exact, so the two configurations' outputs can be compared byte for byte. The speed-ups therefore compare Exact with the unmodified package packed eight processes deep rather than with an idle single process. Eight processes matches Exact's concurrency and is not claimed to be default's fastest packing (larger packings were not tested). Measured separately against this packed baseline, base – the unmodified package run as a single process, here with the same eight-sequence batch (upstream's shipped batch of one sequence would be slower still and was not measured) – takes 42.8 s end to end at 100 residues against default's 15.0 s ($0.36\times$ of default's speed as a paired ratio from the same runs), falling to $0.21\times$ at 500 residues. Packing, not any change to the numerics, accounts for the difference.

Sampling settings. Default and every mode used the same sampling settings: 8 sequences per backbone, decoded as one batch (`--num_seq_per_target 8 --batch_size 8`; upstream default batch size 1), at temperature 0.1 with no backbone noise, using the soluble v_48_020 weights and writing no score or probability files.

Modes. Only one optimized mode exists for this model, Exact; there is no Fast or Big mode, and no reduced-precision path: every configuration runs in float32 with TF32 off. Default's seed is otherwise random from run to run, so bit-for-bit identity is defined only under a shared explicit seed. Under that seed, dedicated identity checks found Exact's sequence output files bit-identical to default's, and every one of the 15 timed speed runs likewise matched Exact's output against a seeded single-process run of the unmodified upstream script, all passing. Passes Exact does not implement – CA-only design, score-only or probability-only requests, tied positions, and non-zero backbone noise – are declined by name and instead run through base, the unmodified upstream package, at base's slower speed.

Optimizations. Exact leaves the network, sampler, and outputs unchanged and reorganizes how the work reaches the GPU, in roughly this order of contribution:

- **Cross-backbone batching in one resident process.** Many backbones are decoded together inside a single long-lived process instead of one backbone at a time in each upstream process, amortizing start-up cost and letting each decode step do more work per launch. This batching is estimated to contribute most of the gain.
- **CUDA-graph decode step.** The repeated per-residue decode step is captured once and replayed, removing the kernel-launch overhead that otherwise dominates this small model.
- **Batched neighbor-message GEMMs.** Matrix products for message passing across several backbones are merged into larger batched operations, enabled only after a start-up check confirms bit-identical results.
- **Fused sampling draw.** Per-backbone random draws at each decode step are replaced by one custom kernel that reproduces the same sampling exactly.
- **Stream pipelining and single-read input parsing.** The decode loops of up to three batches are overlapped on the GPU, and each input structure file is read once rather than once per candidate chain, which shortens input parsing outside the sampling loop.

Results. Across the five input sizes, Exact’s speed-up over default on the headline end-to-end design clock grows with backbone length, from 1.66× at 100 residues to 2.44× at 500 residues (geometric mean 2.09× over the five input sizes). The speed-up grows because more decode steps let the batching and CUDA-graph savings accumulate, while Exact’s fixed start-up cost (about 6 s per pass) becomes a smaller share of each pass. The companion sampling-span clock, which excludes start-up and warm-up, shows a similar rise with length but starts below the headline ratio at the shortest length and finishes slightly above it at the longest. This is consistent with default’s start-up costs – eight concurrent process launches – weighing more heavily at short lengths. Peak GPU memory for Exact stays well below default’s at every input size, since default’s figure includes eight CUDA contexts and model copies while Exact holds one of each. Exact’s ratio rises from 0.35× to 0.48× with length because its activations for the same 64 sequences per step grow with length. Memory does not appear to limit either configuration at these sizes.

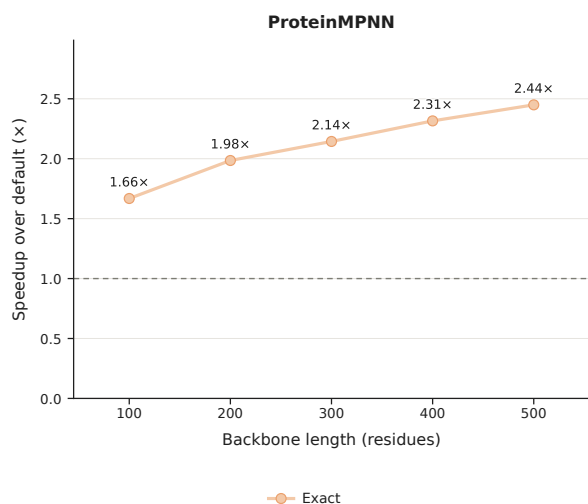


Figure S92. ProteinMPNN: speed-up over default on H100. Speed-up on the end-to-end design wall clock (the headline clock), plotted against backbone length (input size), for the Exact mode relative to default; the dashed line marks default (1×).

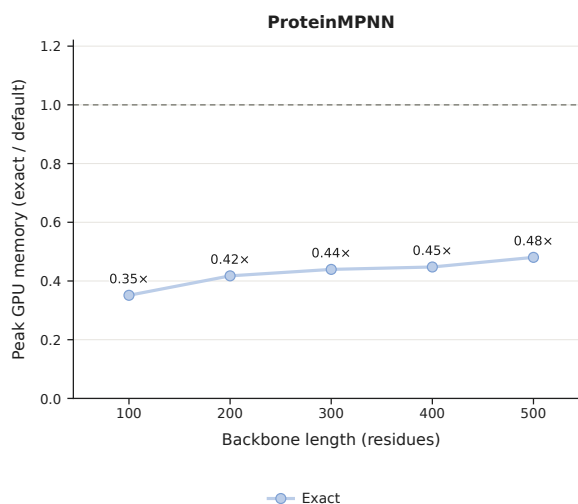


Figure S93. ProteinMPNN: peak GPU memory relative to default on H100. Peak device memory (NVML high-water mark above idle, from dedicated single-configuration memory runs) for Exact relative to default, plotted against backbone length; the dashed line marks default (1×).

S4.2 Caliby

Caliby is a Potts-model sequence designer. An MPNN-style network evaluates a protein backbone once and emits per-position fields plus sparse pairwise couplings; sequences are then drawn by annealed discrete-Langevin sampling (500 sweeps, temperature 1.0 $\rightarrow$ 0.01) under a low-complexity penalty. We benchmarked Caliby (ProteinDesignLab/caliby) as of its main branch in July 2026, 33 commits after its latest tag, v0.4.0 (the package’s own version string still reads 0.1), with the `soluble_caliby_v1` weights [49].

Benchmark setup. Two workloads were timed on an H100 80 GB GPU, one exclusive process per job. A single-backbone workload designs 100 RFDiffusion-generated backbones per input size, and an ensemble workload designs 25 backbones each expanded into a 32-member conformer ensemble (conformers from ProtPardelle-1c). Both request 8 sequences per backbone across five input sizes (100–500 residues). On the single-backbone workload, each configuration ran at its own largest batch of backbones per call that fit, found by probing batch sizes up to 100 for the largest that kept peak memory under 90% of the device. Batches matched except at the largest input size, where the optimized package was limited to 32 backbones per call against default’s 64. On the ensemble workload, every configuration designs one backbone’s ensemble per call. Each input size was measured over three replicate jobs with an untimed warm-up pass, ratioed against the two bracketing default passes. The headline clock is the end-to-end design wall per pass, from process start to the last output written, with the sampling window (start-up excluded) carried alongside as a companion clock. Peak GPU memory is the NVML device high-water mark above idle from dedicated memory runs. Here default is the upstream package run through its documented Python interface, including its structure-cleaning and featurization steps, at the largest batch that fits. Base is the example script shipped with it, which designs 4 backbones per call and has no cleaning step. Base was run with the same 8 sequences per backbone and the same `soluble_caliby_v1` checkpoint as every other configuration, passed explicitly; the script’s own defaults (1 sequence per backbone and the `caliby` checkpoint) reproduce neither. On the sampling window alone, base is slower than default at every input size on both workloads. On the end-to-end clock, however, base finishes sooner than default at 300–500 residues on the single-backbone workload (1.02–1.17 $\times$ default’s speed) and at every input size on the ensemble workload (1.09–1.35 $\times$), because default’s pass also includes the benchmark’s warm-up batch, the cleaning step, and whole-batch featurization before the first design call, none of which the example script performs. Fast and Exact are faster than both default and base on both clocks at every input size.

Sampling settings. Default and every mode used the same sampling settings, which are the tool’s defaults apart from the number of sequences: 8 sequences per backbone (upstream default 1), drawn by 500 annealed discrete-Langevin sweeps with the temperature lowered from 1.0 to 0.01, with every position designed and no positional constraints, using the `soluble_caliby_v1` checkpoint. The number of backbones per call is given above; on the ensemble workload, each call designs one backbone together with its 32 conformers (the input backbone and 31 generated by ProtPardelle-1c).

Modes. Fast serves the single-backbone workload; Exact serves the ensemble workload. Exact’s outputs were verified bit-identical to default’s, byte for byte across CSV fields and structure files, including across hosts. Fast is deterministic, though its floating-point results can differ from default’s in the last float32 bit, occasionally flipping a sampled residue. Its accuracy guard is distributional, checking per-design sequence identity against default’s own spread across seeds, with zero excursion beyond that spread. Fast’s design quality is compared with default’s in Section S7.3, where Fast designed the same sequences as default for 499 of 500 backbones.

Optimizations. The changes below apply to both modes and are ordered by their estimated contribution to the measured gain (no ablation is available to confirm this order). The ensemble workflow (Exact) also speeds up conformer handling in additional ways not detailed here.

- **Graph-captured sampler.** The 500-step sampling loop is recorded once as a CUDA graph (the sequence of GPU operations captured and replayed without a host round trip at each step) and replayed thereafter, removing per-step host synchronization; this is estimated to give most of the gain.
- **Concurrent multi-sequence sampling.** The 8 requested sequences per backbone are sampled at the same time, using one GPU stream and random-number generator per sequence, instead of one call per sequence.
- **GPU-side sparse Potts parameters.** The per-position couplings to each residue’s 48 neighbors are aggregated and kept sparse on the device instead of being expanded into a dense residue-by-residue matrix.
- **Fused low-complexity penalty kernel.** A single custom Triton kernel replaces a chain of elementwise and reduction operations used to compute the penalty term, compiled once at first use and cached.
- **Background output writing.** Each batch’s sequences and structure files are written on worker threads while the GPU samples the next batch, overlapping file I/O with compute.

Results. On the single-backbone workload, the end-to-end speed-up over default rises with input size, reaching a geometric mean of 2.34 $\times$ across the five input sizes tested, as host-side work (start-up, cleaning, featurization), which grows more slowly than sampling time, becomes a smaller share of the total as backbones get longer. The rise from 2.35 $\times$ at 400 residues to 2.98 $\times$ at 500 residues is not batch-matched, because Fast ran 32 backbones per call there against default's 64. The sampling-window gain barely changes (5.16 $\times$ and 5.05 $\times$), and part of the rise comes from Fast's smaller warm-up batch, which the end-to-end clock includes. On the ensemble workload the gain is flatter, with a geometric mean of 1.75 $\times$, since conformer generation, only modestly accelerated by the optimized package, takes up a larger share of each pass. On the sampling window alone the speed-ups are substantially larger for both workloads (4.83–5.16 $\times$ for Fast and 3.40–4.11 $\times$ for Exact). The difference arises because the end-to-end clock dilutes the sampler-side gain with work that the optimized package shortens little or not at all: start-up, cleaning, featurization, and, on the ensemble workload, conformer generation. Peak GPU memory at matched batch size is little changed for Fast and moderately higher for Exact, reflecting extra sampling buffers and the resident conformer model. The one exception is the longest single-backbone input size, where Fast's memory reads 0.53 $\times$ of default's only because it was pinned to a smaller batch there (32 backbones instead of 64), not an actual saving.

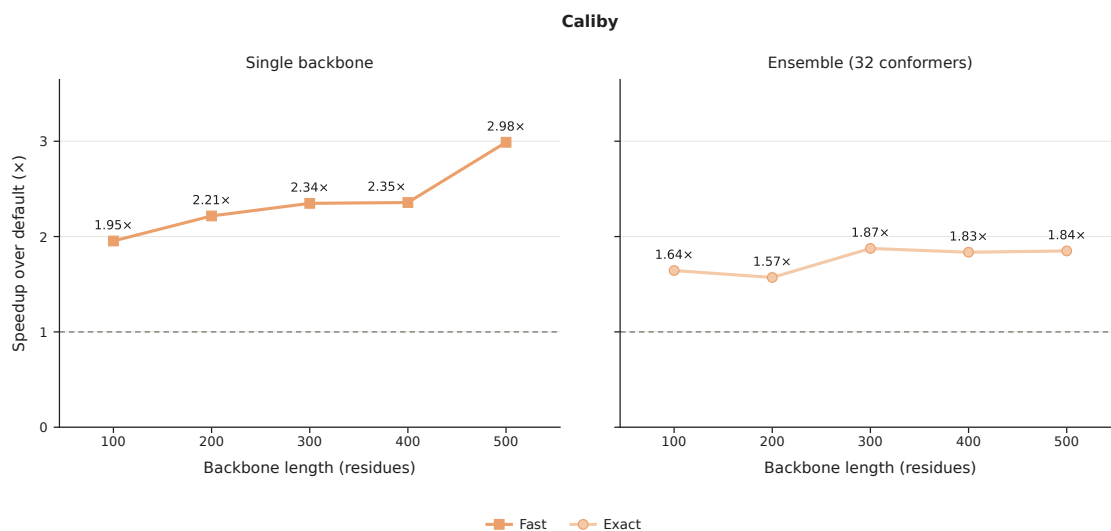


Figure S94. Caliby: end-to-end design wall speed-up over default. Speed-up vs. input size on H100, with one panel for the single-backbone workload and one for the ensemble (32-conformer) workload; each panel shows one mode (Fast on the single-backbone panel, Exact on the ensemble panel), each computed on the end-to-end design wall per pass (at 500 residues on the single-backbone panel, Fast ran 32 backbones per call against default's 64), with the dashed line marking default (1 $\times$).

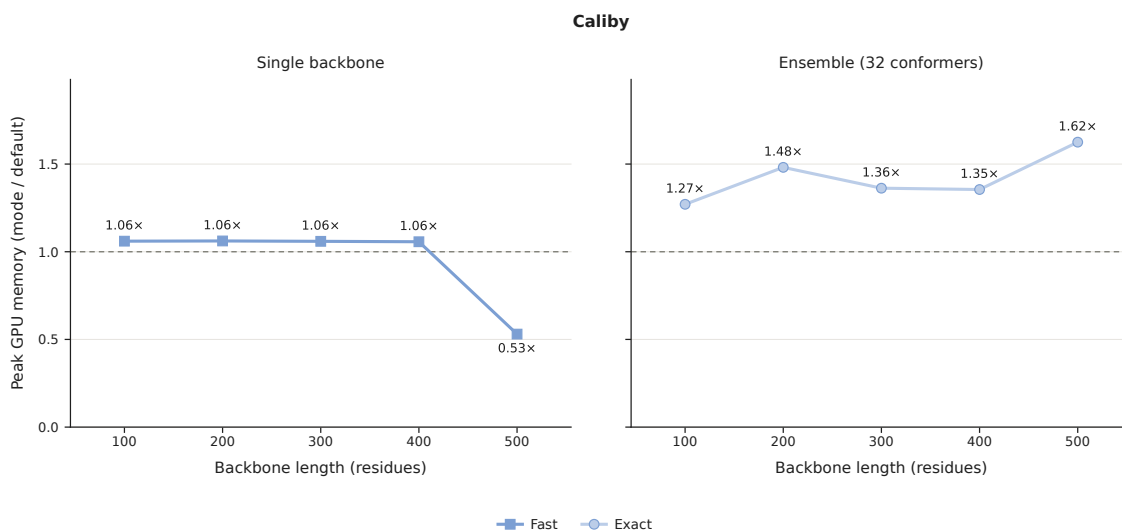


Figure S95. Caliby: peak GPU memory relative to default. Peak GPU memory (NVML device high-water mark above idle, from dedicated memory runs) relative to default vs. input size on H100, one panel per workload as above, one mode per panel (Fast on the single-backbone panel, Exact on the ensemble panel), with the dashed line marking default (1 $\times$).

S4.3 ESM-IF1

ESM-IF1 is an inverse-folding model that, given a protein backbone, samples amino-acid sequences consistent with that structure, using a geometric-vector-perceptron encoder and an autoregressive transformer decoder [21]. The benchmarked checkpoint is `esm_if1_gvp4_t16_142M_UR50` (142 M parameters, float32) from the upstream `facebookresearch/esm` repository [61].

Benchmark setup. The task is sequence design for 100 poly-glycine backbones (chain A) generated with RFdiffusion, at five input sizes, $L = 100, 200, 300, 400,$ and 500 residues, drawing 8 sequences per backbone at temperature 1.0 (800 designed sequences per pass). Runs were repeated three times per input size on an H100 80 GB GPU, with an untimed warm-up pass before the timed runs. The headline speed-up is the end-to-end design wall per pass – process launch to the last sequence written, with every one-time cost (start-up, checkpoint load, input parsing) included for both default and Fast. A companion sampling-span clock, which excludes those costs and reports only the steady-state design window, is reported as context in the Results. Peak GPU memory is the device-level high-water mark above idle, measured outside the running processes. Base, the upstream package run exactly as shipped, invokes the shipped example script once per backbone, one process at a time; each invocation is a fresh Python process that imports the framework and loads the 1.7 GB checkpoint. This leaves the GPU mostly idle because upstream exposes no batch dimension across sequences or backbones, and the per-backbone reload accounts for a large part of base’s gap to default. Default adds process packing on top of base: the same unmodified script is run as six concurrent processes under CUDA MPS (which lets several processes share one GPU at once), each running at batch 1. The six processes and Fast’s batch of 64 were chosen together to give both configurations a similar memory budget at 500 residues (64.3 and 57.3 GB). The same pair is used at every input size, without per-size calibration (denser packings at shorter lengths were not tested). Base itself is much slower than default: its end-to-end design rate is only $0.18\text{--}0.21\times$ default’s rate.

Sampling settings. Default and every mode used the same sampling settings: 8 sequences per backbone (`--num-samples 8`; upstream default 1) at temperature 1.0, designed for chain A of each of the 100 poly-glycine backbones per input size, with the single checkpoint `esm_if1_gvp4_t16_142M_UR50`.

Modes. Only Fast is offered for this model; there is no Big or large-input mode, and no Exact mode. Upstream’s own sampling is unseeded, so default’s outputs differ from run to run and cannot serve as a bit-for-bit reference. Fast’s speed comes from batched sampling, which draws different samples from the same distribution. Instead, Fast was checked distributionally, and the check passed: its designs (fixed seed, batch 64) had to fall, on a one-sided criterion, within the draw-to-draw variation of six independent unseeded runs of the unmodified script at 100, 300, and 500 residues. Fast’s design quality (self-consistency and native sequence recovery) is compared with default’s in Section S7.3.

Optimizations. All of the speed-up comes from one change, batched sampling in a single persistent process. The first four items below are its parts, and the last is the batch-size choice that shapes the comparison.

- **Batched sampling over rows.** Instead of sampling one sequence at a time, up to 64 (backbone, sample) combinations are processed together through the unmodified encoder and decoder, so each decoding step serves 64 sequences instead of one.
- **Persistent process, model loaded once.** The checkpoint and model are loaded once per pass instead of once per backbone, removing a repeated launch-and-load cost.
- **Single input-parsing pass.** Backbone coordinates are parsed once per run rather than separately for every backbone.
- **Inference mode.** Sampling runs under `torch.inference_mode`, so no autograd (gradient-tracking) graph is recorded, cutting overhead and memory that the shipped script otherwise uses.
- **Equal-memory-budget batch size.** The batch size (64) was chosen so that Fast’s peak memory at the largest input size is close to, and below, the packed default’s ($0.89\times$), matching memory use rather than item count.

Results. On the end-to-end design-wall clock, Fast’s speed-up over default is largest at the shortest backbones and narrows as input size grows, from $12.5\times$ at $L=100$ to $8.9\times$ at $L=500$, with a geometric mean of $10.0\times$ across the five input sizes. The sampling-span clock, which excludes one-time start-up and loading costs, shows a larger advantage, from $25.0\times$ at $L=100$ to $11.7\times$ at $L=500$. It narrows with input size because default’s design window includes a process launch and checkpoint load for every backbone, which account for a larger share of that window at short lengths. Fast’s own one-time costs, a large share of its short run at small sizes, roughly halve its end-to-end gain there. Peak GPU memory for Fast is $0.81\text{--}0.98\times$ default’s packed footprint across input sizes, consistent with matching memory use rather than minimizing it.

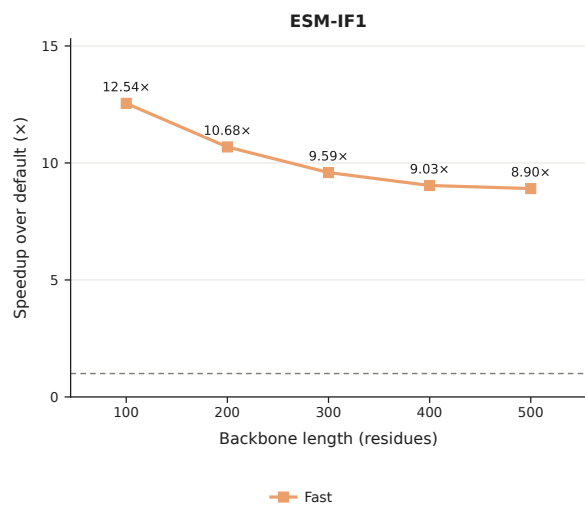


Figure S96. ESM-IF1: speed-up over default vs. input size on H100. End-to-end design-wall speed-up of Fast relative to default (six CUDA-MPS-packed upstream processes) across backbone lengths 100–500 residues; the dashed line marks default (1 $\times$).

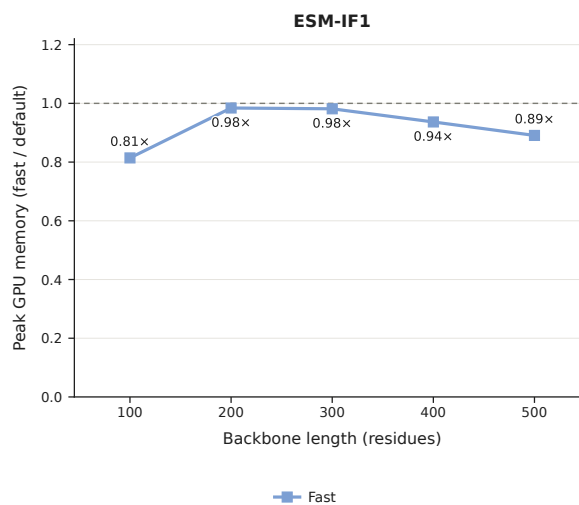


Figure S97. ESM-IF1: peak GPU memory relative to default on H100. NVML device-level peak memory above idle, from dedicated memory runs, for Fast shown as a ratio to default's packed footprint across backbone lengths; the dashed line marks default (1 $\times$).

S5 Genomics models

These models read DNA sequence to predict functional genomic signals, such as gene expression, chromatin accessibility or evolutionary constraint, or, for Evo 2, to score and generate DNA. Because reading and preparing inputs and writing outputs can take longer than the model itself, the three models benchmarked on an end-to-end task (Borzoï, Flashzoi, and ChromBPNet) are shown with the wall-time speed-up of the whole task as well as the forward-pass speed-up.

S5.1 Evo 2

Evo 2 is a genomic language model that scores and generates DNA one base at a time with the StripedHyena 2 architecture, run through upstream’s vortex inference stack [22]. We pin upstream evo2 0.6.0 on vtx 1.1.0 and benchmark the released evo2_7b (one GPU) and evo2_40b (two GPUs, upstream’s own layer split) checkpoints.

Benchmark setup. Two task families are timed: scoring, a raw forward pass over 8,192-bp genomic windows (64 windows per job, one window per call in both configurations; evo2_40b additionally scored 16 sequences per call through upstream’s `score_sequences` entry point), and generation, the paper’s mitochondrial-genome task – a 3,000-nt human mtDNA (rCRS) prompt continued for 16,000 tokens with cached generation, top- k 4, temperature 1.0, one prompt per call. Every job runs on one exclusive NVIDIA H100 80 GB GPU (two for evo2_40b), loads weights once, and runs untimed warm-up calls, then a timed steady-state window. The reported speed-up is the median over three independent jobs. The headline clock is the timed model call in steady state – one forward pass for scoring, the full generation call (reported per token) for the mtDNA task – excluding warm-up and compilation. A companion whole-process wall time, covering imports, checkpoint loading, and, for generation, the entire call, was derived for orientation only and is not reported here. Peak GPU memory is the device high-water mark read through NVML from outside the process, above the card’s idle baseline. This peak reflects each configuration’s true residency rather than an artifact of a larger probed batch, because both configurations score one window per call at all times. In this benchmark, default means upstream run through its own documented, fastest interface: for evo2_7b the constructor is called with `use_kernels=True` (the README’s own speed opt-in enabling vortex’s Triton Hyena kernels), and for evo2_40b the constructor defaults are used, since the kernel switch is numerically invalid across the two-device split. No allocator, precision, or process-packing settings are added on either configuration. This isolates the optimized package’s own kernel-level contribution rather than crediting an unfairly weak baseline. A single 8,192-bp window already keeps one H100 more than 90% busy for the 7B model, so windows were scored one per call (larger batches were not measured). The gap between base and default is notable for evo2_7b: base (`use_kernels=False`, upstream’s constructor setting) scores a window at 0.82× default’s rate, with higher memory. For evo2_40b, base and default are identical, because the kernel switch cannot be used across the two-GPU split.

Sampling settings. Default and every mode used the same sampling settings, described above: scoring uses one 8,192-bp window per call, and generation continues a 3,000-nt prompt for 16,000 tokens with top- k 4 and temperature 1.0, one prompt per call. evo2_7b runs on one H100 GPU and evo2_40b on two, split across the GPUs by upstream’s own layer assignment.

Modes. Exact and Fast modes exist for Evo 2. Exact changes only kernel-level implementation, caching, and scheduling, not arithmetic order, and was verified bit-identical to default on every checked surface: scoring windows, `score_sequences`, teacher-forced logits, and greedy and seeded sampled tokens. Fast applies only to generation, adding speculative sampling with the small evo2_1b_base checkpoint as draft on top of Exact’s scoring path. Its per-step logits and scoring were verified bit-identical to default, and by construction each emitted token is an exact draw from the target model’s own sampling distribution given the prefix. However, the concrete sampled sequence for a given seed differs from upstream’s, because the random stream is consumed differently.

Optimizations. The first five changes below set scoring speed and are listed in roughly descending order of estimated contribution:

- **Hyena filter and spectrum caching.** Each layer’s long-convolution filter and its frequency-domain representation are built once per sequence length and reused across calls instead of being recomputed on every forward pass.
- **Persistent Fourier-transform convolution plumbing.** The Fourier-transform-based long convolutions reuse prebuilt transform plans and memory buffers instead of rebuilding and reallocating them on every call, avoiding repeated allocation and copy work.
- **Fused Triton epilogues.** Small operations around each convolution and normalization step – gating, format casts, and RMSNorm with its neighboring bias and residual additions – are merged into single GPU kernel launches (using Triton, a GPU-kernel programming tool), so intermediate results never need to be written out separately.
- **FP8 projection housekeeping (H100).** Weight-layout folding and FP8 (an 8-bit numeric format used for faster matrix multiplication) casting for upstream’s own input projections from NVIDIA’s Transformer Engine library are precomputed and cached once rather than repeated on every call, without changing the projections’ arithmetic.

- **Scoring launch overhead.** A recurring scoring shape is replayed as one CUDA graph, rotary tables are reused, and a faster matrix-multiply algorithm is used only where it matches PyTorch’s result bitwise.
- **Decode-step CUDA graph replay (generation).** The one-token decode step is recorded once per generation call as a CUDA graph and replayed for the remaining tokens, removing the per-step launch overhead that limits upstream’s decoding (most of the generation gain). The per-token filter-state update runs as one fused kernel per block.
- **Two-GPU pipelining (evo2_40b).** Consecutive windows of a `score_sequences` call are pipelined across the two GPUs (scheduling only).

Results. Scoring’s speed-up reflects direct kernel-level savings, because scoring is device-bound in both configurations: `evo2_7b` Exact scoring reaches 1.57× per window, with `evo2_40b` showing a similar per-window gain and its multi-sequence `score_sequences` call gaining further (2.73×) from pipelining consecutive windows across its two GPUs. Generation shows larger gains, which depend more strongly on model size, because upstream’s decode loop is limited by per-step launch overhead rather than GPU compute: `evo2_7b` Exact generation reaches 3.51× on the per-token clock, and Fast, which adds speculative sampling, reaches 4.90× (Exact and Fast were timed against separate default runs whose per-token times differ by about 10%, so these two values are not directly comparable); `evo2_40b` gains far less under Exact, since its two-GPU decode step is limited by streaming weights across devices rather than by launch overhead, and only Fast brings a substantial improvement. Peak GPU memory stays close to default across configurations, with a modest increase for `evo2_7b` scoring from the persistent filter, spectrum, and transform buffers that accelerate the forward pass.

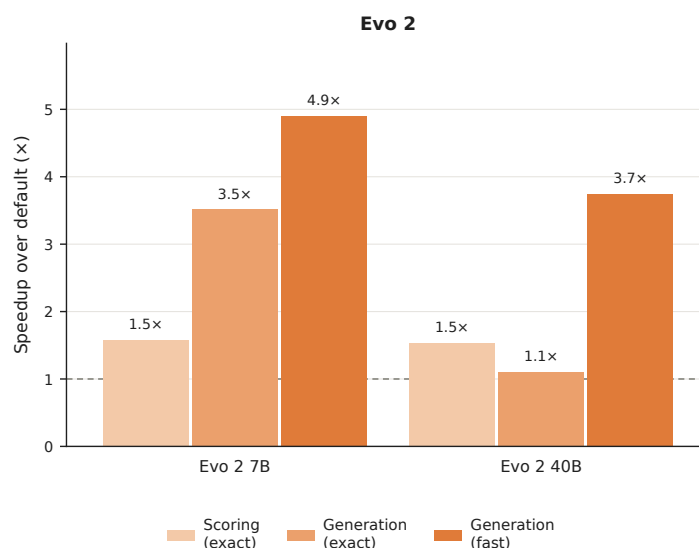


Figure S98. Evo 2: speed-up over default on H100. Bars show the forward-call speed-up for scoring (Exact) and the per-token generation speed-up for Exact and Fast, for `evo2_7b` and `evo2_40b`; bar labels are truncated to one decimal (p. 16); the dashed line marks default (1×).

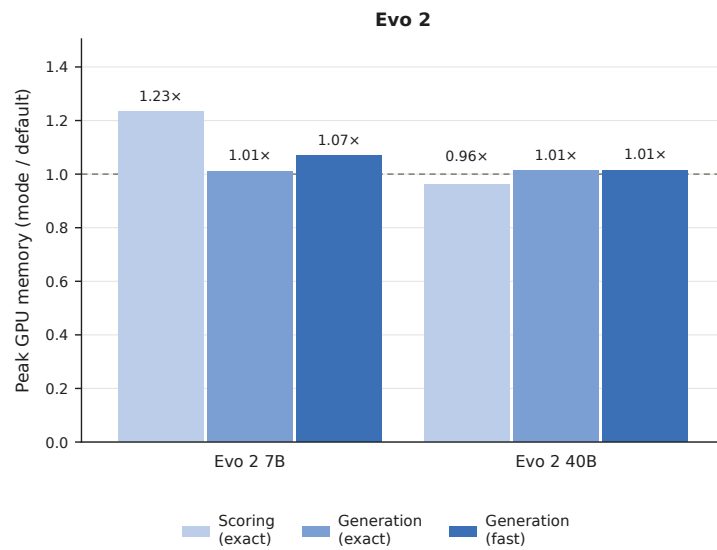


Figure S99. Evo 2: peak GPU memory relative to default on H100. Bars show the NVML device peak (scoring: measured during the timed forward runs; generation: measured in dedicated memory runs) for scoring (Exact) and generation (Exact, Fast), for `evo2_7b` and `evo2_40b`; the dashed line marks default (1x).

S5.2 Borzoi

Borzoi is a convolutional network with a tower of eight self-attention (transformer) layers that predicts RNA-seq and other genomic coverage tracks at 32-bp resolution from a fixed 524,288-bp DNA window [6]. We benchmarked the upstream Calico `borzoi` release together with its `baskerville` dependency, using the published human fold-0 checkpoint.

Benchmark setup. The task is upstream’s variant-scoring command. For each variant, it one-hot encodes a reference and an alternate 524,288-bp window, calls the model once on the pair (a batch of two windows), and computes SAD, logSAD, D2, and logD2 statistics across all 7,611 tracks. Two regimes are timed on a single H100 80 GB GPU: a forward-call regime (batch of 2 windows, 256 GTEx eQTL variants per pass, 20 untimed warm-up calls) and a whole-task regime (the documented command run end to end on 100 GTEx SNVs). The headline clock is the per-call forward time (steady state, warm-up excluded) for the forward-call regime and the whole-command wall time, from process launch to the output file closing, for the task regime. Speed-ups are default’s time over Exact’s. Peak GPU memory is the device’s peak memory use above its idle baseline, read through NVML from outside the process in dedicated runs. TensorFlow’s in-process allocator peak is recorded alongside only as a diagnostic. For the forward-call regime, default is simply the upstream package run exactly as shipped (base), since process packing does not apply to a per-call timer. For the whole-task regime, default adds one thing beyond base: six concurrent unmodified worker processes (upstream’s own multi-worker mode) packed on the GPU under CUDA MPS. This configuration is the fastest of the worker counts we calibrated for default (2, 4, and 6) and the most that fit on the 80 GB card. This choice keeps the comparison against the fastest correct configuration of the unmodified tool rather than an artificially slow one – base run as a single process reaches only 0.22× default’s whole-task speed (9.38 vs 2.10 s per SNV).

Sampling settings. Default and every mode used the same sampling settings, those of upstream’s documented variant-scoring command. Each call scores one variant as a batch of 2 windows (reference and alternate, 524,288 bp each) with one model (fold 0), averaging forward and reverse-complement predictions (`--rc`; off by default). Scores cover all 7,611 tracks (`-t`), with the untransform option (`-u`) and the SAD, logSAD, D2, and logD2 statistics (upstream default SAD only).

Modes. Only Exact exists for Borzoi: there is no Fast or Big mode, since the model always scores one fixed-size window on a single GPU and no reduced-precision path is offered. Exact was verified bit-identical to default, on the same host and GPU, by comparing SHA-256 digests of the forward-pass and 100-SNV task outputs (all 7,611 tracks and an 89-track subset), plus a separate four-case standard suite that also covers upstream’s multi-worker form and the in-process forward call. Both checks passed.

Optimizations. All changes act on dispatch or on the host code around the model call, leaving the model’s arithmetic untouched, so outputs stay bit-identical.

- **Traced-graph forward.** After the first (eager) call, the model is traced once as a TensorFlow graph (XLA off), so later variants run as one graph instead of dispatching operations one by one from Python, with the same kernels in the same order. This traced graph gives the whole forward-call speed-up.
- **Pipelined, column-chunked post-processing.** Upstream post-processes each variant serially while the GPU idles. The optimized package overlaps this host work, split into output-column chunks across a thread pool, with the next variant’s forward call, so the GPU is not left waiting.
- **Lookup-table one-hot encoding.** The per-base Python loop that encodes each window is replaced by a table lookup, speeding up input preparation.
- **Copy-free forward return.** Upstream copies the prediction to the host and then copies it again in a same-type conversion; this second, host-side copy is dropped.
- **Exact statistics writer.** Only the log/sqrt intermediates needed for the requested output statistics are computed, following upstream’s own expressions.

Results. The forward call is 1.84× faster under Exact than default at the same batch of 2 windows. On the whole variant-scoring task, Exact (one process) is 2.33× faster than default’s six MPS-packed processes, a larger gain than the forward-call speed-up. The optimized package also speeds up input preparation and runs the host-side post-processing that dominates default’s task time alongside the next forward call. As a result, its task loop is spent mostly in the model call, whereas default’s six packed processes interleave host-bound work on one GPU. Peak GPU memory falls to 0.28× default’s task-level peak, since one process replaces six packed workers on the card. On the single forward call, Exact’s peak memory instead rises to 1.69× default’s. The traced graph keeps extra intermediates alive, which pushes TensorFlow’s growing allocator into one further reserved region, so this device-level ratio partly reflects reserved block size rather than memory in use.

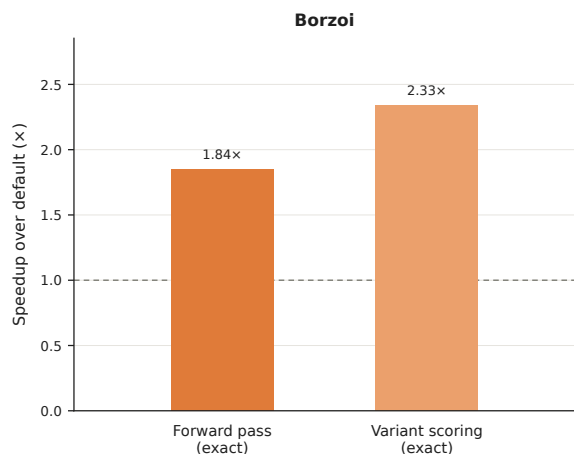


Figure S100. Borzoï: speed-up over default on H100. Bars show the speed-up of Exact over default for the forward-pass call and the whole variant-scoring task, on the headline clock (per-call time for the forward pass, whole-command wall time for the task); the dashed line marks default (1×).

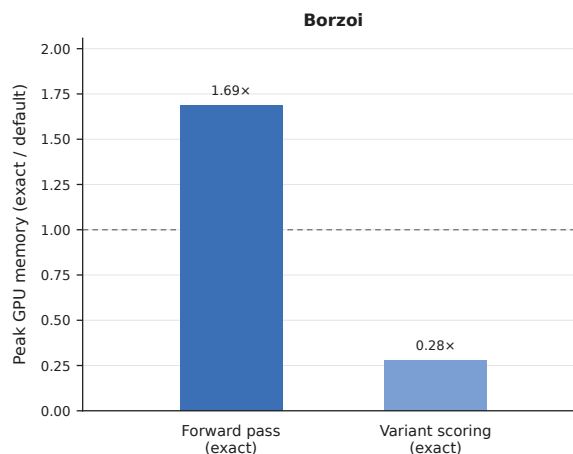


Figure S101. Borzoï: peak GPU memory of Exact relative to default. Bars show the ratio of Exact's peak device memory (NVML, measured in dedicated runs) to default's, for the forward-pass call and the whole variant-scoring task, where default for the task is six packed worker processes under CUDA MPS; the dashed line marks default (1×).

S5.3 Flashzoi

Flashzoi is a FlashAttention-2 variant of Borzoi [50], a convolution–transformer model that maps a single 524,288-bp one-hot DNA window to predicted coverage over 6,144 bins of 32 bp for 7,611 human genomic tracks. We benchmark the upstream PyTorch port *borzoi-pytorch* 0.5.1 (tag v0.5.1) [62], using the four published replicate checkpoints.

Benchmark setup. Two regimes are used. In the forward-pass regime, both configurations run 15 windows per call – upstream’s maximum batch, since larger batches raise an error – cycling eight distinct 524,288-bp hg38 windows under float16 autocast. The headline clock is the median per-call time divided by 15 windows. Upstream offers no speed switch, so here default equals base, run simply at its largest batch. In the task regime we score the first 100 GTEx fine-mapped eQTLs (200 reference/alternate windows, four replicate checkpoints, all 7,611 tracks) one window per call. A steady-state per-window loop clock is reported alongside the whole-command wall-time headline. Base (one unmodified process) leaves the GPU only about 6% busy here, so default instead runs eight concurrent copies of the unmodified package under one CUDA MPS daemon (a mechanism for sharing one GPU across several processes) – the largest of the process counts we calibrated (2, 4, 6, and 8; per-window time was still falling at eight, and larger packings were not tested) – giving the fastest configuration we found without editing code. A single unmodified process reaches only about one-fifth ($0.21\times$) of this packed default’s whole-command rate ($0.15\times$ on the loop clock). This confirms that packing, not any code change, drives default’s task-regime speed. All runs use one exclusive H100 80 GB GPU. Speed measurements follow an untimed warm-up and are medians over three independent jobs. Peak GPU memory comes from one dedicated memory run per regime and is the device high-water mark read through NVML from outside the process; the PyTorch allocator’s own peak is recorded alongside but never substituted for it.

Sampling settings. Default and every mode used the same sampling settings. Flashzoi is deterministic. The forward-pass regime runs batches of 15 windows of 524,288 bp per call (upstream default 1; 15 is the largest batch upstream accepts) with the first of the four published replicate models (`johahi/flashzoi-replicate-0`); the eQTL regime runs all four replicates, one window per call.

Modes. The optimized package offers a single mode, Exact, which reproduces upstream’s own float16-autocast numerics with no approximation. Exact’s outputs were verified bit-identical to default (matching sha256 hashes, zero maximum deviation) on the pinned software stack, with no determinism flags required. No reduced-precision Fast mode or memory-optimized Big mode is offered for this model.

Optimizations. The optimized package fuses and reorders computation inside the model call while leaving the checkpoints, weights, and public API unchanged:

- **Fused convolution tower.** Every pooling–normalization–activation block, and the initial convolution that scans the DNA sequence, is collapsed into single fused GPU kernels instead of many separate steps, removing redundant memory traffic. This fusion contributes most of the overall speed-up.
- **Early crop before upsampling.** The output is cropped down to only the genomic bins that will be returned before the upsampling and decoder stages run, so those stages process a shorter sequence.
- **CUDA-graph replay.** The forward pass is recorded once per input shape (about 0.2 s) and replayed directly on the GPU thereafter, avoiding repeated per-call launch overhead, since the fixed input length causes shapes to recur.
- **Asynchronous host-side output path.** Each replicate’s output is copied to pinned (page-locked) host memory in the background while the next replicate’s forward pass computes, overlapping data transfer with compute.
- **Output head as one batched matrix multiply.** The final per-track prediction layer is rewritten as one large fused matrix multiply, with bias and activation folded in, in the same TF32 arithmetic that upstream’s cuDNN convolution already uses for this layer. Its output is therefore unchanged.

Results. Exact is faster than default on both regimes while remaining bit-identical to it, but the two headline clocks diverge. On the forward-pass regime, Exact reaches $1.81\times$ default’s per-window rate; on the GTEx task, whole-command wall time reaches a smaller $1.59\times$ ($1.82\times$ on the steady-state loop clock). The prediction call is under a third of per-window time in every configuration, so input handling and disk writing, which Exact leaves untouched, dilute the much larger gain on the call itself. Process start-up with checkpoint loading, a fixed cost, takes a fifth (default) to a third (Exact) of the command’s wall time. The loop-clock gain still lands near the forward-pass gain because default’s eight processes contend for the same disk and memory bus. Peak GPU memory also falls for Exact, most sharply on the task regime, where default’s eight packed processes require far more memory than Exact’s single process ($0.19\times$).

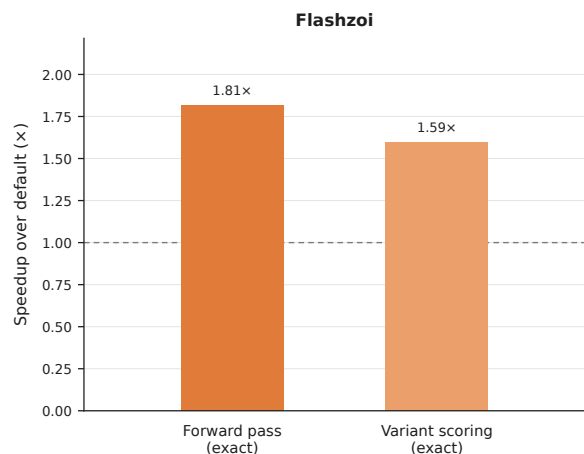


Figure S102. Flashzoi: speed-up over default on H100. Bars show default's time divided by Exact's time, one bar for the forward-pass call (15 windows per call, timed per call) and one for the whole-command wall time of the 100-SNV GTEx variant-scoring task (200 windows, timed end to end); the dashed line marks default (1×).

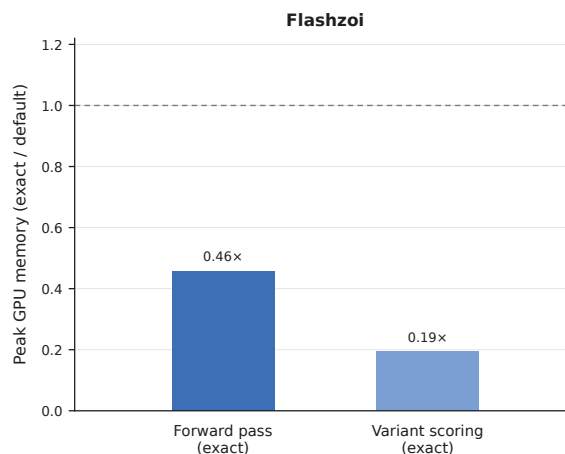


Figure S103. Flashzoi: peak GPU memory, Exact relative to default. Bars show the device peak-memory high-water mark (NVML, dedicated memory runs) for Exact divided by default, one bar for the forward-pass call and one for the GTEx variant-scoring task; the dashed line marks default (1×).

S5.4 ChromBPNet

ChromBPNet (Pampari et al., Kundaje lab) is a bias-factorized, base-resolution convolutional model of chromatin accessibility: from a 2,114-bp DNA window it predicts a 1,000-bp accessibility profile and one log-count. We benchmarked upstream `chrombpnet` 1.0.1 (kundajelab/chrombpnet tag v1.0.1) [51], using the published ENCODE GM12878 ATAC-seq fold-0 model.

Benchmark setup. We timed two regimes on held-out peaks from test chromosomes chr1/3/6. The forward-pass regime is the in-process model call on 1,024 regions per call ($4 \times 1,024$ held-out peaks). The whole-task regime is the documented `chrombpnet pred_bw` command run end to end over all 62,929 held-out regions with the observed track. Forward timings exclude an untimed cold call and warm-up calls, then time a window of at least 200 calls (at least 60 s), reduced to a median. Each task job runs an untimed default warm-up pass and then default, the optimized mode, and default again, and its speed-up (the mean of the two default passes over the mode’s pass) is reported as the median over three independent jobs. All runs used one exclusive NVIDIA H100 80 GB GPU. The headline clock is the forward call: time per item inside the model’s forward/prediction call, steady state, with warm-up and compilation excluded. The declared clock for the whole-task regime is the whole-command wall time (process launch to output files closed); a secondary loop clock (first batch to last output) is also recorded. Peak memory is the device high-water mark read via NVML from outside the process over the whole pass, above the card’s idle baseline, with the TensorFlow allocator’s growth mode forced on for every configuration so that default’s memory reading reflects its working set rather than a pre-reserved pool. In-process allocator figures are kept alongside but never substituted for this meter. Default here is the upstream command run with its own documented throughput flag set to the largest value that runs, `-bs 1024` (the parser’s shipped value is 64), with the driver’s compute cache pre-populated so that default is never timed compiling. Otherwise the command is unmodified. This choice keeps the comparison against the fastest correct configuration reachable through upstream’s own interface and keeps the GPU saturated rather than latency-bound at batch size 1. No separate base result was measured or is reported for this model, because the only difference between this default and base (upstream run exactly as shipped) is that one batch-size flag.

Sampling settings. Default and every mode used the same sampling settings. ChromBPNet is deterministic and has no sampling: each call predicts a batch of 1,024 regions of 2,114 bp (upstream default 64) with a single model, the ENCODE GM12878 ATAC-seq model for fold 0, in which the bias model and the bias-corrected model are combined in one file.

Modes. Exact and Fast are both available; there is no Big mode for this fixed-geometry model. Exact applies TensorFlow’s determinism settings (seed 0, deterministic operations, TF32 disabled) and was verified byte-identical, on every unit checked, to default run under the same settings, including the accessibility-profile logits and log-counts and every output file of the command-line cases. Default as normally run (TF32 on, non-deterministic reductions) is not byte-reproducible even against itself. Fast is the mode the optimized package runs when none is specified: the same forward kernels as Exact, but with convolutions run in TF32 (a lower-precision numeric format) on tensor cores, which is the numeric precision the unmodified model ships with. It is therefore not bitwise-identical to default, but on every checked unit its outputs were within default’s own pass-to-pass variation.

Optimizations. The changes below are ordered by their estimated share of the measured gain, largest first.

- **Triton forward kernels.** The model’s forward pass is rewritten as custom GPU kernels (built with Triton, a GPU-kernel programming tool) that replace the original TensorFlow/Keras graph. In Fast mode, these kernels additionally use TF32 on tensor cores (the GPU’s matrix-multiply hardware) for extra throughput. This is the largest contributor to the forward-call speed-up. On the whole task, the pipelined writer below saves more, since writing output takes about half of default’s task time.
- **Pipelined bigWig output writer.** Writing the predicted track (a bigWig file, a standard genomics track format) is moved to a side process fed chunk by chunk, so the write overlaps the forward pass instead of following it. This removes most of the time that default spends writing after the model has finished.
- **Featurization and prefetch.** The genome is memory-mapped and one-hot encoded via a lookup table, with the next chunk of input prepared on a background thread while the current batch is still being processed.
- **Overlapped finish stage.** With an observed track supplied, reading that track, computing agreement metrics between predicted and observed signal (such as the profile Jensen–Shannon divergence), plotting, and writing final files are started early and run concurrently rather than sequentially after the forward pass.
- **Batch shaping, import preloading, and teardown.** Partial trailing batches are padded to a single fixed shape, calls above 1,024 regions are chunked at 1,024, late imports are preloaded on a thread, and teardown is skipped once outputs are written. Start-up still takes longer than default’s (9 s for Exact and 15 s for Fast, against 7 s), so jobs of a few thousand regions gain little.

Results. The forward-call clock shows a modest gain: at 1,024 regions per call, Exact reaches $1.57\times$ and Fast $2.07\times$ default’s per-region rate. The whole documented job over 62,929 regions shows a substantially larger whole-command speed-up, up to $4.00\times$

for Fast, because a large share of default’s wall time is spent writing output after the forward pass completes. The optimized package overlaps that writing with the faster forward pass, so job time collapses toward roughly the forward time plus a short finish stage, rather than scaling only with the forward-call gain. Peak GPU memory for the optimized package is well below default’s TensorFlow working set in both regimes, reflecting a substantially smaller memory pool for the Triton/torch forward path than for the TensorFlow graph it replaces.

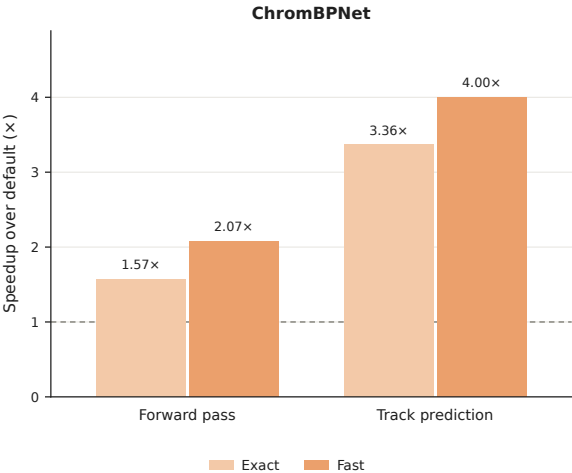


Figure S104. ChromBPNet: speed-up over default on H100. Bars show Exact and Fast speed-ups for the forward-pass regime, measured on the forward-call clock (time per item inside the model’s forward call), and for the track-prediction regime, measured on the whole-command wall clock; groups separate the two regimes. The dashed line marks default (1x).

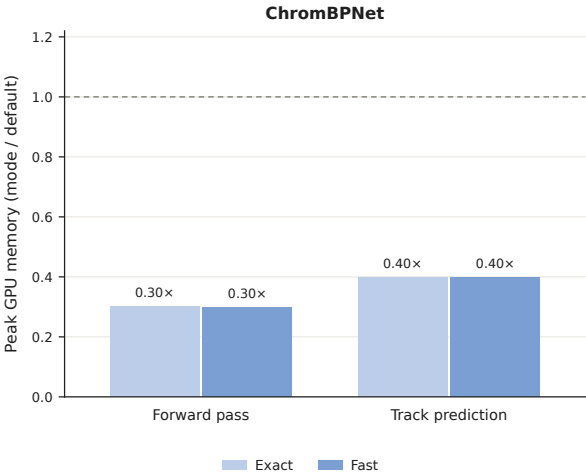


Figure S105. ChromBPNet: peak GPU memory relative to default. Bars show Exact and Fast peak GPU memory (NVML device high-water mark, from dedicated memory runs) as a ratio to default, for the forward-pass and track-prediction regimes. The dashed line marks default (1x).

S5.5 Enformer

Enformer predicts genomic coverage tracks from DNA sequence, taking one 196,608-bp window and producing $896 \text{ bins} \times (5,313 \text{ human} + 1,643 \text{ mouse})$ tracks [5]. The benchmarked release is the PyTorch port `enformer-pytorch` 0.8.12 [63], running EleutherAI/enformer-official-rough, the package’s documented port of DeepMind’s released weights, which, as the package loads it, accepts exactly this window length.

Benchmark setup. The benchmark times Enformer’s forward pass on 88 distinct 196,608-bp hg38 windows from the Basenji test split, in float32, producing both output heads, at two batch sizes: one window per call and 24 windows per call. The batch is held equal on both configurations in every row rather than letting each configuration use its own largest fitting batch. Each job runs on one exclusive H100 80 GB GPU, alternates default and the optimized package within a run, applies at least 20 untimed warm-up calls at the row’s batch size, then times a steady-state window of at least 200 calls and at least 60 seconds. The reported value is the median over three independent jobs. The headline clock is the forward clock – wall time per window inside the model’s forward call in steady state, warm-up and compilation excluded – with a per-item wall-clock figure over the same passes recorded as context only. Peak GPU memory is the device high-water mark read via NVML (the GPU’s hardware monitoring interface) from outside the process over the whole pass, above idle baseline. In-process allocator peaks are also logged but are diagnostics only. For this model, default is the upstream call run at the same batch as the compared mode. At the saturating batch, default therefore uses a batch of 24 windows – the largest that completes on an 80 GB card – rather than base’s single-window call. This keeps the GPU full, so the comparison is not run against a starved configuration. The published package exposes no speed flag or compile path, so batch size is the only setting that can be tuned, and default is simply base run at a larger batch. At one window per call, base reaches only $0.85\times$ the per-window rate of the saturating-batch default, showing that batching alone changes the unmodified package’s own throughput.

Sampling settings. Default and every mode used the same sampling settings. Enformer is deterministic: each call runs one forward pass over a batch of 196,608-bp windows on the forward strand only (no reverse-complement or shift augmentation) and returns both output heads (5,313 human and 1,643 mouse tracks). Batches hold 1 window per call or 24 windows per call, the largest batch that fits on the GPU. The single checkpoint is EleutherAI/enformer-official-rough, run through `enformer-pytorch` 0.8.12.

Modes. Only Exact is offered for Enformer; there is no Fast or Big mode. Exact returns the same bytes as default at the same batch size, verified bit-identical on 88 of 88 windows by comparing output-byte digests, together with two call-path checks covering the single-window and batched call forms. This equivalence holds without any tolerance band because the model is deterministic, with no sampling or seeds.

Optimizations. The optimized package fuses the trunk’s operations and removes per-call overhead while keeping upstream’s float32 operations and their order:

- **Fused elementwise trunk.** The convolutional trunk’s many separate elementwise steps – bias adds, normalization, activation, residual adds, and attention pooling – are combined into a handful of custom GPU kernels, removing intermediate round trips through GPU memory. This fusion is reported as most of the gain.
- **Exact relative-position attention.** Each transformer layer’s positional-attention logits are computed only over the band that upstream’s relative shift keeps, with the bias add and softmax fused into one kernel launch. This cuts launch and memory-traffic overhead while keeping the same arithmetic.
- **Positional-basis cache.** The relative-position basis used by attention is computed once per window length and device and kept on the GPU, instead of being rebuilt and copied from host to device on every call.
- **CUDA-graph replay of the trunk.** The trunk’s sequence of GPU operations is recorded once per batch size and replayed as a single unit, cutting per-call launch overhead. This replay helps most at small batches and does not apply once the batch is too large for the captured graph to fit in free memory.

Results. The forward-call speed-up is $1.39\times$ at the saturating batch of 24 windows per call and $1.43\times$ at one window per call. At the 24-window batch, the optimized package runs without CUDA-graph replay, so this gain reflects the other three changes alone. Peak GPU memory is essentially unchanged, reaching $1.00\times$ default at the saturating batch, where both configurations’ caching pools fill the 80 GB card. Only this per-window speed-up was benchmarked, because the package provides a single forward call rather than a multi-stage task. Checkpoint loading and set-up are fixed per-process costs outside the timed window, so the gain is realized when many windows are scored in one process.

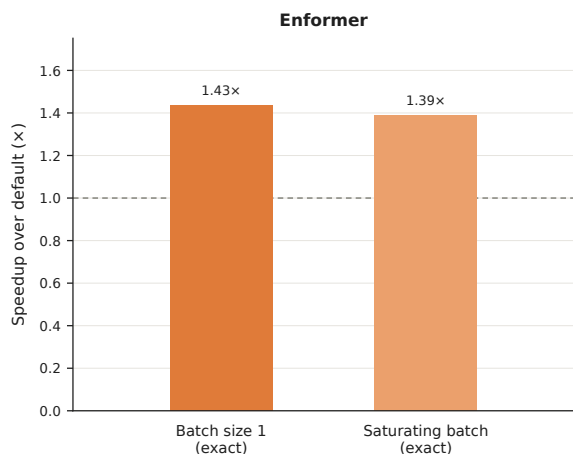


Figure S106. Enformer: forward-call speed-up over default on H100. Bars show the speed-up of Exact over default on the forward clock (time per window in steady state, warm-up excluded) for the batch-size-1 and saturating-batch (24 windows per call) regimes; the dashed line marks default (1 $\times$).

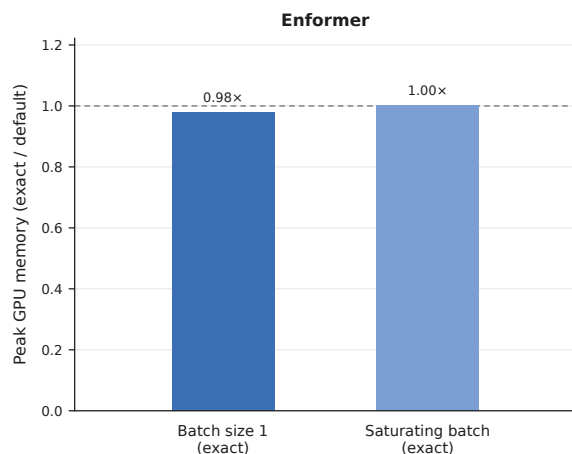


Figure S107. Enformer: peak GPU memory of Exact relative to default. Bars show the ratio of Exact's peak GPU memory to default's, measured as the NVML device high-water mark of the whole pass in dedicated memory runs, for the batch-size-1 and saturating-batch regimes; the dashed line marks default (1 $\times$).

S5.6 Enformer (original)

Enformer predicts genomic coverage tracks from DNA sequence. The version benchmarked here is the model as its authors serve it for inference: the TF-Hub SavedModel `deepmind/enformer/1`, documented in the `google-deepmind/deepmind-research` repository [5, 64].

Benchmark setup. Each input window is 393,216 bp, of which the network uses the central 196,608 bp (the PyTorch port’s window), and is one-hot encoded on the host. Each window yields 896-bin coverage predictions across 5,313 human and 1,643 mouse tracks. The benchmark uses 88 real genome windows (8 published test windows plus 40 human hg38 and 40 mouse mm10 test-split loci) at two batch sizes: one window per call, and a saturating batch of 14 windows per call. In both configurations, a 32-bit bias-add limit in the underlying framework sets the largest batch, not a memory limit. Each run warms up for at least 20 calls before timing a window of at least 40 calls and at least 60 seconds (at least 64 calls at batch size 1), and the reported speed-up is the median over three replicate containers on an H100 80 GB GPU. The headline clock is host wall time around the model’s prediction call plus a device synchronization, averaged per window. A companion whole-process wall time, which also includes process start-up and model loading, was recorded for orientation only and is not used for any reported speed-up. Peak GPU memory is the device-level (NVML) high-water mark above an idle baseline, measured in a dedicated run with the memory allocator set to grow only as needed. Without that setting, a naive reading would show the entire card as in use. For this model, base and default coincide. Default is the published call run through its documented interface, departing from a literal clone-and-run only in three environment settings shared by default and Exact: TensorFlow 2.17 in place of upstream’s pinned 2.5, which has no H100 build; the convolution autotuner disabled, so repeated runs stay bitwise consistent; and the allocator-growth setting used for the memory meter. None of the three settings is a performance tuning choice. These shared settings keep the comparison to the fastest correct way of running the unmodified model and pin both configurations to the same batch so that neither is slowed by a mismatch.

Sampling settings. Default and every mode used the same sampling settings. Enformer is deterministic: each call runs one forward pass over a batch of one-hot 393,216-bp windows on the forward strand only and returns both the human and the mouse heads. Batches hold 1 or 14 windows per call. The single model is the published `deepmind/enformer/1` release.

Modes. Only an Exact mode is benchmarked for this model; no Fast or Big variants exist. Exact reproduces, for each replaced group of operations, the same float32 arithmetic, operation order, and rounding as the kernels it replaces. It was verified to give bit-identical outputs to default across all 88 windows (plus additional 1- and 4-window call shapes) under deterministic settings, with the convolution autotuner disabled so that repeated runs agree byte-for-byte.

Optimizations. At the first prediction, the optimized package rewrites the model’s graph once, keeping its variables and call signature. It leaves the convolutions and large matrix multiplications on the original GPU-library kernels and fuses or precomputes the surrounding operations that dominate the graph’s cost; the main rewrites are:

- **Fused pooling matmul.** In each of the seven attention-pooling stages, computes the pooling logits with one large matrix multiply instead of a very large batch of two-row matrix products that run far below the GPU’s peak rate.
- **Fused pooling pipeline.** Merges each pooling stage’s transpose, softmax, weighting, and summation steps into a single kernel (one launched GPU operation) instead of five separate kernels over the largest activations.
- **Fused batch-norm plus activation.** Fuses the fourteen inference batch-normalization/GELU (a smooth activation function) chains, each of five elementwise kernels, into one kernel apiece.
- **Fused attention softmax.** Combines the relative-position shift, logit addition, and row softmax inside each of the eleven attention blocks into a single kernel.
- **Fused normalization.** Replaces the ten-step mean/variance normalization used at 22 sites in the network with one kernel per site, preserving the original summation order.
- **Precomputed constants.** Input-independent sub-graphs (relative-position keys, batch-normalization scale and shift) are evaluated once, removing about 1,000 small kernel launches per call, and a reshape size held on the host removes eleven host-device round trips per call.

Results. The speed-up is essentially flat across batch size because the input length is fixed and a single window already saturates the GPU: the optimized package is 4.96× faster than default at batch size 1 and 5.03× faster at the saturating batch of 14 windows per call, on the per-window forward clock. Peak device memory is unchanged (1.00× at both batch sizes), because the memory allocator grows only in large steps and both configurations land on the same step. Upstream ships no end-to-end tool, so no whole-task benchmark was run. Whole-process wall time improved less than the forward call in single orientation runs (not benchmark

statistics), because process start-up, model loading, and, for the optimized package, a one-time graph rewrite are fixed costs outside the timed window that do not amortize when only a few windows are processed.

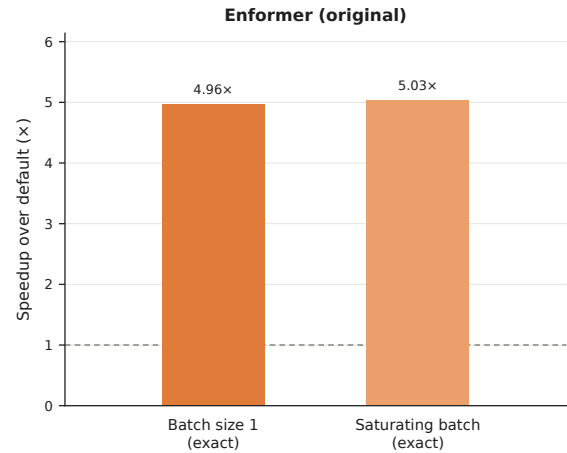


Figure S108. Enformer (original): forward-pass speed-up over default on H100. Bars show the per-window forward-call speed-up of the Exact mode over default at batch size 1 and at the saturating batch of 14 windows per call; the dashed line marks default (1×).

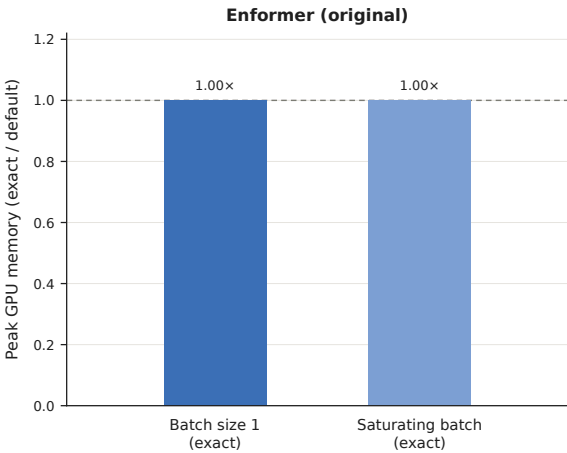


Figure S109. Enformer (original): peak GPU memory relative to default. Bars show the peak device-level (NVML) memory high-water mark of the Exact mode relative to default at batch size 1 and at the saturating batch of 14 windows per call, from dedicated memory runs; the dashed line marks default (1×).

S5.7 GPN-Star

GPN-Star is a masked DNA language model over whole-genome multi-species alignments: it predicts the masked center base of the human row in a 128-bp hg38 window aligned to 99 other vertebrates. We benchmarked upstream gpn 0.9.0 with the released checkpoint `songlab/gpn-star-hg38-v100-200m` [52].

Benchmark setup. The benchmarked task is a single masked-base forward pass over a 128-bp hg38 window with its 99 aligned vertebrate rows, drawn from a fixed set of 2,048 genome-wide random windows. Two batch-size regimes are used: a saturating-batch regime of 700 windows per call and a latency regime of one window per call. Default and the optimized mode use the same batch size in each regime. Each run performs at least three untimed warm-up calls at the timed shape, then a timed window of 200 calls (capped at 60 seconds, synchronized per call). Reported values are medians over three independent jobs on one exclusive NVIDIA H100 80 GB GPU. The headline clock times the bare forward call in steady state, excluding weight loading, warm-up, and any one-time setup. No end-to-end whole-command timing was measured. A separate command-line check, outside the timed measurements, confirmed byte-identical output on shipped examples. Peak GPU memory is the NVML device high-water mark above the card’s idle baseline, read from outside the process in a dedicated memory job with one pass per mode. Peaks seen during the timing runs themselves are diagnostic only. For this model, default adds two settings to base: upstream’s own documented `--tf32` flag, and the batch size being timed (700 or 1) rather than the library’s shipped batch size of 8. These two settings together give the fastest configuration we measured through upstream’s own interface (its `--torch-compile` and `--bf16-full-eval` options were not benchmarked). Default includes both so that neither the flag nor the batch size is credited to the optimized package.

Sampling settings. Default and every mode used the same sampling settings. GPN-Star is deterministic: each call runs one masked-language-model forward pass over a batch of 128-bp windows with their 100-species alignment, 700 windows per call in the saturating-batch regime and 1 in the latency regime. The checkpoint is `songlab/gpn-star-hg38-v100-200m`, run with upstream’s `--tf32` option.

Modes. Only one optimized mode, Exact, is offered for GPN-Star; no Fast or Big mode exists. Exact was verified to give bit-identical outputs to default at the same batch and settings, with no tolerance band. Identity was checked with a per-window SHA-256 digest over the 2,048 random and 768 stratified windows at the saturating batch, plus in-job identity checks at both timed batch sizes. This verification is scoped to runs with `--tf32` enabled.

Optimizations. The optimized package changes only how much redundant data reaches upstream’s own matrix multiplications, never the arithmetic itself.

- **De-duplicated clade key/value projection.** A clade row’s key and value depend only on its alignment column, and real alignment windows contain few distinct columns (1–2.5%). The optimized package therefore projects keys and values once per distinct column per layer and gathers them into place. Upstream instead projects every window, position, and clade row in its most costly step (de-duplication is off below four windows per call).
- **Column attention on pre-built operands.** Attention operands (the input tensors of the matrix multiplications) are built once, directly in the memory layout those multiplications need, avoiding the three full-size copy passes per layer that upstream performs before its matrix multiplications.
- **Fused gather-attention kernels.** Two custom GPU kernels compute attention scores and context directly from the de-duplicated rows through an index lookup, so the full-size key/value arrays are never materialized.
- **CUDA graph capture for small batches.** For small batch shapes, the whole forward pass is recorded once as a CUDA graph (a sequence of GPU operations captured once and replayed without repeated Python overhead) and replayed on later calls, replacing about 1,100 separate launches with one.
- **Constant caching and device-resident indexing.** Small per-shape constants and species indices are computed once and kept on the GPU rather than recomputed or copied from the CPU on every call, which also removes host synchronization that would otherwise block CUDA graph capture.

Results. The forward-pass speed-up over default is larger at one window per call than at the saturating batch (intermediate batch sizes were not measured). Exact reaches 3.47× default’s forward-call speed at the saturating batch of 700 windows per call and 5.90× at a single window per call, where the launch overhead removed by the CUDA graph matters proportionally more. Peak GPU memory moves in the opposite direction with batch size. De-duplicated attention lowers Exact’s memory use to 0.65× default’s at the saturating batch. Exact’s memory is slightly above default’s at batch size 1 because of the small CUDA graph memory pool. No end-to-end task was benchmarked for this model. End-to-end speed-ups on small variant sets should be expected to be much smaller

than these forward-call ratios, because start-up, model loading, and the package’s one-off set-up per input shape (graph capture and kernel compilation) are fixed per-process costs outside the timed window.

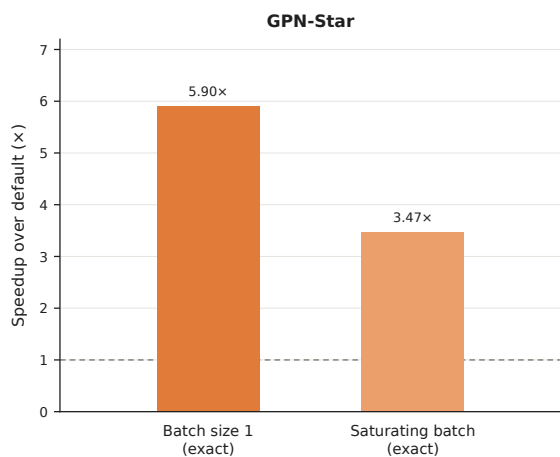


Figure S110. GPN-Star: forward-pass speed-up of Exact over default on H100. Bars show the speed-up on the per-call forward clock for the batch-size-1 (latency) and saturating-batch (700 windows per call) regimes; the dashed line marks default (1×).

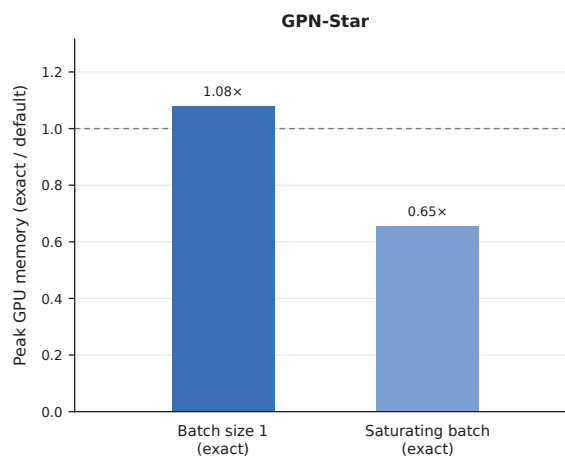


Figure S111. GPN-Star: peak GPU memory of Exact relative to default. Bars show the NVML device peak-memory ratio (Exact / default) from dedicated memory runs, for the batch-size-1 and saturating-batch (700 windows per call) regimes; the dashed line marks default (1×).

S6 Protein language models

These models learn from protein sequences alone and are used to score sequences, compute embeddings or generate new sequences. For ESM C and Profluent-E1, the headline condition is a saturating batch, which is the fairest comparison of throughput, with batch size 1 shown alongside; ProGen2 is benchmarked on its shipped scoring and generation tasks.

S6.1 ESM C

ESM C (ESM Cambrian) is a family of protein language models from EvolutionaryScale, released with 300M, 600M, and 6B parameters, that computes per-residue logits and embeddings from a protein sequence in a single forward pass [24]. We benchmarked the unmodified `esm` package (3.4.0, at the state just before its release tag) from Biohub’s repository [57], with Biohub’s `transformers` fork (4.57.6) that it pinned, and the ESMC-300M, ESMC-600M, and ESMC-6B weights from Hugging Face.

Benchmark setup. Each timed call is a forward pass over 512-residue sequences, at batch size 1 and at a saturating batch, the largest power-of-two batch that default completes on the GPU (512, 256, and 64 sequences for the 300M, 600M, and 6B models). Default and Exact are both run at this batch. Each configuration runs 200 timed calls after an untimed warm-up, in its own process on one H100 80 GB GPU (at the saturating batch, Exact is timed over 50 calls at default’s batch). The reported speed-up is the median over five independent jobs. The headline clock is the forward call itself. Peak memory is the NVML device peak above idle in dedicated batch-size-1 runs and the PyTorch allocator peak at the saturating batch, so memory ratios should be compared only within a batch regime. For this model, default is the package’s documented `ESMC.from_pretrained` interface in `bfloat16` with the optional accelerated kernels installed (flash attention and Transformer Engine, which the package lists among its `accel` dependencies). Base, a plain installation without them, falls back to slower kernels with slightly different numerics and was not measured. At the saturating batch both configurations use the interface’s native batching.

Sampling settings. Default and every mode used the same sampling settings. ESM C is deterministic and runs one forward pass per call, with no recycling or sampling. The 300M, 600M, and 6B checkpoints are separate models, not an ensemble. The batch-size-1 regime embeds one sequence per call; the saturating-batch regime embeds 512, 256, and 64 sequences per call for the 300M, 600M, and 6B models, the largest batches default completes.

Modes. ESM C has a single optimized mode, Exact. Its logits and embeddings were checked bit for bit against default across sequence lengths, at batch size 1 and in batched calls, for all three model sizes, and all checks passed.

Optimizations. The optimized package rewrites the forward pass around the same underlying kernels and adds a faster loading path, without editing the upstream code.

- **Fewer CPU–GPU synchronizations.** Metadata for variable-length attention is computed once per forward call instead of once per layer, removing pauses in which the GPU waits for the CPU. These pauses dominate single-sequence calls.
- **Fused kernels for small per-layer operations.** Custom kernels combine the residual addition, the residual–LayerNorm–QKV-projection step, and the query/key LayerNorm with the rotary position embedding, reproducing the library’s rounding order exactly.
- **No redundant copies.** Float32 copies of LayerNorm weights are made once at load time, normalized queries and keys are written directly into the fused QKV buffer, and hidden states are kept only when the caller requests them.
- **Padding-free execution at batch size 1.** For a single sequence, removing and restoring padding become views of the same memory rather than copies.
- **Faster loading.** The checkpoint is read once and converted directly on the GPU with the library’s own rounding, skipping a throw-away random initialization. This shortens start-up, most visibly for the 6B model, but does not affect the steady-state timings.

Results. For all three model sizes, Exact gains most at batch size 1, where the removed synchronizations and small kernel launches dominate the forward call. At the saturating batch, large matrix multiplications that are identical in both configurations dominate, and the gain is smaller. At 512 residues, Exact is 1.23–1.31× faster than default at the saturating batch and 2.98–3.09× faster at batch size 1. For ESM C 6B at batch size 1, the gain is largest at 512 residues and narrows for longer sequences: 2.94× at 128 residues, 3.09× at 512, 2.62× at 1,024, and 1.46× at 2,046. Exact lowers the device peak at batch size 1 (0.64–0.70× default) and the allocator peak at the saturating batch (0.78–0.79× default) for all three model sizes.

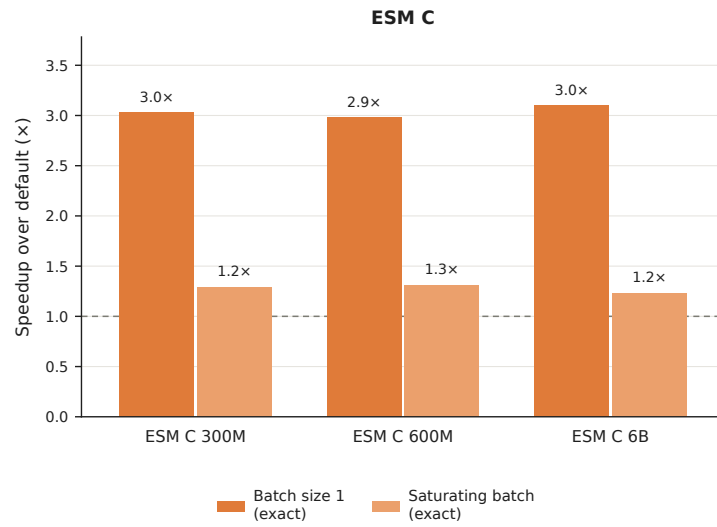


Figure S112. ESM C: forward-call speed-up of Exact over default. Bars show the forward-call clock speed-up (default time divided by Exact time) for each model size at two regimes: batch size 1 (one sequence per call) and saturating batch (both configurations run at default's largest completing batch for that model size). Bar labels are truncated to one decimal (p. 16); the dashed line marks parity with default (1×).

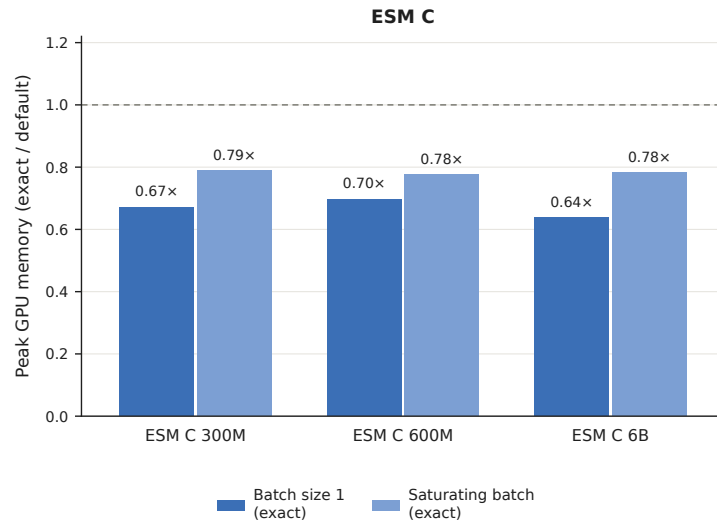


Figure S113. ESM C: peak GPU memory of Exact relative to default. Bars show the ratio of Exact's peak memory to default's at batch size 1 (NVML device peak above idle, dedicated runs) and at saturating batch (in-process torch allocator peak at the shared batch); the two regimes use different meters and are not directly comparable to one another.

S6.2 Profluent-E1

Profluent-E1 is a masked protein language model from Profluent, released in three parameter sizes (150M, 300M, 600M) [53]. Profluent-E1 supports both single-sequence scoring and retrieval-augmented inference, in which a query is conditioned on cached homologous context sequences drawn from its multiple-sequence alignment.

Benchmark setup. Three regimes were benchmarked on one exclusive NVIDIA H100 80 GB GPU. The batch-size-1 regime uses 64 natural proteins cropped to residues 1–1,024, one protein per call. The saturating-batch regime uses the same proteins at the largest batch the default configuration completes on the card (768/512/256 sequences for the 150M/300M/600M sizes). The retrieval-augmented regime applies masked-marginal scoring to the human P53 DMS parent (393 residues), conditioned on homologous ProteinGym context, with 16 masked query positions per call. The 15 context sets have 7,168, 14,336, or 21,504 tokens at five sequence-similarity ceilings from 1.0 to 0.5. Each configuration ran at least 20 untimed warm-up calls followed by a timed window of the longer of 8 calls and 60 seconds (of 256 calls and 60 seconds for batch-size-1 rows; the 600M saturating-batch Exact window was only 8 calls, about 7.5 seconds). Each row reports the median over 2 independent jobs. The benchmark clock is time spent inside the model’s forward/prediction call, measured in steady state with warm-up and compilation excluded. Peak GPU memory is the device-level peak above idle for a fresh process at batch size 1, the in-process allocator peak (`torch.cuda.max_memory_allocated`) at the shared saturating batch, and the device peak captured during the timed retrieval-augmented run. For this model, default and base are the same unmodified upstream package, run through its documented interface in its fastest shipped configuration, and they coincide in the batch-size-1 and retrieval-augmented regimes. In the saturating-batch regime they differ only in batch size: the tool as shipped scores with a budget of 65,536 tokens per call (63 sequences of 1,028 tokens), whereas default uses the largest batch that completes (see Sampling settings). No separate base measurement was made, because batch size is the only difference. The saturating-batch regime holds default and the optimized package to the same batch so that gains reflect genuinely GPU-saturated conditions rather than a comparison against an under-loaded default.

Sampling settings. Default and every mode used the same sampling settings. Scores are masked-marginal log-likelihoods from the model’s standard prediction interface, on sequences cropped to 1,024 residues. The batch-size-1 regime scores one sequence per call; the saturating-batch regime scores 768, 512, and 256 sequences per call for the 150M, 300M, and 600M models, in place of upstream’s default budget of 65,536 tokens per batch (`--max-batch-tokens`); the retrieval-augmented regime scores 16 masked positions per call against one homologous context set.

Modes. Only Exact mode is shipped for Profluent-E1; no reduced-precision or approximate mode is offered. Exact’s logits and last-layer embeddings were verified bit-identical to default under the release’s deterministic settings: at 1,024 residues across all three model sizes, for the retrieval-augmented call at 600M, and for a code-path suite at 600M. Four recorded checks on H100 (one per model size and the code-path suite) cover these settings, and all four pass.

Optimizations. The optimized package keeps every kernel upstream already uses and removes the surrounding overhead. It applies the changes when the scorer is constructed, without editing upstream files.

- **Direct unpadding flash-attention calls.** The flash-attention kernel is called directly on flat batch views with sequence offsets, removing the per-layer un-padding/re-padding operations and the CPU–GPU synchronizations they cause. This change is the largest single contribution.
- **Reused global-attention block mask.** An index-only attention mask for global layers is built once per batch shape instead of being rebuilt on the host every forward call.
- **Single dynamic-shape flex-attention compile.** One compiled kernel configuration serves every sequence length, so shorter inputs neither trigger recompilation nor switch code path.
- **Fused elementwise Triton kernels.** Four small Triton kernels (which combine several elementwise operations into one launch) fuse clamp+rotary embedding, gating, residual-add+normalization, and the embedding add. The kernels reproduce the library’s bfloat16 rounding order exactly.
- **Retrieval-augmented context packing.** Cached context keys/values in retrieval-augmented calls are viewed and packed once per forward pass instead of being copied and gathered on every layer. This packing accounts for most of the gain in the retrieval-augmented regime.

Results. Exact is 1.56–1.62× faster per protein than default at the saturating batch, where attention and matrix-multiply kernels that are identical in both configurations dominate. Its speed-up is largest at batch size 1: 3.02–3.60× across the three sizes (3.41× for the 150M model). Per-protein latency at this batch size is dominated by per-layer host work and small kernel launches that the optimized package removes. Exact’s speed-up in the retrieval-augmented regime is 2.51–3.26× per masked position, largest for the

150M model. Exact’s peak device memory is 0.71–0.81× default at batch size 1 and 0.53–0.78× default in the retrieval-augmented run, while its allocator peak at the shared saturating batch is 0.99–1.15× default.

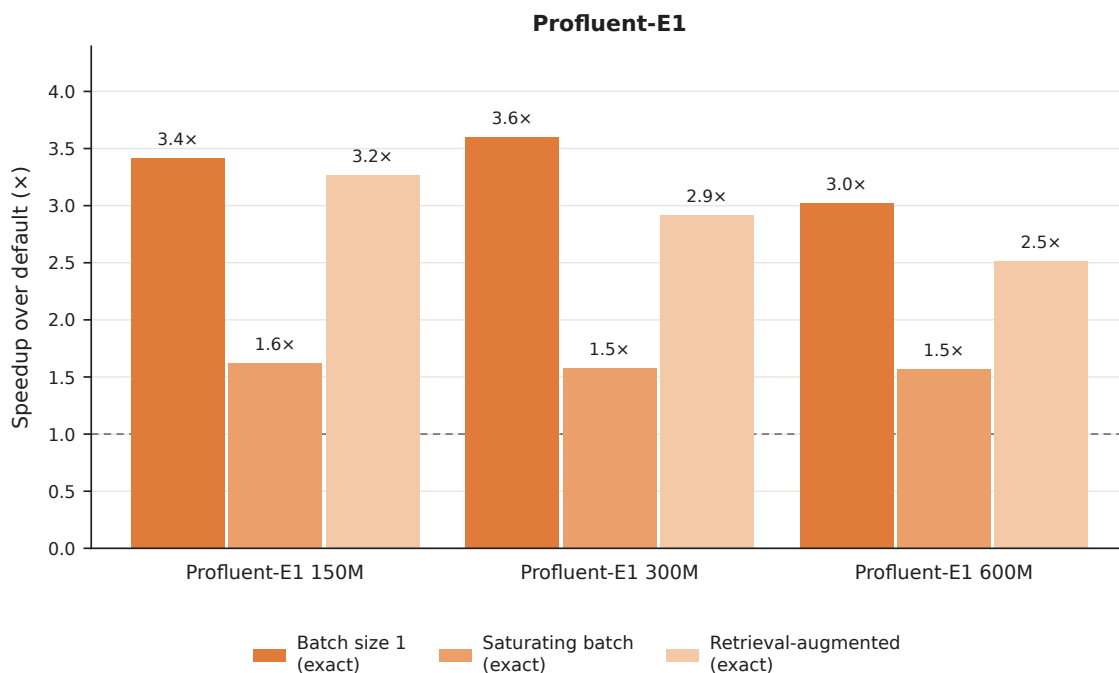


Figure S114. Profluent-E1: forward-call speed-up of Exact over default on H100. Bars show the speed-up on the forward/prediction clock for each model size (150M, 300M, 600M) in each of three regimes – batch size 1, saturating batch, and retrieval-augmented scoring; bar labels are truncated to one decimal (p. 16); the dashed line marks default (1×).

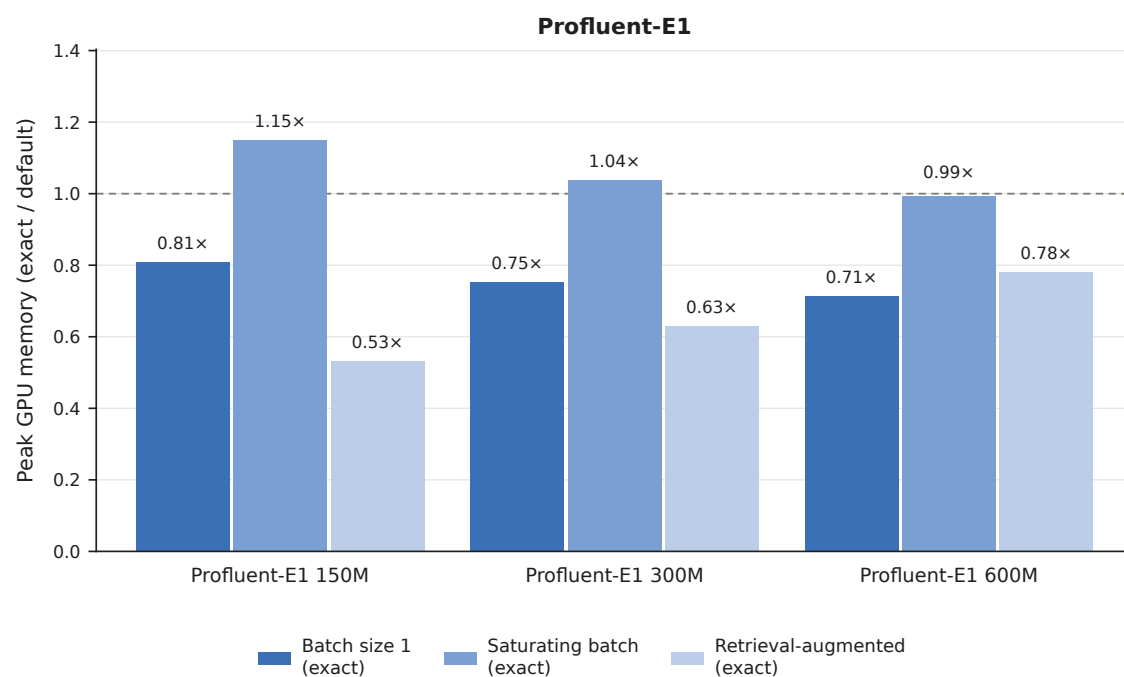


Figure S115. Profluent-E1: peak GPU memory of Exact relative to default on H100. Bars show the ratio of Exact's peak memory to default's for each model size in each regime: device peak at batch size 1, torch allocator peak at the shared saturating batch, and device peak during the timed retrieval-augmented run; the dashed line marks default (1x).

S6.3 ProGen2-xlarge

ProGen2 is Salesforce Research’s autoregressive protein language model family, released as the `salesforce/progen` repository together with its two shipped command-line tools for sequence generation and log-likelihood scoring [23]. All measurements benchmark ProGen2-xlarge (6.4 billion parameters, float16 weights, 1,024-token context), the largest released checkpoint, pinned at a fixed upstream commit. Upstream’s requirements pin a PyTorch 1.9 and CUDA 11.1 build that has no kernels for the H100, so every configuration ran on PyTorch 2.8.0 (CUDA 12.8, Python 3.9), with upstream’s transformers (4.16.2) and tokenizers (0.10.3) versions unchanged; no model code was modified.

Benchmark setup. Two tasks mirror the model’s shipped tools: sequence generation from a 32-residue prefix at five target lengths (101–501 tokens), 80 samples per call in both configurations, and log-likelihood scoring of natural proteins cropped to 100–500 residues, one sequence per call. Each input size is measured as the median over three independent jobs, each preceded by at least 20 untimed warm-up calls before the timed window. All runs used one exclusive NVIDIA H100 80GB GPU. For scoring, the headline clock is the time per model forward pass, computed from the steady-state time to score one sequence (warm-up and compilation excluded) and the number of forward passes per sequence (four in the upstream script, two in the optimized package); the time per scored sequence is reported alongside it. For generation, the clock is the time of the whole generation call, which produces 80 samples. Peak GPU memory is the device high-water mark read via NVML from outside the process over the whole pass, above the card’s idle baseline. For generation, the default figure reflects the caching allocator’s high-water mark rather than the live tensor footprint, because the unmodified package’s key/value cache is regrown by concatenation at every decoding step. For this model, default runs the upstream scripts’ own sampling and log-likelihood code, settings unchanged, in one process that loads the model once, at the same batch size in both configurations. Base, the scripts invoked as shipped, also reloads the model (and, for sampling, runs a sanity check) at every invocation, outside the steady-state clock, and was not measured separately. Generation uses a declared batch of 80 samples per call (the tool’s own `--num-samples` setting, whose shipped default is 1) in both configurations. Default’s fastest sample count was not searched.

Sampling settings. Default and every mode used the same sampling settings. Generation uses nucleus sampling with temperature 0.2 and top- p 0.95, drawing 80 samples per call (`--num-samples 80`; upstream default 1) from one 32-residue prefix, with a maximum length of $L + 1$ tokens for target length L (101–501 tokens; upstream default 256). Scoring evaluates one sequence per call. The model is ProGen2-xlarge (6.4B parameters) with fp16 weights.

Modes. Only Exact is shipped for ProGen2; no Fast or Big mode exists. Exact’s generated sequences and log-likelihood values were verified bit-identical to default across all measured generation and scoring units, run under deterministic settings (fixed seed, deterministic cuDNN) so that the same random stream and reductions are reproduced exactly.

Optimizations. The optimized package leaves the two command-line entry points and their flags untouched and instead changes how the forward pass and generation loop execute underneath them.

- **Single-forward scoring.** The unmodified scoring tool runs a separate forward pass (one full evaluation of the network) for each reduction and reading direction. The optimized package derives both reductions from a single forward pass’s output, halving the number of forward passes needed per scored sequence.
- **Fused per-layer kernels.** The query/key/value split with the rotary position embedding (a way of encoding token position inside attention) becomes one fused kernel using position tables kept on the GPU, and the MLP activation, the scaling and masking around attention, the residual additions, and the layer normalizations each become one fused kernel. This cuts launch overhead on the single-sequence scoring forward pass, where tokenization, host–device copies, and result assembly also run on a host thread alongside the next forward pass.
- **Preallocated key/value cache slots.** During generation, the cache that stores previously computed attention keys and values is written into preallocated per-batch-size buffers instead of being regrown by a new allocation and copy at every decoding step, avoiding memory churn that grows with sequence length.
- **Single-read checkpoint loading.** The checkpoint is memory-mapped and read once into the model without first allocating and discarding randomly initialized weights, shortening load time (outside the steady-state timing window).

Results. Per model forward pass, Exact scores sequences 1.94–2.16 $\times$ faster than default across the tested lengths (geometric mean 2.01 $\times$), from the fused per-layer kernels and the host work run alongside the next forward pass. Because the optimized package also runs two forward passes per sequence instead of four, each scored sequence is 3.89–4.32 $\times$ faster than with the upstream scoring script. Peak memory is essentially unchanged (0.97–0.99 $\times$), since each forward pass has the same shape. Generation speed-up instead tends to rise with sequence length, from 1.25 $\times$ at the shortest length to 1.67 $\times$ and 1.64 $\times$ at the two longest (1.43 $\times$ and 1.39 $\times$).

in between), consistent with the avoided per-step cache copies growing as the cache grows. Generation peak memory for Exact is 0.36–0.55× default’s in the dedicated memory runs shown in the figure and 0.66–0.77× during the longer timed runs. Default’s figure stays near the card’s capacity because its cache is regrown at every decoding step, so it reflects the allocator’s high-water mark rather than the live-tensor footprint.

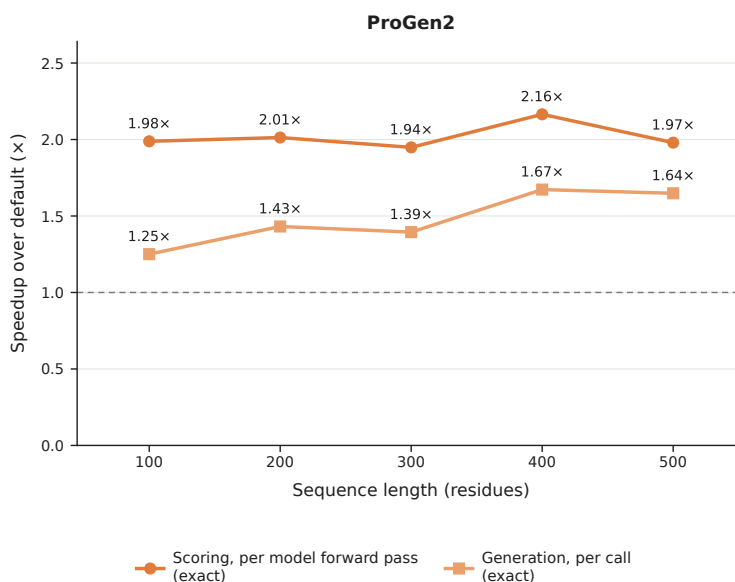


Figure S116. ProGen2-xlarge: speed-up of Exact over default. Lines show the speed-up (default divided by Exact) against sequence length for the scoring task, per model forward pass (one sequence per call), and for the generation task, per generation call of 80 samples; the dashed line marks default (1×).

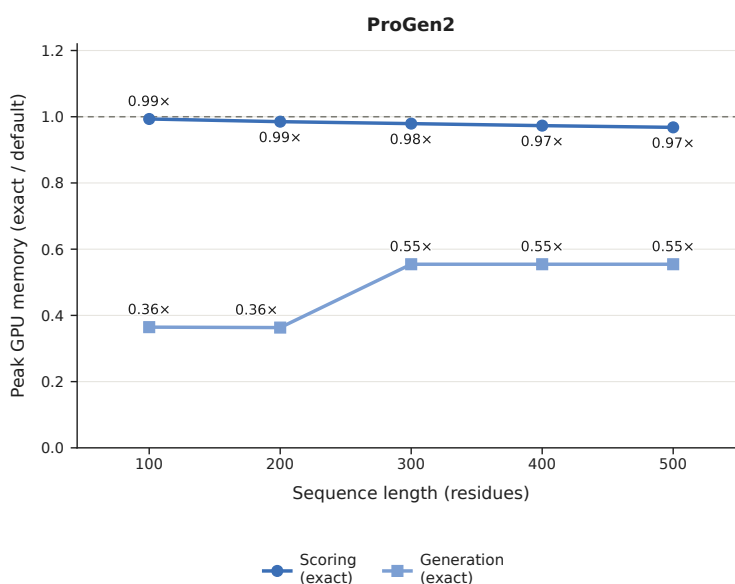


Figure S117. ProGen2-xlarge: peak GPU memory of Exact relative to default. Lines show the whole-pass device memory high-water mark (read via NVML), Exact divided by default, against sequence length, for the scoring and generation tasks measured in dedicated memory runs; the dashed line marks default (1×).

S7 Design quality under the optimized modes

Four benchmarks compare designs made under the optimized modes with designs made under default, with everything else held fixed: the same design requests (targets, lengths, and random seeds), the same sequence design step, and the same refolding and scoring. BinderBench and UnconditionalBench cover the hallucination and structure generation models; SCBench and NSRBench cover the inverse folding models. All generation, sequence design, and refolding ran on H100 GPUs, and ESMFold2 was always run in Exact mode. Differences from default are given with 95% paired percentile bootstrap confidence intervals (10,000 resamples of designs, backbones, or structures, each keeping its results under the two modes together, and leaving out the few scored under only one mode; for averages over targets or lengths, resampling is done within each target or length). Point estimates use every scored result, as the figures do. The benchmarks' own summaries are the distributions and means shown in the figures, without a statistical test. The figures were drawn by the same plotting package as the speed and memory figures.

S7.1 BinderBench: binder design

Setup. In BinderBench, 9 models designed binders against 5 protein targets: PD-L1, IL-7R α , TREM2, EGFR domain III, and TrkA (construct sizes 115, 193, 156, 205, and 101 residues), prepared as in a previously published set of experimentally confirmed binders. Each target has five hotspot residues: the target residues with the most heavy-atom contacts to the strongest binder of that target confirmed by both testing laboratories in that set (for IL-7R α , a contiguous surface patch; PD-L1 Y39 R96 M98 S100 Y106; IL-7R α L41 V42 K45 I66 N70; TREM2 M23 W26 N50 L51 L57; EGFR domain III Q75 H100 F103 I129 K156; TrkA H10 M15 H16 L52 H62, in construct numbering). They were passed to each model through its own hotspot option. ESMFold2-inverse hallucination takes no hotspots in its published protocol and was run without them. The models are the three hallucination models (ESMFold2-inverse hallucination, ColabDesign, and Mosaic) and the six structure generation models (RFdiffusion, RFdiffusion3, Genie 3, BoltzGen, PXDesign, and Proteina-Complexa), each run with its published binder design protocol and released weights under default, Fast, and, where the model has one, Big. Exact was not tested, because by design it does not change default's arithmetic (p. 16). Each model made 256 designs per target per mode, with binders of 80 residues (BoltzGen: 100, because at 80 residues its designs often had ubiquitin-like sequences), from a seed schedule shared by all modes of a model, so that every design request exists under every mode. The structure generation models output backbones, and each backbone received one sequence from SolubleMPNN [65] at temperature 0.1 with the target chain fixed (for Genie 3, whose backbones have C α atoms only, the C α -only ProteinMPNN model [4]); the hallucination models are scored on their own sequences. Every design was refolded five times with ESMFold2 [35] in Exact mode (binder and target as two chains without alignments or templates, 10 recycles, 68 sampling steps), and each refold was scored by ipSAE_{min} [31]: the interface ipSAE, computed from residue pairs across the interface with predicted aligned error below 10 Å, in both directions, taking the smaller value. The score of a design is the mean over its five refolds. Of the 115 combinations of model, target, and mode, 112 hold 256 scored designs and 3 hold 255.

Results. Figure S118 shows the distributions. Averaged over the five targets, the change in median ipSAE_{min} from default ranges from -0.064 (ColabDesign Fast; 95% confidence interval -0.124 to 0.002) to +0.034 (PXDesign Big; -0.005 to 0.065), and for none of the 14 combinations of model and optimized mode does the interval exclude zero. ColabDesign Fast, the largest change, had a lower median than default on 4 of the 5 targets, but none of its single-target intervals excludes zero. Of the 70 comparisons for single targets, 4 have intervals that exclude zero, about the number expected by chance at the 5% level (3.5): RFdiffusion Fast on PD-L1 (median 0.120 against 0.206 under default) and 3 comparisons on EGFR domain III in which the medians under both modes are below 0.02 (lower for RFdiffusion Fast and Proteina-Complexa Fast; higher for Mosaic Big; by at most 0.005).

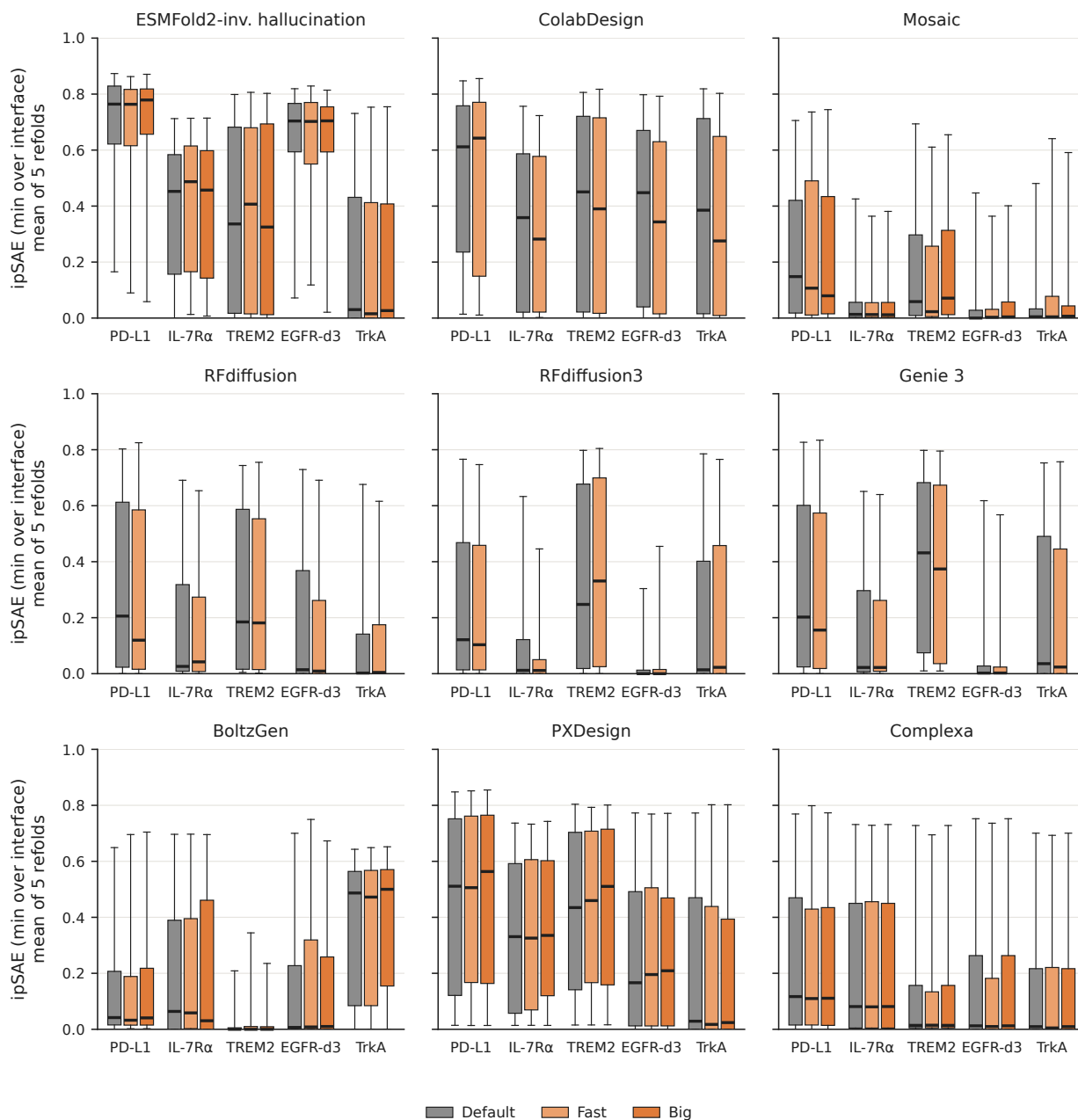


Figure S118. BinderBench: binder design quality under each mode. Each panel shows one model (Proteina-Complexa is labeled Complexa); x , target; one box per mode (default, Fast, and Big where the model has one). Each box summarizes 256 designs made from the same design requests (targets and seeds) under every mode (255 in three boxes); the score of a design is the mean ipSAE_{min} over its five ESMFold2 refolds in Exact mode. Heavy line, median; box, quartiles; whiskers, the most extreme designs within the 5th to 95th percentiles (Section S7.1).

S7.2 UnconditionalBench: unconditional backbone generation

Setup. The five structure generation models whose released versions can generate proteins without a target, PXDesign (default, Fast, Big), Genie 3 (default, Fast), RFDiffusion (default, Fast), RFDiffusion3 (default, Fast) and BoltzGen (default, Fast, Big), each generated 100 backbones per length (100, 200, 300, 400, and 500 residues; RFDiffusion3, 96) per mode, from a seed schedule shared by all modes of a model, with their own samplers at the released settings (PXDesign: 400 diffusion steps, $\eta = 2.5$). Proteina-Complexa's released model has no unconditional mode. Each backbone received one SolubleMPNN sequence at temperature 0.1 (Genie 3: the $C\alpha$ -only ProteinMPNN model), which was refolded as a single chain by ESMFold2 in Exact mode (one sample, fixed seed) and compared with the generated backbone by TM-score (scTM) [54] and by $C\alpha$ root-mean-square deviation after

superposition (scRMSD). A backbone is designable if its scTM is at least 0.5 and its scRMSD at most 2 Å; the designable fraction is this benchmark’s usual summary, and the figure shows mean scTM. Every combination has 100 refolded backbones except Genie 3 default at 100 residues, which has 95.

Results. Averaged over the five lengths, the change in mean scTM from default is at most 0.005 in magnitude for every model and optimized mode, and no interval excludes zero (Figure S119). Of the 35 comparisons at single lengths, 2 have intervals that exclude zero (1.75 expected by chance): BoltzGen Fast at 500 residues (change in mean scTM -0.049 ; 95% confidence interval -0.090 to -0.014) and PXDesign Fast at 100 residues (mean scTM 0.973 against 0.975 under default; the two modes designed the same sequence for 48 of 100 backbones, identified by identical refolds, and scTM is close to its maximum of 1, so the paired differences are small and the interval is narrow). The designable fraction changes by -6.0 to $+7.0$ percentage points across the 35 comparisons, and none of these changes has an interval that excludes zero, although for 4 of them, including the largest (BoltzGen Fast at 300 residues, $+7.0$ points), the interval ends exactly at zero. BoltzGen Big produced default’s sequence for 500 of 500 backbones.

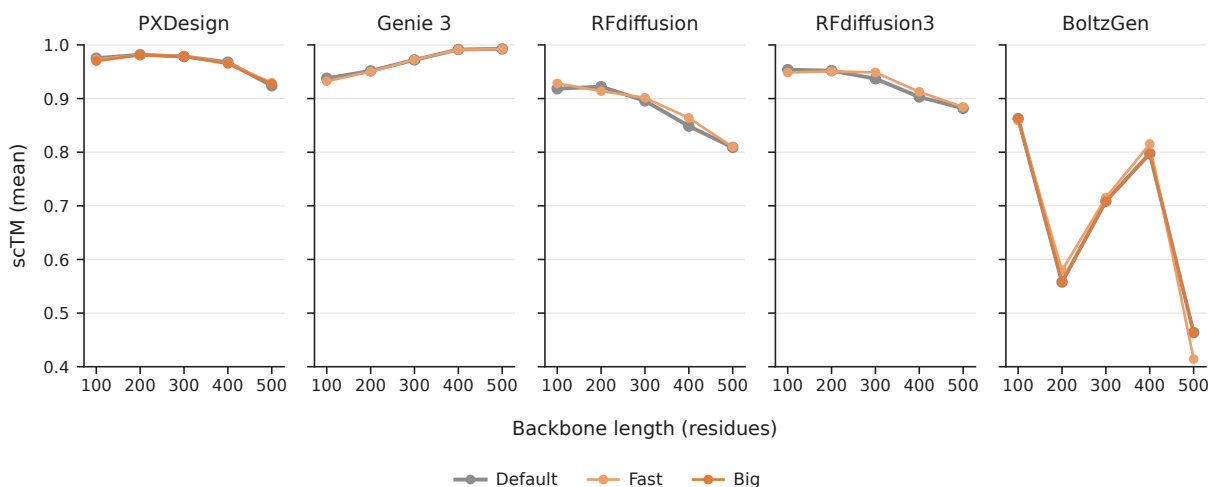


Figure S119. UnconditionalBench: quality of unconditionally generated backbones under each mode. Mean scTM of the ESMFold2 refold of one designed sequence per backbone against the generated backbone, per length and mode (100 backbones per point; RFdiffusion3, 96; Genie 3 default at 100 residues, 95), one panel per model. In this benchmark, BoltzGen Big reproduces default’s backbones and sequences, so its line coincides with default’s (Section S7.2).

S7.3 SCBench and NSRBench: inverse folding

Setup. Both benchmarks compare the optimized mode of each inverse folding model with default: ProteinMPNN Exact (vanilla weights v_48_020, temperature 0.1; its speed benchmark in Section S4.1 used the soluble weights), Caliby Fast (soluble checkpoint, its default sampler) and ESM-IF1 Fast (temperature 1.0, its default). In SCBench, each model designed one sequence under each mode for each of 100 backbones per length (100, 200, 300, 400, and 500 residues), generated once by RFdiffusion [3] and then held fixed. Each sequence was refolded as a single chain by ESMFold2 in Exact mode and compared with its input backbone by scTM, as in UnconditionalBench. NSRBench measures native sequence recovery, the fraction of residues whose designed amino acid matches the native one, on 200 protein structures released by the PDB between January 2023 and August 2026 (resolution 2.5 Å or better, selected as one representative per cluster at 30% sequence identity, although a few closely related chains remain, for example two structures of *E. coli* maltose-binding protein): 100 single chains and 100 chains from complexes, designed with their partner chains as fixed context. Each model designed four sequences per structure under each mode, and the recovery of a structure is the mean over its four sequences. Caliby’s structure reader treats a few complexes differently from the other models (it re-labels three, reads two as single chains and omits up to four unresolved residues in three), identically under both modes.

Results. ProteinMPNN Exact returned default’s sequence for 500 of 500 SCBench backbones and default’s four sequences for 200 of 200 NSRBench structures; Caliby Fast did so for 499 and 199, so their results coincide with default’s (Figures S120 and S121). ESM-IF1 Fast samples different sequences from default: it matched default’s sequence for none of the 500 SCBench backbones, and on 9 of the 200 NSRBench structures its mean recovery happened to equal default’s. Its change in mean scTM ranges from -0.011 to $+0.030$ across the lengths (default: 0.30 to 0.57), with no interval excluding zero, and is $+0.006$ averaged over the lengths (95% confidence interval -0.009 to 0.021). Its mean native sequence recovery changes by -0.07 percentage points (95% confidence interval -0.29 to 0.14; default 57.8%).

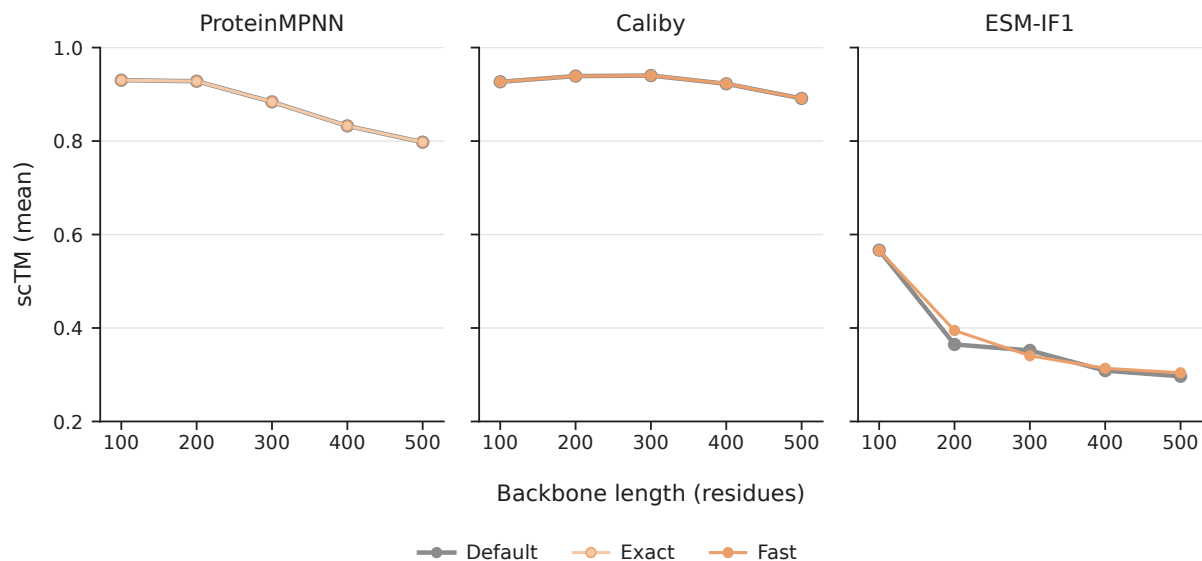


Figure S120. SCBench: self-consistency of inverse folding under each mode. Mean scTM of the ESMFold2 refold of one designed sequence per backbone against the input backbone, per length and mode (100 backbones per point; 99 for ESM-IF1 default at 400 residues), one panel per model. ProteinMPNN Exact returns default's sequences, and Caliby Fast does so for 499 of 500 backbones, so their lines coincide with default's (Section S7.3).

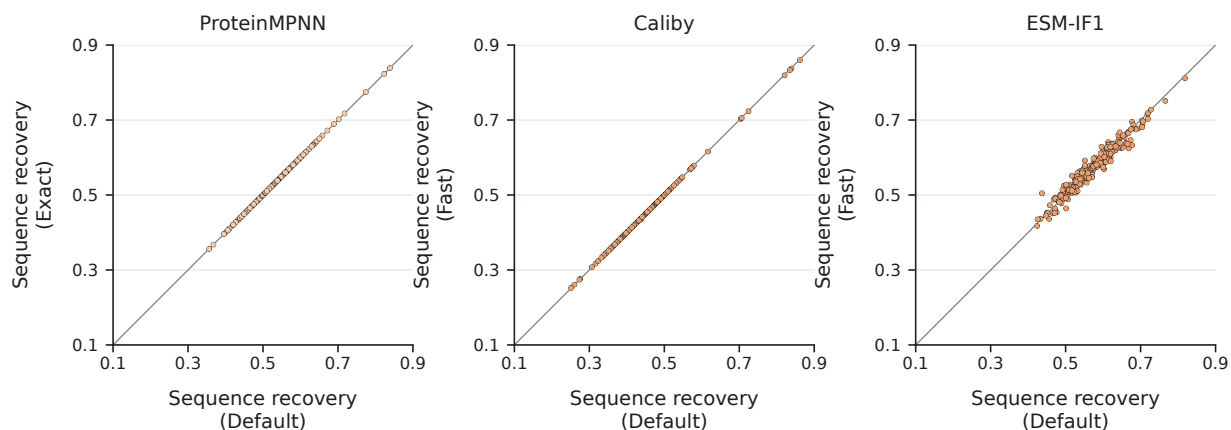


Figure S121. NSRBench: native sequence recovery under each mode. One point per structure (200 per model): x , recovery under default; y , recovery under the optimized mode, each the mean over four designed sequences; diagonal line, $y = x$. ProteinMPNN Exact reproduces default's sequences for all 200 structures and Caliby Fast for 199, so their points lie on or, for one Caliby structure, just below the diagonal (Section S7.3).

S8 Binder design: prompt and further results

The main-text section on binder design (p. 7) summarizes a benchmark in which a single Claude model, with one NVIDIA H200 GPU and 24 hours of wall time, designed miniprotein binders against each of 16 targets. This section gives the prompt of those runs and further results. Three Claude models (Mythos 5.1, Mythos 5, and Opus 5) ran five times against each target in each of two setups, with the accelerated models of this work and without them, for 480 runs in all. Scores are defined as in Figure 9: for each run and hour, the median or maximum *in silico* binding score of the designs the run had selected by that hour, averaged first over the runs of a target and then over targets. For the two GDF-8 targets (marked † in the figures), the score is adjusted for binding to the related protein GDF-11 as described there. The dashed or solid reference lines mark the corresponding scores of the final 30 designs of the earlier Mythos 5.1 campaigns (Methods).

S8.1 Prompt

Prompt design. We call the prompt the brief. It asks for 30 designs of 50 to 120 residues, a five-column table (sequence, design method, target region, the model’s own confidence, and a file with the designed complex), and a short report. It asks for the model’s own *de novo* designs and tells it not to submit known binders of the target or close variants of them. Because the task is to make a stated score as high as possible, the brief gives that score exactly, including its constants. It also describes the target, the machine, the input files, and the clock. Its task line asks for a structure-based design-and-filter approach but does not specify stages, filters, thresholds, or which tools to use, and the brief leaves the choice of structure, region, and epitope to the model. Its other instructions on how to work concern the clock: use the whole budget, run long jobs in the background, and keep the two output files current. Section S8.3 gives the brief in full. With 1,071 to 1,359 words per target, it is much shorter than the prompts of the earlier campaigns, which had about 16,000 words each [7].

Model inputs. Each run was a single instance of one Claude model, at its maximum reasoning effort, with two tools (a shell and a file editor) and no sub-agents or human input. For each target, the brief was the only task text the model received, and it was the same for all three models and both setups. Briefs for different targets differ only in their target-specific passages, and the two GDF-8 briefs also add selectivity against GDF-11. The input files were a description of the target, its UniProt entry, precomputed multiple-sequence alignments, the fixed target chains against which the designs are scored, the start time of the clock, up to 40 Protein Data Bank entries containing the target (leaving out entries with a designed or engineered binder or a single-domain antibody bound to the target), and a folder of 46 papers and method documents on binder design methods, structure predictors, and interface metrics, largely the same as in the earlier campaigns, together with text copies of 10 related web pages, mostly blog posts. The model could also open a reference sheet that describes each installed tool and how to run it. In the setup with the accelerated models, this sheet opens with a paragraph stating that the tools are faster builds of the same models, with the same weights and interfaces, and giving approximate speed-ups measured on H100 GPUs; it also gives the accelerated tools’ folder paths and which build of ESMFold2 is used. The machine had no network access, so the model could use only these files, the installed tools, and what it already knew.

During a run. Each run had a budget of 24 hours on a clock that counted wall time, including the model’s thinking. The clock paused whenever the model service kept a request waiting, for example during a service outage, so such waits did not use up the budget. From its second turn, the model saw the elapsed, budgeted, and remaining time at the start of every turn. If it replied without a tool call before the limit, it was told to continue, up to four times; a fifth such reply ended the run. Each command was stopped after 60 minutes, a reminder was sent about 75 minutes before the end, and the conversation was compacted automatically whenever it outgrew 180,000 tokens. At the limit, a closing message was sent and no further commands would run. Section S8.3 includes these messages.

Scores in this report. The scores in this report use only the ipSAE part of the benchmark’s score: for each design, the mean over the three predictors of its best-of-five-seed ipSAE, from a re-scoring of the saved selections, with half the corresponding score against GDF-11 subtracted for the two GDF-8 targets (Figure 9); each selection is summarized by the median and the maximum over its designs. The benchmark’s own score is instead the mean value over the 30 designs, in which ipSAE is blended with DockQ against the submitted designed complex, multiplied by the diversity factor, with zero for copies of known binders or of the target’s chains, and, for the two GDF-8 targets, with the selectivity term stated in their briefs. None of these enter the report’s scores.

S8.2 Further results

With and without the accelerated models. After 24 hours, all three models scored higher, averaged over the 16 targets, with the accelerated models than without them (Figure S122). With against without, the median design scored 0.781 against 0.753 (Mythos 5.1), 0.739 against 0.727 (Mythos 5), and 0.785 against 0.765 (Opus 5), and the best design scored 0.825 against 0.807, 0.813 against 0.805, and 0.833 against 0.822, in the same order. Without the accelerated models, the target-averaged median reached

that of the earlier campaigns after 22 hours for Mythos 5.1, not within 24 hours for Mythos 5, and after 17 hours for Opus 5; with them, it did so after 13 hours, not within 24 hours, and after 12 hours, in the same order. Target by target, the 24-hour median was higher with the accelerated models for 10 (Mythos 5.1), 11 (Mythos 5), and 12 (Opus 5) of the 16 targets. The largest difference was for Cas9, up to 0.260 for Mythos 5.1; outside Cas9, differences were at most 0.075 (EGFR, Mythos 5). Because the two setups also differed in their reference sheets and code versions (Section S8.1), these differences are not attributable to the accelerated models alone.

Targets. Figures S123 and S124 show the scores of each target over time. With the accelerated models, the run-averaged 24-hour median matched or exceeded that of the earlier campaign for 8 (Mythos 5.1), 3 (Mythos 5), and 9 (Opus 5) of the 16 targets, and the run-averaged best design did so for 8, 5, and 8, in the same order. The five runs of a model against a target differed in their 24-hour median by 0.046 (median over the 48 model–target combinations), and by up to 0.304 (Mythos 5, Cas9).

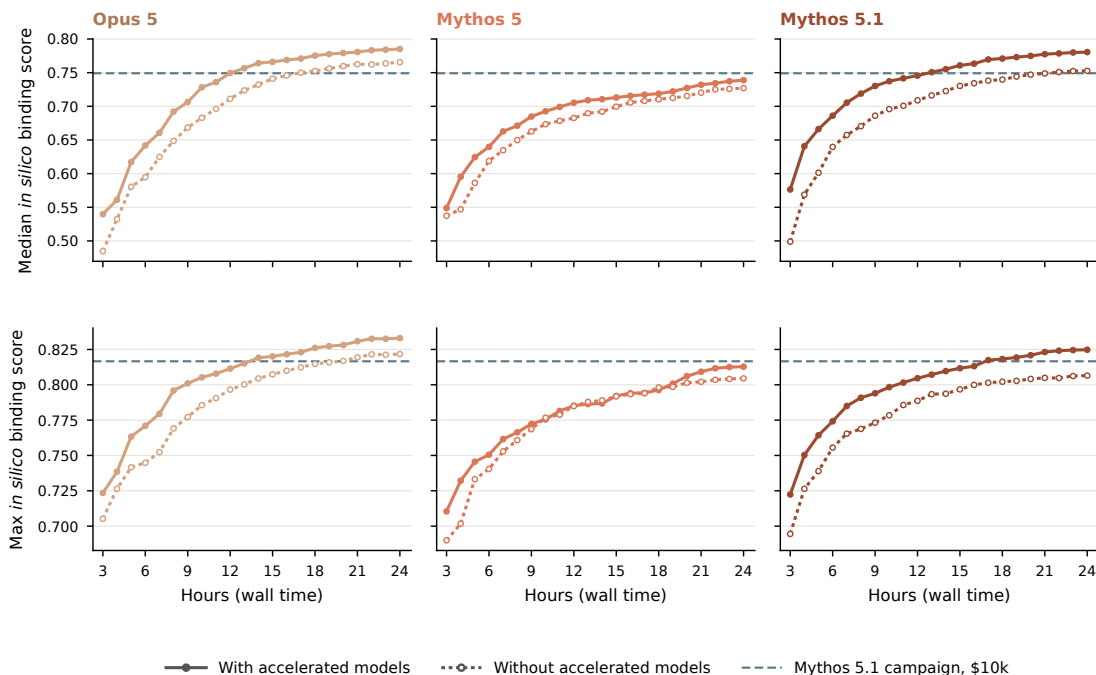


Figure S122. Binder design with and without the accelerated models. Median (top) and maximum (bottom) *in silico* binding score of the designs selected by each hour, averaged over the five runs of each target and then over the 16 targets, for runs with the accelerated models (solid) and without them (dotted). The dashed line marks the earlier Mythos 5.1 campaigns. Curves start at hour 3. The *in silico* binding score is ipSAE, a structure predictor’s confidence (0 to 1) in the predicted interface between binder and target, computed as in Figure 9: the lower of the complex’s two directional values (best of five seeds), averaged over ESMFold2, ESMFold2-Fast, and Protenix v2. †, half of the corresponding score against the related protein GDF-11 is subtracted.

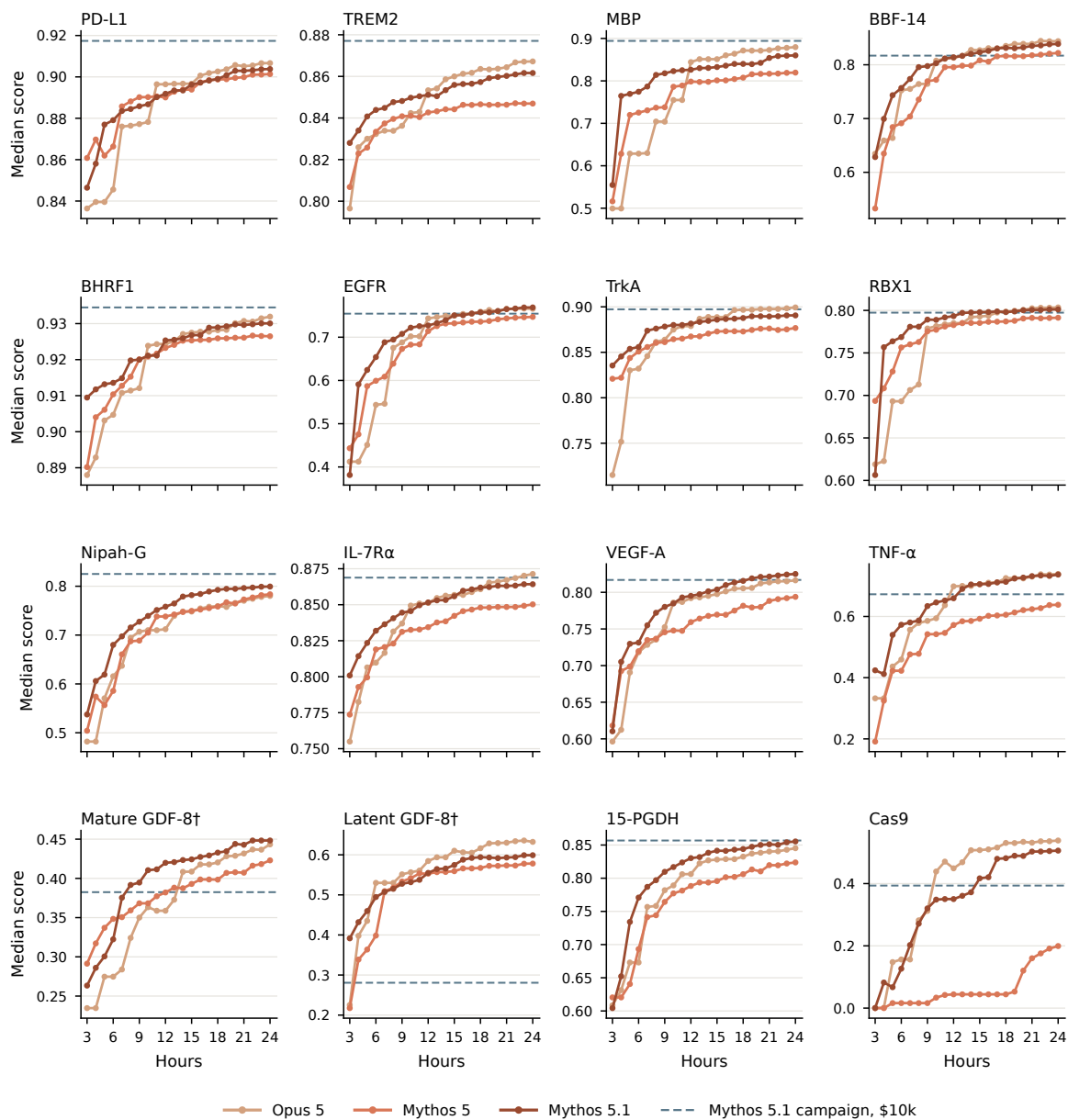


Figure S123. Median score of each target over time. Run-averaged median *in silico* binding score of the designs selected by each hour, with the accelerated models; the dashed line marks the earlier Mythos 5.1 campaign against the same target. Vertical axes differ between panels. The *in silico* binding score is ipSAE, a structure predictor's confidence (0 to 1) in the predicted interface between binder and target, computed as in Figure 9: the lower of the complex's two directional values (best of five seeds), averaged over ESMFold2, ESMFold2-Fast, and Protenix v2. †, half of the corresponding score against the related protein GDF-11 is subtracted.

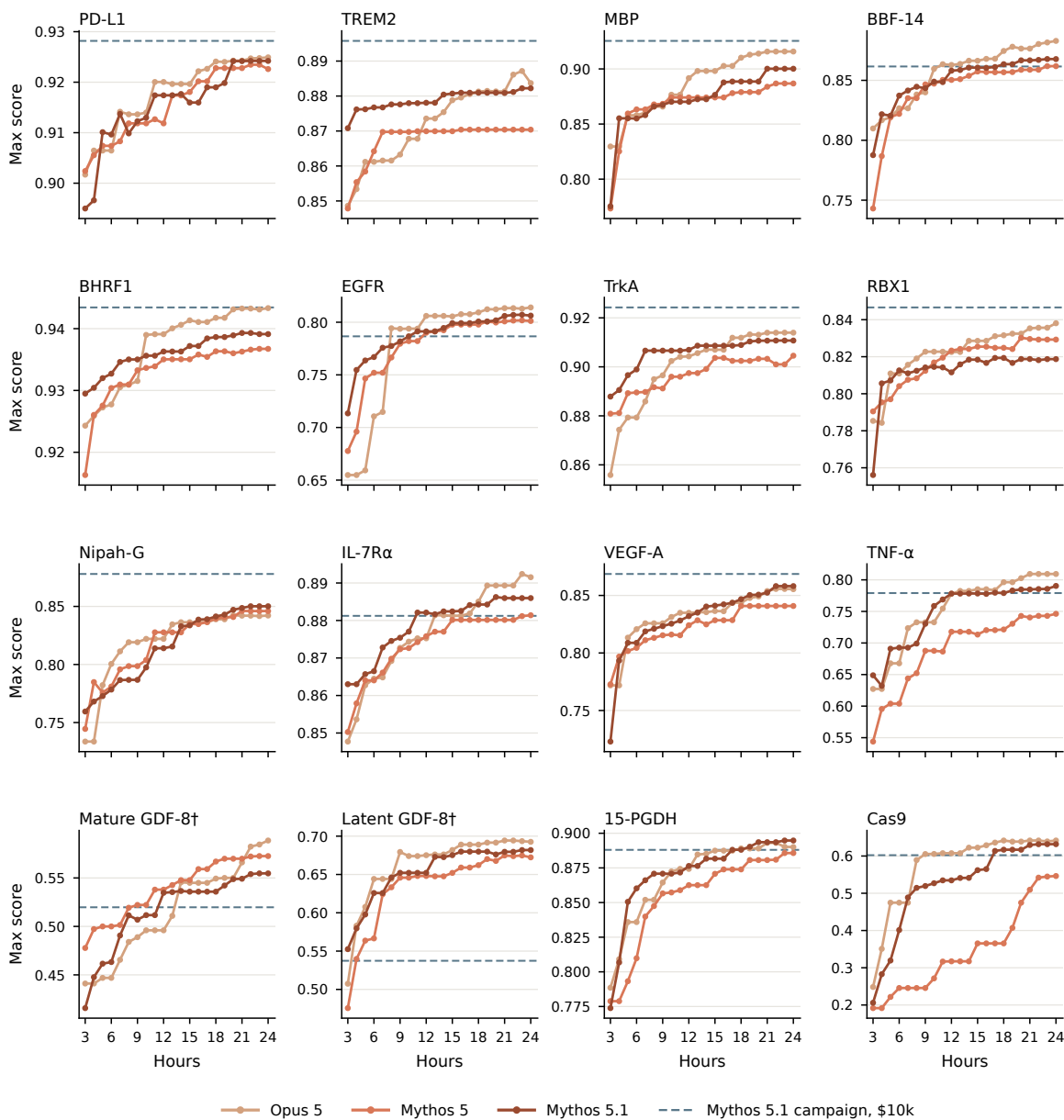


Figure S124. Best score of each target over time. As Figure S123, for the maximum score. The *in silico* binding score is ipSAE, a structure predictor’s confidence (0 to 1) in the predicted interface between binder and target, computed as in Figure 9: the lower of the complex’s two directional values (best of five seeds), averaged over ESMFold2, ESMFold2-Fast, and Protenix v2. †, half of the corresponding score against the related protein GDF-11 is subtracted.

S8.3 Prompt texts

The texts below are reproduced exactly as the model received them, except that long lines are wrapped. Labels in gray *italics* were added here.

Score stated in the briefs. Written as formulas, the Objective paragraph of each brief below asks for 30 designs that maximize

$$\text{score} = Q \times D,$$

the product of a quality score Q and a diversity score D .

Quality score. Q is the mean value of the 30 designs,

$$Q = \frac{1}{30} \sum_{k=1}^{30} v_k, \quad v_k = \frac{4 \overline{\text{ipSAE}}_k + \overline{\text{DockQ}}_k}{5},$$

where v_k is the value of design k and each bar is a mean over the three structure predictors (ESMFold2-Fast, ESMFold2, and Protenix v2). For each predictor, ipSAE_k is the lower of the two directional ipSAE values of the refolded complex, and DockQ_k compares the refolded complex with the designed complex submitted for design k (0 if none was readable). For the two GDF-8 targets, the value of a design also rewards selectivity against GDF-11:

$$v_k = \frac{4 \overline{\text{ipSAE}}_k + \overline{\text{DockQ}}_k + 4 \overline{\Delta}_k}{9}, \quad \Delta_k = \text{ipSAE}_k - \text{ipSAE}_k^{\text{GDF-11}},$$

where ipSAE_k is taken against the target form of GDF-8 and $\text{ipSAE}_k^{\text{GDF-11}}$ against the matching form of GDF-11.

Diversity score. D grows with the effective number of structurally distinct designs, N_{eff} , and reaches 1 at 15 such designs:

$$D = \min\left(1, \frac{N_{\text{eff}}}{15}\right), \quad N_{\text{eff}} = \frac{n^2}{\sum_{i=1}^n \sum_{j=1}^n S_{ij}},$$

where n is the number of delivered designs that could be refolded, $S_{ij} = \max(0, (\text{TM}_{ij} - 0.7)/0.3)$ with $S_{ii} = 1$, and TM_{ij} is the TM-score between the refolded binders of designs i and j (details in the brief). The scores in this report use only the ipSAE part of this rule (Section S8).

Brief for PD-L1. The brief as served for PD-L1 in all six combinations of model and setup. Its statement that the machine has “one NVIDIA H100-class GPU” is as served; the machines had an NVIDIA H200 GPU.

Task: Autonomously *de novo* design miniprotein binders against human programmed death-ligand 1 (PD-L1) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

Human PD-L1 (UniProt Q9NZQ7) is a monomeric type-I transmembrane immune-checkpoint ligand; the design target is its extracellular IgV domain.

For hotspot/epitope selection, prioritize biologically relevant interfaces and epitopes already explored in the design literature; novel epitopes are allowed when you have a differentiated hypothesis. Prefer functional epitopes where a miniprotein binder can plausibly achieve a measurable mechanism of action.

Objective

Submit 30 designs. Each design is refolded together with the target by three co-folding predictors - ESMFold2-Fast (single sequence), ESMFold2 (full model, with the target's multiple-sequence alignment) and Protenix v2 (with the target's multiple-sequence alignment) - each at several seeds, taking the predicted interface confidence, ipSAE (the lower of the two directions), from each, and each predictor's refolded complex (at its best-scoring seed) is compared with the designed complex you submit for that design (its complex_pdb file, see Deliverable; the binder chain is recognised by your design's sequence or, for a backbone-only model, by its length, and the target chain by the target's sequence) by DockQ, the standard 0-to-1 measure of how closely two models of the same interface agree (1 = the same pose); a design's value is $(4 \times \text{the mean of its three ipSAE values} + 1 \times \text{the mean of its three DockQ values}) / 5$, and a design submitted without a readable designed complex counts DockQ 0. The target chain in those refolds is fixed and the same for every design: UniProt Q9NZQ7 residues 18-132, 115 residues, written to /data/target/scoring_construct.fasta. The score is your designs' average value (sum over the 30 slots divided by 30) multiplied by a diversity factor that grows with the effective number of structurally distinct designs in your sheet - computed from the pairwise TM-scores of the refolded binder structures, where pairs with TM-score ≤ 0.7 count as fully distinct and near-identical backbones count as one - and reaches full credit at fifteen or more structurally distinct designs. Precisely: for delivered designs i, j with pairwise TM-score TM_{ij} between their binder structures taken from the ESMFold2 (full model, with the target's multiple-sequence alignment) refolded complexes, or ESMFold2-Fast's where the full model gave none (best seed, normalised by the shorter chain), $S_{ij} = \max(0, (\text{TM}_{ij} - 0.7)/0.3)$ with $S_{ii} = 1$; $N_{\text{eff}} = n^2 / \sum_{i,j} S_{ij}$ over your n delivered designs (those the grader could refold); $\text{score} = (\text{sum of values} / 30) \times \min(1, N_{\text{eff}} / 15)$. Higher is better for every one of the 30, and there is no threshold. Keep improving the set for as long as you have budget.

Environment

This machine has one NVIDIA H100-class GPU, no network access, and the design and analysis software pre-installed; /opt/ref/BINDER_TOOLS.md describes every tool and how to call it (binder backbone generators RFdiffusion3, PXDesign, FreeBindCraft, Proteina-Complexa and ESMFold2 gradient-guided binder design; SolubleMPNN/ProteinMPNN for sequence design; ESMFold2, AlphaFold2/AF-Multimer, Protenix, Boltz-2 and Chai-1 for structure prediction; TM-align, US-align, DSSP, MMseqs2, DockQ plus an ipSAE calculator and structure utilities). Staged inputs, read-only:

/data/target/target.json the target name and description above
/data/target/uniprot/ the UniProt entry for Q9NZQ7 (json, txt and fasta)
/data/target/msa/ precomputed multiple-sequence alignments (a3m) for the target protein, with a .meta.json saying what each query sequence is
/data/target/scoring_construct.fasta the fixed target chain your designs are scored against (see Objective)
/data/target/EPISODE_START when your wall-clock budget started: UTC time on line 1, epoch seconds on line 2 (see Budget)
/data/target/pdb/ 34 Protein Data Bank entries containing this protein, as deposited (mmCIF)
/data/literature/ papers on binder design methods, structure predictors and interface metrics (text), with an index

There is no network access; beyond these files and what you already know, nothing else is available. Choosing the structure(s), the region to design against and the epitope is part of the task.

Budget

You have 24 hours of wall-clock time for this episode. From your second turn on, each turn begins with an <elapsed_status> line giving the elapsed, budget and remaining seconds; that line is the authoritative clock. It runs through your thinking between commands as well as the commands themselves (only waits caused by the model service being busy are refunded, so the machine's own clock, whose reading at the start of the episode is recorded in /data/target/EPISODE_START, may run ahead of it). About 75 minutes before the end you will get a reminder. When the limit is reached no further commands run: whatever is already in /tmp/outputs at that moment is your submission, so keep /tmp/outputs/binders.csv and /tmp/outputs/report.md current as you go, especially after the reminder. Use the time: stop early only if you are confident that no further work would improve the 30 designs.

Each individual command is stopped after 60 minutes, so run long jobs in the background and poll them. Your working context is compacted automatically whenever the conversation outgrows its 180k-token window: earlier turns are replaced by a summary and you carry on from where you were, with everything on disk untouched - so keep /tmp/outputs/binders.csv, /tmp/outputs/report.md and any working notes on disk current enough to pick up from. Compaction is routine, not a failure, and context is not a budget (the wall clock above is the only one); do not cut the work short on its account.

Deliverable

1. /tmp/outputs/binders.csv - a CSV with a header row and exactly 30 data rows, columns:
binder_sequence,design_method,target_region,confidence,complex_pdb
- binder_sequence: 50-120 amino acids, only the 20 canonical residues (ACDEFGHIKLMNPQRSTVWY), all rows unique. These 30 sequences are your final designs. Designs must be your own de novo work; do not submit known binders of this target or close variants of them.
- design_method: short free text, how the design was generated.
- target_region: short free text, which part of the target it is meant to bind (structure/chain/residues as you define them).
- confidence: your own confidence that the design binds (free text or a number).
- complex_pdb: path to a PDB or mmCIF file under /tmp/outputs/ with the designed complex for this row - your design model of this binder bound to the target as it came out of your design process (the generator's or pipeline's own pose, not a separate re-prediction made for this file). Include one for every design.
 2. /tmp/outputs/report.md - a short report: the target region(s) you chose and why, the approach you took, how the final 30 were selected, and anything a reviewer should know.
- Write both files early and keep them updated; only their final state counts.

Target-specific text. For each of the other 15 targets, the passages that differ from the brief for PD-L1: the task line, the target description, the sentence of the Objective that names the target chains the designs are refolded against, and the lines of the file list that differ. For the two GDF-8 targets, the whole Objective paragraph is given, because it adds selectivity against GDF-11.

EGFR

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human epidermal growth factor receptor (EGFR) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description; its last part was added for this benchmark]

Human EGFR (UniProt P00533) is a monomeric receptor tyrosine kinase; the design target is its extracellular region.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: UniProt P00533 residues 25-645, 621 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P00533 (json, txt and fasta)
/data/target/pdb/ 33 Protein Data Bank entries containing this protein, as deposited (mmCIF)

TrkA

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human tropomyosin receptor kinase A (TrkA) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human TrkA (UniProt P04629) is a monomeric receptor tyrosine kinase; the design target is its extracellular Ig-like domain d5.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: UniProt P04629 residues 282-382, 101 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P04629 (json, txt and fasta)
/data/target/pdb/ 7 Protein Data Bank entries containing this protein, as deposited (mmCIF)

IL-7R α

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human interleukin-7 receptor subunit alpha (IL-7R α) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human IL-7R α (UniProt P16871) is a monomeric type-I cytokine receptor chain; the design target is its extracellular fibronectin type-III domain pair.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: 193 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P16871 (json, txt and fasta)
/data/target/pdb/ 6 Protein Data Bank entries containing this protein, as deposited (mmCIF)

TREM2

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human triggering receptor expressed on myeloid cells 2 (TREM2) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human TREM2 (UniProt Q9NZC2) is a monomeric immunoreceptor; the design target is its extracellular Ig-like V-type domain.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: UniProt Q9NZC2 residues 19-174, 156 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for Q9NZC2 (json, txt and fasta)
/data/target/pdb/ 9 Protein Data Bank entries containing this protein, as deposited (mmCIF)

MBP

[Task line]

Task: Autonomously *de novo* design miniprotein binders against *E. coli* maltose-binding protein (MBP) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

E. coli maltose-binding protein (MBP; UniProt P0AEX9) is a monomeric periplasmic solute-binding protein.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: UniProt P0AEX9 residues 27-396, 370 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P0AEX9 (json, txt and fasta)
/data/target/pdb/ 15 Protein Data Bank entries containing this protein, as deposited (mmCIF)

BHRF1

[Task line]

Task: Autonomously *de novo* design miniprotein binders against Epstein-Barr virus BHRF1 (BHRF1) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Epstein-Barr virus BHRF1 (UniProt P03182) is a monomeric viral Bcl-2 homolog.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: UniProt P03182 residues 2-158, 157 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P03182 (json, txt and fasta)
/data/target/pdb/ 7 Protein Data Bank entries containing this protein, as deposited (mmCIF)

BBF-14

[Task line]

Task: Autonomously *de novo* design miniprotein binders against the benchmark target BBF-14 using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

BBF-14 is a 110-residue *de novo* designed monomer used as a benchmark target.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: 110 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/pdb/ 1 Protein Data Bank entries containing this protein, as deposited (mmCIF)

Nipah-G

[Task line]

Task: Autonomously *de novo* design miniprotein binders against Nipah virus attachment glycoprotein G (Nipah-G) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description; its last part was added for this benchmark]

Nipah virus glycoprotein G (UniProt Q9IH62) is a receptor-binding attachment glycoprotein; the design target is its receptor-binding head domain.

[Objective, sentence naming the refolded target]

The target chain in those refolds is fixed and the same for every design: 416 residues, written to /data/target/scoring_construct.fasta.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for Q9IH62 (json, txt and fasta)
/data/target/pdb/ 13 Protein Data Bank entries containing this protein, as deposited (mmCIF)

RBX1

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human RING-box protein 1 (RBX1 / ROC1) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human RBX1 (UniProt P62877) is a monomeric RING-domain protein that coordinates three structural Zn²⁺; its native context is a heteromeric Cullin-RING ligase.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the chain written to /data/target/scoring_construct.fasta (UniProt P62877 residues 1-108, 108 residues) together with 3 zinc ions, folded with your design as one binder chain. ipSAE is taken between your binder and the target protein chain (the other molecules are part of the fold; ipSAE and DockQ are taken over protein chains only).

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P62877 (json, txt and fasta)
/data/target/pdb/ 15 Protein Data Bank entries containing this protein, as deposited (mmCIF)

TNF- α

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human tumor necrosis factor alpha (TNF α) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human TNF α (TNF; UniProt P01375) is a soluble homotrimer.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the complete assembly, three copies of the chain written to /data/target/scoring_construct.fasta (UniProt P01375 residues 77-233, 157 residues each), folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P01375 (json, txt and fasta)
/data/target/pdb/ 30 Protein Data Bank entries containing this protein, as deposited (mmCIF)

VEGF-A

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human vascular endothelial growth factor A (VEGF-A) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human VEGF-A (VEGFA; UniProt P15692) is a disulfide-linked homodimer.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the complete assembly, two copies of the chain written to /data/target/scoring_construct.fasta (UniProt P15692 residues 220-313, 94 residues each), folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P15692 (json, txt and fasta)
/data/target/pdb/ 19 Protein Data Bank entries containing this protein, as deposited (mmCIF)

15-PGDH

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human 15-hydroxyprostaglandin dehydrogenase (15-PGDH / HPGD) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

Human 15-PGDH (HPGD; UniProt P15428) is a C2 homodimer with a constitutive NAD⁺ cofactor.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the complete assembly, two copies of the chain written to /data/target/scoring_construct.fasta (UniProt P15428 residues 1-266, 266 residues each) together with 2 NAD molecules, folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together (the other molecules are part of the fold; ipSAE and DockQ are taken over protein chains only).

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for P15428 (json, txt and fasta)
/data/target/pdb/ 6 Protein Data Bank entries containing this protein, as deposited (mmCIF)

Cas9

[Task line]

Task: Autonomously *de novo* design miniprotein binders against *Streptococcus pyogenes* Cas9 (SpCas9) using a structure-based design-and-filter approach, targeting biologically relevant interfaces with two goals: 1) high-affinity binders zero-shot and 2) high overall hit rate.

[Target description]

S. pyogenes Cas9 (UniProt Q99ZW2) is a monomeric endonuclease whose functional unit is a 1:1 SpCas9:sgRNA ribonucleoprotein (RNP). The full RNP construct is to be used for both design and scoring.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the chain written to /data/target/scoring_construct.fasta (UniProt Q99ZW2 residues 1-1368, 1368 residues) together with a 98-nucleotide RNA chain, folded with your design as one binder chain. ipSAE is taken between your binder and the target protein chain (the other molecules are part of the fold; ipSAE and DockQ are taken over protein chains only).

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for Q99ZW2 (json, txt and fasta)
/data/target/pdb/ 40 Protein Data Bank entries containing this protein, as deposited (mmCIF)

Mature GDF-8

[Task line]

Task: Autonomously *de novo* design miniprotein binders against human mature Myostatin / GDF-8 that do NOT bind human mature GDF-11 using a structure-based design-and-filter approach, targeting biologically relevant interfaces with three goals: 1) selectivity, 2) high-affinity binders zero-shot, and 3) high overall hit rate.

[Target description; its last part was added for this benchmark]

Human Myostatin / GDF-8 (UniProt O14793) is a homodimer (C2, disulfide-linked). Human GDF-11, the counter-target (UniProt O95390), is also a homodimer (C2, disulfide-linked). Designs against human GDF-8 must be selective: they must NOT bind GDF-11, or must demonstrate a substantial binding preference for GDF-8 over GDF-11 (e.g. >100-1,000-fold higher affinity to GDF-8 than to GDF-11). The design target is the mature growth factor on its own: the C-terminal 109-residue domain of the precursor (UniProt residues 267-375) as the disulfide-linked homodimer, without the prodomain.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the complete assembly, two copies of the chain written to /data/target/scoring_construct.fasta (UniProt O14793 residues 267-375, 109 residues each), folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together.

[Environment, file-list lines that differ from PD-L1]

/data/target/uniprot/ the UniProt entry for O14793 (json, txt and fasta)
/data/target/pdb/ 8 Protein Data Bank entries containing this protein, as deposited (mmCIF)
/data/antitargets/gdf11_mature/ the counter-target mature GDF-11: antitarget.json (name, UniProt accession and the description above), scoring_construct.fasta (the fixed chain your designs are refolded against for selectivity, see Objective), pdb/ (5 Protein Data Bank entries containing it, as deposited, mmCIF)

[Objective, whole paragraph (adds selectivity against GDF-11)]

Objective

Submit 30 designs. Each design is refolded together with the target by three co-folding predictors - ESMFold2-Fast (single sequence), ESMFold2 (full model, with the target's multiple-sequence alignment) and Protenix v2 (with the target's multiple-sequence alignment) - each at several seeds, taking the predicted interface confidence, ipSAE (the lower of the two directions), from each, and each predictor's refolded complex (at its best-scoring seed) is compared with the designed complex you submit for that design (its complex_pdb file, see Deliverable; the binder chain is recognised by your design's sequence or, for a backbone-only model, by its length, and the target chains by the target's sequence) by DockQ, the standard 0-to-1 measure of how closely two models of the same interface agree (1 = the same pose). Each design is also refolded, in the same way (the same predictors, with that protein's own multiple-sequence alignment where one is used), with the counter-target mature GDF-11 (the complete assembly, two copies of UniProt 095390 residues 299-407, 109 residues, written to /data/antitargets/gdf11_mature/scoring_construct.fasta), and for each predictor the design's mature GDF-11 ipSAE is subtracted from its mature GDF-8 ipSAE: that predictor's selectivity difference, which can be negative. A design's value is $(4 \times \text{the mean of its three mature GDF-8 ipSAE values} + 1 \times \text{the mean of its three DockQ values} + 4 \times \text{the mean of its three selectivity differences}) / 9$, and a design submitted without a readable designed complex counts DockQ 0. The target in those refolds is fixed and the same for every design: the complete assembly, two copies of the chain written to /data/target/scoring_construct.fasta (UniProt 014793 residues 267-375, 109 residues each), folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together. The score is your designs' average value (sum over the 30 slots divided by 30) multiplied by a diversity factor that grows with the effective number of structurally distinct designs in your sheet - computed from the pairwise TM-scores of the refolded binder structures, where pairs with TM-score ≤ 0.7 count as fully distinct and near-identical backbones count as one - and reaches full credit at fifteen or more structurally distinct designs. Precisely: for delivered designs i, j with pairwise TM-score TM_{ij} between their binder structures taken from the ESMFold2 (full model, with the target's multiple-sequence alignment) refolded complexes, or ESMFold2-Fast's where the full model gave none (best seed, normalised by the shorter chain), $S_{ij} = \max(0, (TM_{ij} - 0.7)/0.3)$ with $S_{ii} = 1$; $N_{\text{eff}} = n^2 / \sum_{i,j} S_{ij}$ over your n delivered designs (those the grader could refold); $\text{score} = (\text{sum of values} / 30) \times \min(1, N_{\text{eff}} / 15)$. Also reported, and not part of the score: the share of your designs whose mature GDF-8 ipSAE is higher than their mature GDF-11 ipSAE on most of the three predictors. Higher is better for every one of the 30, and there is no threshold. Keep improving the set for as long as you have budget.

Latent GDF-8

[Task line]

Task: Autonomously **de novo** design miniprotein binders against human latent Myostatin / GDF-8 (the pro-form precursor, promyostatin) that do NOT bind human latent GDF-11 using a structure-based design-and-filter approach, targeting biologically relevant interfaces with three goals: 1) selectivity, 2) high-affinity binders zero-shot, and 3) high overall hit rate.

[Target description; its last part was added for this benchmark]

Human latent Myostatin / GDF-8 (UniProt 014793) is the unprocessed precursor assembled as a C2-symmetric 2:2 complex: two disulfide-linked growth-factor chains, each non-covalently caged by its N-terminal prodomain. Human latent GDF-11, the counter-target (UniProt 095390), has the same 2:2 precursor architecture. Designs against human GDF-8 must be selective: they must NOT bind GDF-11, or must demonstrate a substantial binding preference for GDF-8 over GDF-11 (e.g. >100-1,000-fold higher affinity to GDF-8 than to GDF-11). The design target is this latent precursor (prodomain and growth-factor domain in one chain, UniProt residues 24-375, two chains), not the mature growth-factor dimer on its own.

[Objective, sentence naming the refolded target]

The target in those refolds is fixed and the same for every design: the complete assembly, two copies of the chain written to /data/target/scoring_construct.fasta (UniProt 014793 residues 24-375, 352 residues each), folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together.

[Environment, file-list lines that differ from PD-L1]

```
/data/target/uniprot/ the UniProt entry for 014793 (json, txt and fasta)
/data/target/pdb/      3 Protein Data Bank entries containing this protein, as deposited (mmCIF)
/data/antitargets/gdf11_latent/ the counter-target latent GDF-11: antitarget.json (name, UniProt
accession and the description above), scoring_construct.fasta (the fixed chain your designs are refolded
against for selectivity, see Objective), pdb/ (5 Protein Data Bank entries containing it, as deposited,
mmCIF)
```

[Objective, whole paragraph (adds selectivity against GDF-11)]

Objective

Submit 30 designs. Each design is refolded together with the target by three co-folding predictors - ESMFold2-Fast (single sequence), ESMFold2 (full model, with the target's multiple-sequence alignment) and Protenix v2 (with the target's multiple-sequence alignment) - each at several seeds, taking the predicted interface confidence, ipSAE (the lower of the two directions), from each, and each predictor's refolded complex (at its best-scoring seed) is compared with the designed complex you submit for that design (its complex_pdb file, see Deliverable; the binder chain is recognised by your design's sequence or, for a backbone-only model, by its length, and the target chains by the target's sequence) by DockQ, the standard 0-to-1 measure of how closely two models of the same interface agree (1 = the same pose). Each design is also refolded, in the same way (the same predictors, with that protein's own multiple-sequence alignment where one is used), with the counter-target latent GDF-11 (the complete assembly, two copies of UniProt 095390 residues 25-407, 383 residues, written to /data/antitargets/gdf11_latent/scoring_construct.fasta), and for each predictor the design's latent GDF-11 ipSAE is subtracted from its latent GDF-8 ipSAE: that predictor's selectivity difference, which can be negative. A design's value is $(4 \times \text{the mean of its three latent GDF-8 ipSAE values} + 1 \times \text{the mean of its three DockQ values} + 4 \times \text{the mean of its three selectivity differences}) / 9$, and a design submitted without a readable designed complex counts DockQ 0. The target in those refolds is fixed and the same for every design: the complete assembly, two copies of the chain written to /data/target/scoring_construct.fasta (UniProt 014793 residues 24-375, 352 residues each), folded with your design as one binder chain. ipSAE is taken between your binder and all target protein chains together. The score is your designs' average value (sum over the 30 slots divided by 30) multiplied by a diversity factor that grows with the effective number of structurally distinct designs in your sheet - computed from the pairwise TM-scores of the refolded binder structures, where pairs with TM-score ≤ 0.7 count as fully distinct and near-identical backbones count as one - and reaches full credit at fifteen or more structurally distinct designs. Precisely: for delivered designs i, j with pairwise TM-score TM_{ij} between their binder structures taken from the ESMFold2 (full model, with the target's multiple-sequence alignment) refolded complexes, or ESMFold2-Fast's where the full model gave none (best seed, normalised by the shorter chain), $S_{ij} = \max(0, (TM_{ij} - 0.7)/0.3)$ with $S_{ii} = 1$; $N_{\text{eff}} = n^2 / \sum_j S_{ij}$ over your n delivered designs (those the grader could refold); $\text{score} = (\text{sum of values} / 30) \times \min(1, N_{\text{eff}} / 15)$. Also reported, and not part of the score: the share of your designs whose latent GDF-8 ipSAE is higher than their latent GDF-11 ipSAE on most of the three predictors. Higher is better for every one of the 30, and there is no threshold. Keep improving the set for as long as you have budget.

Messages during a run. The paragraph on context compaction in the system prompt, and the three messages the benchmark could send during a run. The message to continue was sent as a user message; the reminder and the closing message were sent as notifications.

In the system prompt: the paragraph on context compaction.

Each context window holds 180000 tokens. When the conversation outgrows it, it is compacted automatically: earlier turns are summarised and you continue the same task in a fresh window, with your files on disk intact. This is expected to happen during this task, possibly several times, and needs nothing from you beyond keeping your files on disk current enough to pick up from. Context is not a budget here and there is no countdown for it; do not stop or cut the work short on its account.

Sent once, about 75 minutes before the end.

No more than 75 minutes of wall-clock time remain (the <elapsed_status> line has the exact figure). Keep /tmp/outputs/binders.csv and /tmp/outputs/report.md current with your best designs from here on; when the limit is reached no further commands will run.

Sent at the limit.

Wall-clock time limit reached. No further commands will run; the files in /tmp/outputs as they are now are your submission. Reply with a brief closing note and no tool calls.

Sent if the model replies without a tool call before the limit.

The episode has not ended; it runs until the wall-clock limit (the `<elapsed_status>` line has the figure). If further work could improve the designs, continue. If you are finished, make sure `/tmp/outputs/binders.csv` and `/tmp/outputs/report.md` hold your final set and say so; the episode then ends at the time limit or after a few such replies.

S9 Kernel benchmark

In a Pairformer, triangle operations update the feature vector of each token pair (i, j) in the pair representation from the pairs that i and j each form with every token k . Triangle attention and triangle multiplication account for much of the runtime of Pairformer-based structure predictors, and their compute grows with the cube of the token count. Main-text Figure 2 compares FlashPairformer v1, our kernels for both operations, with the field-standard kernels [13] on one NVIDIA H100 80 GB GPU.

Timed operations. The pair width is the number of channels per token pair: 128 in panels A and B, and 256 in C and D. Triangle attention (panels A and C) is timed as the sum of one Pairformer block’s starting-node and ending-node calls, which attend along the pair representation’s rows and columns, respectively. Attention uses 4 heads of 32 channels at pair width 128 and 8 such heads at pair width 256. Triangle multiplication (panels B and D) is timed as the sum of the outgoing and incoming layers, which combine two rows or two columns, respectively. Each layer is timed in full: input LayerNorm, gated projections, triangle contraction, output LayerNorm, and gated output projection. At pair width 256, the layer’s hidden width equals the pair width. FlashPairformer v1 is timed with its fast kernels for both operations.

Inputs and hardware. All timings are forward passes at batch size 1 and sequence lengths of 256, 384, 512, 768, 1,024, 1,536, and 2,048 tokens. Inputs are unmasked, seeded random normal tensors, with weights at LayerNorm scale. Inputs and outputs are in bfloat16, with TensorFloat-32 (TF32) disabled.

Timing protocol. Only each kernel’s own work is timed, because inputs were converted to its expected memory layout and data type beforehand. Each call was captured as a CUDA graph after a warm-up of at least 10 calls and 100 ms of GPU time. Replaying a CUDA graph removes per-operation Python and kernel-launch overhead. The two kernels were then replayed in interleaved blocks of about 1 s over four rounds, alternating which kernel ran first. The first block of each kernel was discarded, and at least 20 replays of each call were kept. CUDA events bracketed each replay to time it on the GPU. Blocks logged as throttled were rejected. Each call’s time is the median of its kept replays.

Correctness. The outputs of both kernels matched a float64 reference to within bfloat16 rounding in the same runs, and repeated calls gave identical outputs.

Speed-up summary. Each plotted speed-up is the field-standard time divided by the FlashPairformer v1 time. Values above 1 mean that FlashPairformer v1 is faster. Each panel is summarized by the geometric mean and range of its seven speed-ups, truncated to two decimals (p. 16). FlashPairformer v1 was faster at all 28 plotted points (Table S1). Triangle attention ran 2.69× faster with FlashPairformer v1 at pair width 128 (geometric mean; range 1.74–3.92×) and 2.89× faster at pair width 256 (2.04–4.22×). Triangle multiplication ran 1.69× faster at pair width 128 (1.51–1.86×) and 3.20× faster at pair width 256 (2.82–3.46×).

Table S1. Speed-up of FlashPairformer v1 over the field-standard kernels on one H100 80 GB GPU. Each value is the field-standard time divided by the FlashPairformer v1 time, as plotted in panels A–D of main-text Figure 2. Values above 1 mean that FlashPairformer v1 is faster. Attention times are summed over one Pairformer block’s starting- and ending-node triangle attention calls, and multiplication times over the outgoing and incoming triangle multiplication layers. The last row is the geometric mean over the seven sequence lengths. Values are truncated to two decimals.

Sequence length (tokens)	Pair width 128		Pair width 256	
	A Attention	B Multiplication	C Attention	D Multiplication
256	1.74	1.86	2.04	3.14
384	2.22	1.79	2.32	3.46
512	2.48	1.76	2.51	3.46
768	2.77	1.72	2.80	3.38
1,024	2.88	1.66	3.02	3.19
1,536	3.43	1.59	4.22	3.03
2,048	3.92	1.51	3.94	2.82
Geometric mean	2.69	1.69	2.89	3.20