SUPER APRIEL: ONE CHECKPOINT, MANY SPEEDS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce SUPER APRIEL, a 15B-parameter supernet in which every decoder layer provides four trained mixer choices—Full Attention (FA), Sliding Window Attention (SWA), Kimi Delta Attention (KDA), and Gated DeltaNet (GDN). A *placement* selects one mixer per layer; placements can be switched between requests at serving time without reloading weights, enabling multiple speed presets from a single checkpoint. The shared checkpoint also enables speculative decoding without a separate draft model. The `all-FA` preset matches the Apriel 1.6 teacher performance; recommended hybrid presets span $2\times$ to $10.7\times$ decode throughput at 97% to 77% quality retention, with throughput advantages that compound at longer context lengths. With four mixer types across 48 layers, the configuration space is vast. A surrogate that predicts placement quality from the per-layer mixer assignment makes the speed–quality landscape tractable and identifies the best tradeoffs at each speed level. We investigate the convergence dynamics of the layer placement rankings. Rankings stabilize early at both scales, and while the ordering within the frontier placements may change, the frontier set itself converges over distillation. SUPER APRIEL is trained by stochastic distillation from a frozen Apriel 1.6 teacher, followed by SFT. We release the supernet weights, training & serving code and placement optimization toolkit.

1 INTRODUCTION

As language models scale to longer contexts and higher throughput requirements, full attention (FA) becomes an inference bottleneck. Each FA layer maintains a KV cache that grows linearly with sequence length; during autoregressive decoding, data movement dominates runtime as each step processes a short query against a long cached context (Kwon et al., 2023; Shah et al., 2024). *Hybrid* architectures address this by retaining FA in critical layers while using efficient alternatives elsewhere—sliding-window attention, linear recurrences like Mamba (Gu & Dao, 2024) or GDN (Yang et al., 2025; Kimi Team et al., 2025).

The central question is *which layers* should use which mixer. Prior work treats this as a one-time decision: placements are fixed either at design time via spacing heuristics (NVIDIA et al., 2025a; Zuo et al., 2025) or post-hoc via layer importance scoring (Ostapenko et al., 2025). Either way, the released model exposes a single operating point on the speed–quality curve.

A fixed placement forces deployment compromises: different workloads (long-context decode vs. short-prompt batch, peak vs. idle traffic) need different speed–quality tradeoffs, requiring runtime flexibility. SUPER APRIEL targets the goal of *flexible placement at serving time* from a single checkpoint. Derived from a 48-layer 15B transformer, every decoder layer exposes four trained token mixer options (FA, SWA, KDA, GDN) that can be selected at runtime. A *placement* selects one token mixer per layer; placements share most parameters (FFNs, embeddings, normalization) and differ only in mixer blocks.

This flexibility is useful only if placement can be chosen well. The $4^{48} \approx 10^{28}$ combinatorial space is intractable to search exhaustively. We address this with a *cost-constrained surrogate guided search* that decomposes placement quality into local interaction terms, enabling exact cost-constrained optimization via dynamic programming. The surrogate is fitted on a set of diverse evaluated placements and is iteratively updated to rank the full space, identifying Pareto-optimal placements at each cost target.

Contributions:

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

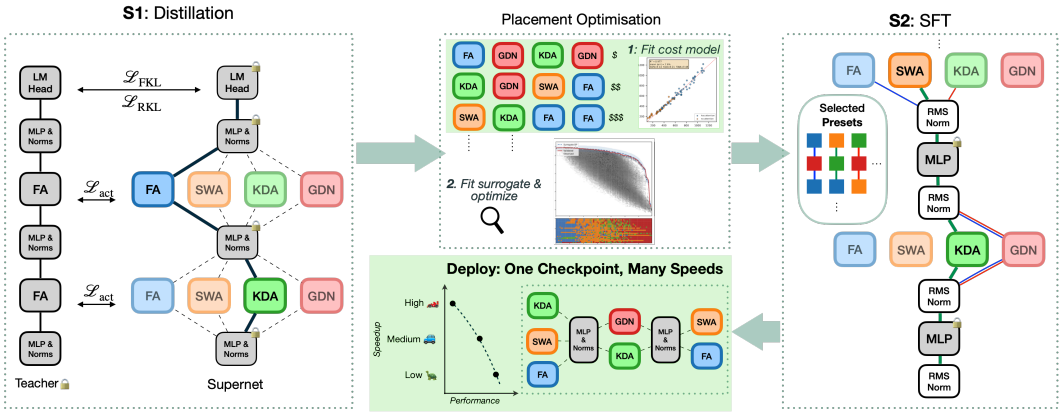


Figure 1: An overview of SUPER APRIEL training recipe. We leave out residual connections for simplicity. Training starts with `S1: Distillation` phase (Section 2.2), where at every step and every layer, a mixer is drawn uniformly from one of four mixer types. Distillation uses activation matching, forward KL, and reverse KL losses. All parameters (MLP, LayerNorm, etc) except mixers remain frozen throughout distillation. After distillation, we perform placement optimization (Section 2.3) and find a set of candidate placements for each target cost-tier. In the second training stage `S2: SFT`, we perform targeted training sampling exclusively from a set of candidate presets. One may also perform SFT on all placements and use global sampling of mixers in this phase. The final checkpoint supports multiple placements that are at the throughput-performance Pareto frontier according to the cost model used in the placement optimization phase.

- **Token-mixer supernet.** Base and instruct checkpoints with four trained mixer choices per layer, enabling flexible cost-quality navigation at deployment and speculative decoding with a mixer preset as draft model (Appendix F).
- **Training recipe.** Stochastic distillation from a frozen teacher followed by targeted SFT.
- **Placement optimization.** Cluster-expansion surrogate with exact cost-constrained DP, sweeping the Pareto frontier.
- **Open release.** Supernet weights, training code, vLLM extension, and placement toolkit.

2 METHOD

2.1 ARCHITECTURE

Teacher. We start from Apriel 1.6 (Madhusudhan et al., 2025) — an open weight 15B-parameter reasoning transformer with 48 layers. See details in Appendix B.

Mixer vocabulary. Each supernet layer contains four mixer options (each described in detail in Appendix B): Full Attention (FA) — standard grouped-query attention with full KV cache; Sliding Window Attention (SWA) — attention limited to a fixed window (4096 tokens), reducing KV cache to constant size; Gated DeltaNet (GDN) — state-space model with gated updates and delta rule (Yang et al., 2025); Kimi Delta Attention (KDA) — a GDN variant with channel-wise gating (Kimi Team et al., 2025).

A *placement* $\mathbf{x} \in \{1, 2, 3, 4\}^{48}$ assigns one mixer per layer. All placements share FFN, embedding, and normalization parameters; only mixer parameters differ. Retaining all four mixer options at every layer brings the full supernet to ∼25B parameters—a ∼67% increase over the ∼15B of any individual placement—although serving a single placement requires only its ∼15B active subset, with inactive mixers offloaded (Appendix G). The `all-FA` placement exactly recovers the teacher architecture. Each placement is also associated with a cost c_m, e.g. $c_{\text{FA}} = 1$, $c_{\text{SWA}} = 0.48$, $c_{\text{KDA}} = 0.21$, $c_{\text{GDN}} = 0.14$, that can be predicted via a fitted linear model as discussed in the next section. Hence maximal cost of a placement is equal to the number of layers L.

2.2 TRAINING

Training consists of two stages: distillation and supervised fine-tuning (SFT). The training data composition is described in Table 2. The training loss curves are in Fig. 14.

S1: Distillation. As visualized in Fig. 1, similarly to Gu et al. (2025), in the first training stage we train all mixer options simultaneously via stochastic sampling. At each forward pass, one mixer is sampled uniformly per layer. All non-mixer parameters (FFN, LayerNorm, embeddings) remain frozen from the teacher; only mixer parameters are trained.

The loss combines three terms:

$$\mathcal{L} = \lambda_{\text{act}}\mathcal{L}_{\text{act}} + \lambda_{\text{FKL}}\mathcal{L}_{\text{FKL}} + \lambda_{\text{RKL}}\mathcal{L}_{\text{RKL}} \tag{1}$$

where $\mathcal{L}_{\text{act}}$ is per-layer L2 activation matching between student and teacher mixer outputs, and $\mathcal{L}_{\text{FKL}}$, $\mathcal{L}_{\text{RKL}}$ are forward and reverse KL divergences on output logits. Forward KL encourages mode-covering (student covers all teacher modes), while reverse KL encourages mode-seeking (Agarwal et al., 2024). In the rest of the manuscript, the post-distillation checkpoint is referred to as `S1: Distil`.

S2: Supervised Fine-Tuning. After distillation, we run placement optimization (Section 2.3) to identify Pareto-optimal placements. `S2: SFT` then trains only these selected presets on reasoning-heavy data now sampling from the set of available presets per batch, which is contrast to `S1` where mixers are sampled per layer.

2.3 PLACEMENT OPTIMIZATION

Given a trained supernet, we seek placements that maximize quality subject to inference cost constraints. The search space ($4^{48} \approx 10^{28}$) is too large for enumeration. The overall algorithm consists of two stages: **exploration** and **refinement**.

Stage 1: Exploration. The goal of the exploration phase (Algorithm 1) is to build a diverse dataset $\mathcal{D} = \{(\mathbf{x}^{(k)}, S(\mathbf{x}^{(k)}))\}_{k=1}^{N}$ of placement–score pairs. Each placement is associated with a cost, and we perform cost-stratified sampling to cover the full cost range. A secondary goal is to determine which feature configuration (order and range) best describes the data.

In order to fit a surrogate model to placement quality scores, we decompose the score function using the cluster expansion from statistical physics (Sanchez et al., 1984; de Fontaine, 1994): layers correspond to lattice sites, mixer types to atomic species, and the supernet score to the energy. The expansion expresses S as a sum of n-body interactions:

$$S(\mathbf{x}) = \underbrace{\sum_{i=1}^{L} V_i(x_i)}_{\text{order 1}} + \underbrace{\sum_{i=1}^{L-1} V_{i,i+1}(x_i,\, x_{i+1})}_{\text{order 2}} + \cdots \tag{2}$$

where each potential decomposes over indicator features. For order 1, $V_i(x_i) = \sum_{m\in\mathcal{M}} V_i(m)\,\phi^{(1)}_{i,m}$ with $\phi^{(1)}_{i,m} = \mathbb{1}[x_i{=}m]$ ("is layer i set to mixer m?") and $V_i(m)$ a learned scalar. For order 2, $V_{i,i+1}(x_i, x_{i+1}) = \sum_{m,m'} V_{i,i+1}(m,m')\,\phi^{(2)}_{i,mm'}$ with $\phi^{(2)}_{i,mm'} = \mathbb{1}[x_i{=}m \wedge x_{i+1}{=}m']$. Collecting all indicators into a feature vector $\boldsymbol{\phi}(\mathbf{x})$ and all potentials into a weight vector $\mathbf{w}$, the model is linear: $S(\mathbf{x}) = \mathbf{w}^\top\boldsymbol{\phi}(\mathbf{x})$. To obtain $S(\mathbf{x})$, we run the supernet with placement $\mathbf{x}$ on a set of cheap proxy tasks and use the resulting metric as the score. By default we use average normalized log-likelihood of teacher-generated math traces under placement candidates (MATH-500, AIME24, AIME25) to estimate the score. As detailed below, we also used other tasks in some preliminary experiments (Section 3).

Two hyperparameters control the expansion's expressiveness, i.e. which features are included in $\boldsymbol{\phi}$: **Order**—the maximum number of layers in a single feature (the expansion above shows orders 1 and 2); **Range**—the maximum distance between interacting layers; range 1 restricts pairwise features to adjacent layers $(i, i{+}1)$, range 2 adds pairs up to two apart $(i, i{+}2)$, and so on.

Assuming that layer interactions decay geometrically with distance through the residual stream justifies truncation to low order (≤ 3) and short range (≤ 3). Feature counts at each expansion order are given in Appendix C.

Surrogate. We fit a Bayesian Ridge regression model (MacKay, 1992) on $\mathcal{D}$ to learn the coefficients $\mathbf{w}$. Because Eq. 2 decomposes S into potentials over individual layers and small contiguous groups, it has the structure of a Markov random field (MRF): each layer is a node, and the fitted potentials V_i, $V_{i,i+1}$ act as unary and pairwise MRF potentials. Finding the highest-scoring placement is then MAP inference on a chain, solvable exactly by *dynamic programming* (DP)—processing layers sequentially while tracking a bounded window of recent assignments as state (Viterbi, 1967). While a nonlinear surrogate could approximate the score landscape more flexibly, it would forfeit exact optimization with DP. Fig. 7 shows that both ρ and R^2 for the ridge surrogate rise sharply from pure allocation to order-1 features, with diminishing returns beyond.

Adaptive expansion selection. Higher-order features are more expressive but introduce exponentially more parameters (Appendix C, Table 5), risking overfitting with limited data. We therefore require a minimum 2:1 sample-to-feature ratio to admit a given (order, range) configuration, and among admitted configurations select the one that maximizes Bayesian model evidence (marginal likelihood), which naturally penalizes unnecessary complexity. As more placements are evaluated, higher-order configurations become admissible and are adopted when the evidence supports them. The selected order is also diagnostic: it reveals whether the supernet's quality landscape is governed by per-layer effects alone or by multi-layer interactions (Appendix C).

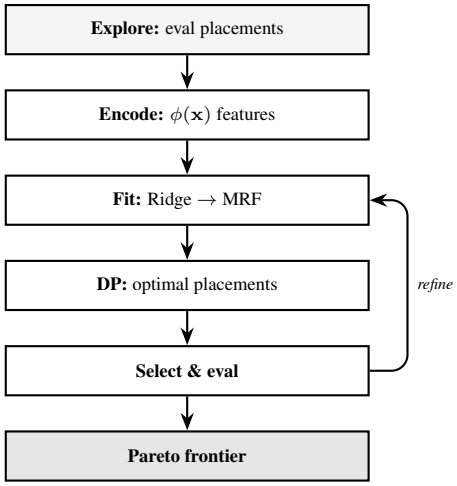


Figure 2: Surrogate-guided search. Features encode placements; Ridge regression yields MRF potentials; DP finds optima at all cost levels.

Cost model. Inference cost depends primarily on the mixer *allocation*—the count of each mixer type across layers—rather than which specific layers they occupy. We fit an additive per-layer cost model $c(\mathbf{x}) = \sum_{i=1}^{L} c_{x_i}$, where c_m is the per-layer cost of mixer m, calibrated from measured latencies (Appendix H). Because both the surrogate (Eq. 2) and the cost model are additive over layers, the cost constraint can be folded directly into the DP: a single forward pass finds the optimal placement at *every* cost level simultaneously (seconds on CPU).

Stage 2: Refinement. Given a set of cost targets, the surrogate is iteratively refined to better predict frontier placements at each target. At each round, a DP pass generates candidate placements per cost level from the current surrogate (Algorithm 2); candidates are then selected via a risk-bucket acquisition that splits the evaluation budget between a *safe* bucket (scoring by $\hat{\mu} - \beta\hat{\sigma}$, exploitation) and an *upside* bucket (scoring by $\hat{\mu} + \beta\hat{\sigma}$, exploration), where $\hat{\mu}$ and $\hat{\sigma}$ are the posterior predictive mean and standard deviation (details in Appendix C). The selected candidates are evaluated on the supernet to obtain $S(\mathbf{x}')$ and added to the training set $\mathcal{D}$, and the surrogate is refit. A few rounds suffice to converge (details in Appendix C).

Pareto frontier. Since the DP covers all cost levels in a single pass, it directly yields the complete Pareto frontier of quality vs. inference cost. The frontiers in Fig. 3 are obtained by fitting a single surrogate on the combined data from exploration and refinement rounds. ***In summary***: the MRF assumption makes placement quality estimation dependent only on layer-local information enabling DP solver. As detailed in App.C.3, DP state is indexed by layer index, mixer type and cost (expressed as allocation[1]) and hence can be run in parallel on CPU for all possible allocations.

[1]Allocation describes how many of each mixer type, each associated with a fixed cost measure by the cost model, is assigned to a placement; there is finite number of possible allocations.

3 EXPERIMENTS

3.1 SETUP

Training (15B). Distillation totals 266B tokens in two sub-stages: the first distills from Apriel 1.5 (Radhakrishna et al., 2025) for 197B tokens (GDN & KDA mixers only), the second switches to the Apriel 1.6 teacher for 69B tokens, details are in Appendix I. SFT uses 60B tokens of reasoning-heavy data with targeted training (Section 2.2) on 8 placement presets: the `all-FA` placement, 4 presets obtained via surrogate-guided search with the regression cost model (Section 2.3), and 3 with an idealized cost model that estimates per-layer cost from pure-placement throughputs which was used in preliminary experiments before the regression model was calibrated (Appendix E). Training runs on 256 H100 GPUs.

Training (0.5B development model). To enable fast ablations (Section 3.3), we construct a 0.5B supernet from Qwen2-0.5B (Yang et al., 2024a) (24 layers), equipped with the same four mixer types. The base model is first fine-tuned for 18K steps on OpenThoughts (Guha et al., 2025), then distilled for 60K steps into a supernet 0.5B. Details in Appendix I.

Evaluation. During model building and selection we use dev benchmarks: MATH-500 (Hendrycks et al., 2021b), AIME24/25, GSM8K (Cobbe et al., 2021), MMLU (Hendrycks et al., 2021a), FDA, SWDE, and RULER (Hsieh et al., 2024). We use a set of held-out benchmarks to evaluate the final SUPER APRIEL after `S2: SFT` stage: MMLU-Pro (Wang et al., 2024b), GPQA (Rein et al., 2023), HLE (Phan et al., 2026), LiveCodeBench (Jain et al., 2024), τ^2-Bench (Barres et al., 2025), and IFEval (Zhou et al., 2023). Further evaluation details are in Appendix J. For placement scoring $S(\mathbf{x})$ we use either normalized average log-likelihood on teacher-generated math traces (MATH-500, AIME24, AIME25), denoted `Lklhd`, or additionally include cheap dev benchmarks MMLU (10 problems per subject), FDA, SWDE, and RULER, denoted `All`.

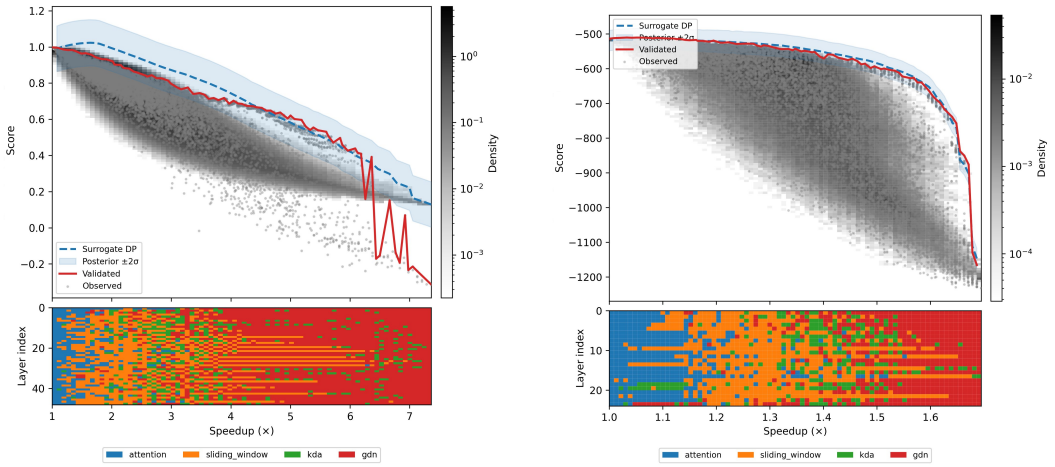
Baselines. We evaluate against multiple hybrids in a similar size category: Falcon-H1 (Zuo et al., 2025) — parallel Mamba-MHA hybrid trained from scratch; Qwen-3.5 27B — an attention-GDN sparse mixture-of-experts (MoE) hybrid; Nemotron-3-Nano 30B (NVIDIA et al., 2025b) — hybrid sparse MoE model trained from scratch as an attention-Mamba hybrid; Apriel-H1 (Ostapenko et al., 2025) — similar sized Mamba hybrid distilled from Apriel 1.5 (Radhakrishna et al., 2025).

Serving modes. As detailed in Appendix G, in *single-preset* mode, only the selected placement is loaded to GPU; changing preset costs one-time weight transfer and CUDA-graph recapture. This suffices whenever the operating point changes at a lower granularity (deployment tier, per-customer SLA, traffic shape etc.), and it is used for all throughput numbers unless stated otherwise. In *supernet* mode, all registered placements are kept on GPU so placements can be switched *per request*, trading GPU memory, KV-cache capacity and maximum batch size for runtime switchability.

3.2 PARETO FRONTIER AND PLACEMENT PRESETS

Fig. 3a shows the 15B placement landscape after distillation. The frontier is smooth up to $\sim 6\times$ speedup; beyond this the decline steepens as the last SWA layers are evicted. The layer-wise heatmap reveals the eviction order: mid-network layers (14–28) retain sliding-window attention longest, with an overall FA→SWA→KDA→GDN transition as the cost budget tightens. Fig. 3b shows that the 0.5B development supernet reproduces a similar qualitative landscape structure yet demonstrates a longer flat region followed by a steep decline.

Table 1 lists the 8 presets used for targeted SFT, all discovered via surrogate-guided search on the final distillation checkpoint `S1: Distil` of the 15B SUPER APRIEL. Each preset is named `{Cost model}|{Proxy tasks}-cost`: the **cost model** is either `Regression` (fitted from measured latencies, Section 2.3) or `Idealized` (from pure-placement throughputs, Appendix E); the **proxy tasks** used to estimate $S(\mathbf{x})$ are either `Lklhd` (trace log-likelihood on math benchmarks only) or `All` (math plus remaining dev tasks); and the **cost** is the placement cost under the respective model.



(a) 15B supernet Pareto frontier

(b) 0.5B development supernet Pareto frontier

Figure 3: Pareto frontiers for (a) 15B supernet and (b) 0.5B development supernet: placement quality vs. inference speedup (according to the `Regression`-based cost model, see Section E). **Top panels:** Score for all feasible placements (grayscale density, log scale), observed supernet evaluations (grey dots), surrogate-predicted optima (blue curve), and validated optima (red solid line). For the 15B model, the score is the average normalized log-likelihood on MATH500, AIME24, and AIME25 traces. For the 0.5B model, the score is the average raw log-likelihood on MATH500 traces. **Bottom panels:** Layer-wise mixer assignment of the optimal placement at each speedup level. Mid-network layers retain attention longest. The 0.5B model reproduces a landscape that has a long flat region with a subsequent steep decline as last SWA mixers are wiped.

Table 1: Recommended presets after targeted SFT. Avg Acc. is average across dev excluding MMLU&NIAH + held-out, (full results are in Table 4). Speedup measured @32k sequence length.

Preset	FA	SWA	KDA	GDN	Avg Acc.	Speedup
all-FA	48	0	0	0	74.2	1.0×
Reg|Lklhd–26	12	26	6	4	71.1 (96%)	2.9×
Reg|Lklhd–18	3	25	4	16	69.7 (94%)	4.8×
Reg|Lklhd–13	0	16	13	19	60.2 (81%)	6.9×
Reg|Lklhd–10	0	10	5	33	57.2 (77%)	10.7×
Idealized|All–18	13	32	1	2	71.8 (97%)	2.0×
Idealized|Lklhd–6	0	30	5	13	66.8 (90%)	6.2×
Idealized|All–6	0	30	5	13	65.3 (88%)	6.1×

3.3 Placement Landscape Dynamics

A key question for supernet training is whether placement rankings stabilize early enough to guide training decisions—in particular, when to switch from stochastic sampling (uniform random mixer per layer) to targeted training (sampling exclusively from a fixed set of deployment presets). We study this via controlled ablations on the 0.5B supernet and confirm key findings at 15B scale.

Rank stability under stochastic training. At a coarse scale, rankings are remarkably stable: Spearman ρ between the earliest checkpoint ($\sim$3% of training) and the final ranking exceeds 0.98 at both scales (Fig. 8a), and Kendall's W (multi-rater concordance across all checkpoints) is near-perfect ($W \approx 1.0$) over the full placement set. However, restricting to the top-5% tier—the frontier placements most relevant for deployment—reveals higher volatility: W=0.77 at 0.5B and W=0.88 at 15B (Fig. 8b). While the ordering within the frontier placements changes, Fig. 4 shows that the *set* of top placements is largely stable: top-50 overlap ≈ 1.0 and top-10 overlap stabilizes to 1 across majority of the cost bins later in training as well (e.g. see step 32k).

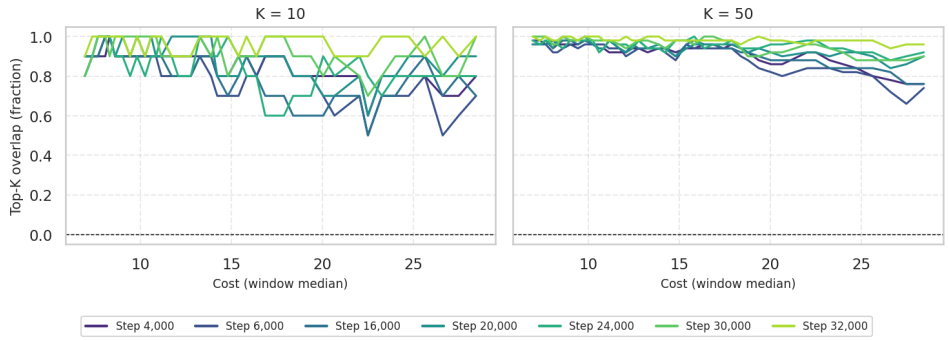


Figure 4: Top-k overlap fraction between early checkpoints and the final checkpoint at 15B for rolling cost-windows of 200 placements. Top-50 overlap is close to 1.0 throughout training, while top-10 overlap is above 0.5 for all training steps and reaches 1 for most cost windows later in training, i.e. at step 32k—showing that the set of frontier placements stabilizes, even though their internal ordering fluctuates (Fig. 8, Appendix).

Training regime ablation (0.5B). Should distillation concentrate on deployment presets or train all placements uniformly? We compare three regimes on identical data: fully stochastic, targeted (cycle through presets only), and hybrid (p=0.5 mix). To track the effect on placements outside the targeted set, we monitor 1,000 random *canary* placements at each checkpoint. In Fig. 9 stochastic training improves all placements uniformly—both canaries and presets. Targeted training gives presets an initial advantage, but stochastic catches up and surpasses it, while canary placements improve much more slowly under targeted training because canaries overlap with the targeted presets. At this scale, stochastic distillation is preferable: it yields the best presets at convergence without sacrificing the broader landscape.

Practical implications. These ablations justify the design of the training procedure for SUPER APRIEL described in Fig. 1. Since rankings among the frontier placements remains somewhat volatile during distillation and stochastic training matches or exceeds targeted training on preset quality, we use fully stochastic sampling throughout distillation. Placement optimization is then run on the final distillation checkpoint. Since the frontier set converges throughout distillation, the SFT stage is conducted with targeted training on the presets selected post distillation (Section 2.2).

3.4 BENCHMARK COMPARISON

Downstream performance. Fig. 5 plots SUPER APRIEL presets' and external baselines' performance vs. speedup: the top panel plots average performance over all benchmarks (dev and unseen) vs. throughput speedup; the bottom panels show math reasoning (left) and all dev tasks (right). The `all-FA` preset matches or exceeds the Apriel-1.6 teacher (74.2 vs. 73.9 all-tasks average). Efficient presets retain strong performance: `Reg|Lklhd-26` achieves 71.1 (96% retention) at 2.9× speedup; `Reg|Lklhd-18` achieves 69.7 (94%) at 4.8×. Full per-benchmark results are in Table 4.

Comparison with external hybrids. The Pareto frontier formed by SUPER APRIEL placements is nearly linear, validating the cost model assumptions. Key comparisons: Apriel-H1 (15B) achieves 2.0× speedup with 58.4 all-tasks average; SUPER APRIEL's `Reg|Lklhd-26` at 2.9× achieves 71.1 (+12.7 points). Qwen-3.5 27B achieves highest absolute performance (80.7) but operates at 0.5× speedup. Nemotron-3-Nano 30B achieves 4.1× at 72.6 average; SUPER APRIEL's `Reg|Lklhd-18` achieves 69.7 at 4.8×. Nemotron-3-Nano 30B dominates the SUPER APRIEL frontier, pointing towards hybrid MoE as a promising direction for future work. Notably, fine-tuning a single SUPER APRIEL placement in isolation (`Idealized|Lklhd-6`[2], 6.2× speedup) achieves 90.9 math average, surpassing Nemotron-3-Nano 30B (89.0 at 4.1×) despite being half the parameter count—

[2]We note that this placement has been finetuned on a version of the SFT dataset, denoted as SFT*, without the tool calling subset.

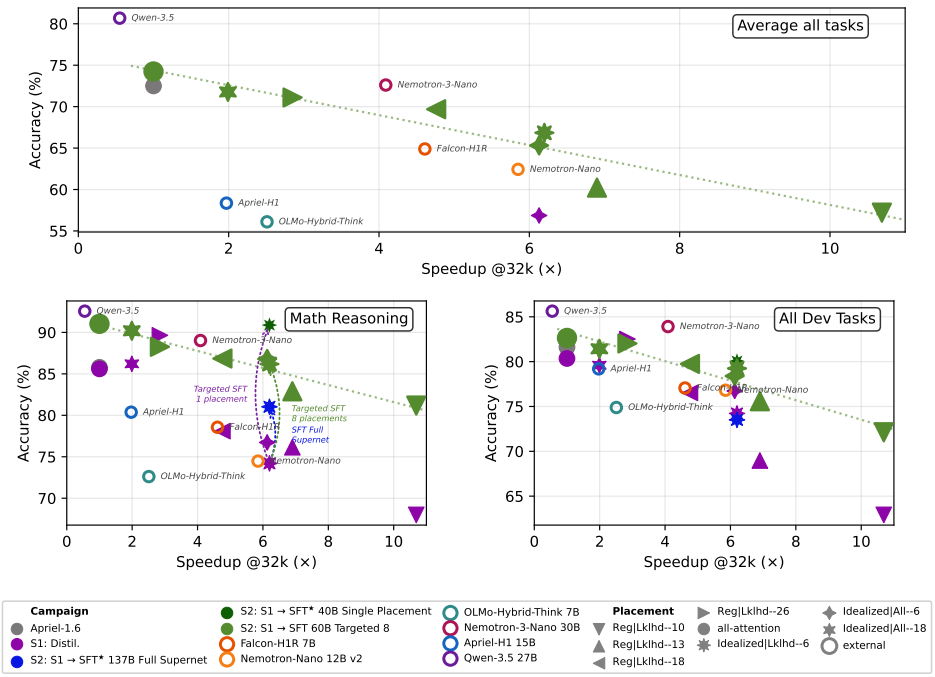


Figure 5: **Performance vs. throughput speedup @32k of `all-FA` preset.** Top: average over all tasks (dev excluding MMLU&NIAH + held-out, Table 4). Bottom left: math reasoning average (AIME'24, AIME'25, MATH-500, GSM8K). Bottom right: average over all tasks (detailed view). Filled markers are SUPER APRIEL presets; open markers are external hybrids. Dotted line shows linear fit to SUPER APRIEL presets. Higher and further right is better.

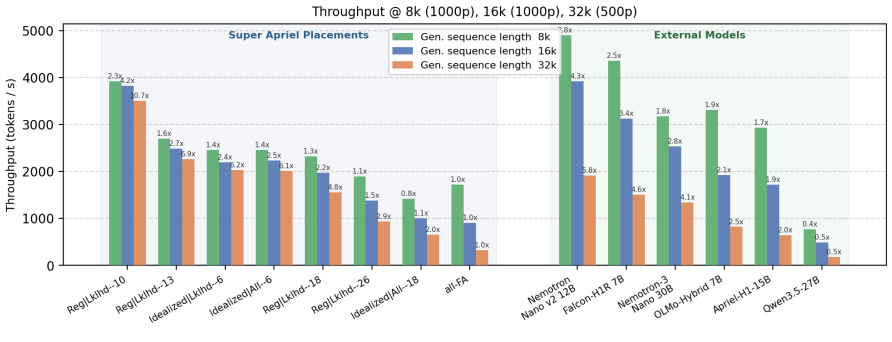


Figure 6: Decode throughput (tokens/s) on a single H100 GPU for SUPER APRIEL placements (left) and external hybrid models (right) at 8k, 16k, and 32k generation lengths. Numbers above bars indicate speedup relative to `all-FA` at the same context length. Efficient placements achieve increasing relative speedups at longer contexts.

demonstrating that domain-specific fine-tuning of discovered placements can push individual operating points beyond the supernet frontier.

Throughput. Fig. 6 reports absolute decode throughput on a single H100 GPU at 8k, 16k, and 32k generation lengths. SUPER APRIEL placements achieve substantially higher relative speedups at longer contexts: from 16k to 32k, relative speedups increase by 81–158%, compared to only 4–46% for external hybrids. This is expected: as context grows, the FA KV-cache cost dominates, and placements with fewer FA layers benefit disproportionately.

Speculative decoding. The supernet also enables speculative decoding without a separate draft model: the `all-FA` placement serves as verifier and any efficient placement as draft. Because draft and verifier share most parameters, acceptance rates remain high across the full cost range. The cheapest draft (all-GDN) yields the largest end-to-end speedup (Fig. 12, Appendix F).

4 Related Work

Hybrid architectures. Several recent models interleave full attention with efficient alternatives at a fixed ratio: Nemotron-H (NVIDIA et al., 2025a) (Mamba-2 + FA, 9:1), Falcon-H1 (Zuo et al., 2025) (parallel Mamba-2 + FA), Jamba (Lieber et al., 2024) (Mamba + FA), and Kimi Linear (Kimi Team et al., 2025) (KDA + MLA, 3:1), as well as OLMo-Hybrid (Merrill et al., 2026) and NVIDIA's models (NVIDIA et al., 2025a;b). All fix placement at training time with repetitive patterns. We instead operate in the conversion regime, where the placement pattern is inferred from the teacher's learned circuits and can be changed at serving time.

Attention-to-efficient conversion. Mamba-in-the-Llama (MIL) (Wang et al., 2024a) showed that pretrained attention weights can initialize linear RNN mixers, enabling conversion with academic-scale compute. MOHAWK (Bick et al., 2025b) proposed multi-stage conversion for Mamba-2; Llamba (Bick et al., 2025a) distilled Llama-3 into pure Mamba-2 at 1B–8B scale. Apriel-H1 (Ostapenko et al., 2025) extended MIL with progressive conversion and reverse-KL distillation applied to reasoning models at production scale. RADLADS (Goldstein et al., 2026) combines multiple techniques and applies them to effectively convert pretrained base and instruct models into RWKV (Peng et al., 2023) hybrids. Jet-Nemotron (Gu et al., 2025) (JNT) introduced PostNAS at a 4B scale that constructs a similar supernet but uses beam search over a mixer vocabulary (FA, SWA, JetBlock). We extend this to production-scale reasoning model and replace beam search with a surrogate guided search, covering all cost-feasible placements rather than the limited subset explored by beam search. Importantly, this work aims for deployment-time flexibility, while JNT primarily targets a single operating point, using supernet solely to estimate where to place attention layers.

Neural architecture search. Weight-sharing NAS supernets (Pham et al., 2018; Bender et al., 2018; Guo et al., 2020; Cai et al., 2020) train a single overparameterized network that supports many sub-architectures, using path sampling (Pham et al., 2018; Bender et al., 2018; Guo et al., 2020) or progressive shrinking (Cai et al., 2020). Predictor-based NAS methods (Kandasamy et al., 2018; White et al., 2021) train surrogate models to predict architecture quality from structural features. Our supernet differs in initialization (warm-started from teacher weights), objective (distillation rather than self-supervised training), and the nature of the choices (fundamentally different architectures—attention vs. linear recurrence—rather than parametric variants of the same operation).

Surrogate-assisted optimization. Our explore–exploit loop follows the Bayesian optimization paradigm (Jones et al., 1998; Shahriari et al., 2016). The cluster expansion (Sanchez et al., 1984; de Fontaine, 1994) is standard in materials science for parameterizing configuration-dependent properties of crystalline alloys; we adapt it to neural architecture search, with transformer layers as lattice sites and mixer types as atomic species. The linear decomposable structure enables exact DP optimization rather than heuristic search (Hutter et al., 2011; White et al., 2021).

5 Conclusion

Super Apriel demonstrates that a single supernet checkpoint can serve multiple speed–quality operating points by training all mixer options at every layer. Surrogate search enables efficient navigation of the 4^{48} placement space, and targeted SFT preserves quality across the Pareto frontier. This shifts the throughput–quality tradeoff from a release-time commitment to a runtime control.

Limitations. As noted in Section 2.1, the full supernet carries a $\sim$67% parameter overhead ($\sim$25B total vs. $\sim$15B per placement). Placement search requires proxy task evaluations. Long-context retrieval degrades as FA layers are removed (NIAH, RULER) while long-context reasoning is well preserved (MATH, AIME). Current linear-mixer kernels require `eager-mode` for numerical stability, so benchmark evaluations use eager-mode while throughput measurements use CUDA graphs—the two execution modes may not yield identical results (Appendix H).

AI USE STATEMENT

In this work, we used generative AI tools for helping design conceptual frameworks and implementing parts of the proposed methods, as well as interpreting and providing feedback on the results of the experiments and methodology. We have not used generative AI tools for these tasks: generate synthetic data sets, formulate mathematical claims, propose or refine hypotheses, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis; and these tasks are not applicable to this work: provide critical ingredients for proving mathematical claims, assist in the writing of proofs.

Additionally, we used generative AI tools for creating scientific figures and creating and editing software code and polishing some of the manuscript's text.

We have reviewed all AI-assisted work: LLM-generated code was tested using unit and integration tests and reviewed by multiple authors for its correctness. The parts of the manuscript that were polished/edited by AI assistants were reviewed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

In order to ensure reproducibility of this research we undertake the following efforts:

- The training framework that implements training code required to train all the novel models presented in this manuscript is open-source (cannot be named in the interest of anonymity);
- We will release the code required for the architecture search part of the work; that code is not yet ready for release as it must go through organization-internal clearance process before the release;
- We will openly release training logs (training, validation curves etc.) upon acceptance;
- We will openly release the training checkpoints that were the outcome of the research described in this manuscript upon acceptance;
- We clearly state the training and inference (vLLM) hyperparameters used to produced results described in this manuscript; we also state versions of critical packages such as vLLM and observed limitations (see e.g., Appendix H).

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, et al. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, et al. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-main.298/.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, et al. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes, 2025. URL https://arxiv.org/abs/2304.09433.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment, 2025. URL https://arxiv.org/abs/2506.07982.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and Simplifying One-Shot Architecture Search. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 550–559. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/bender18a.html.

Aviv Bick, Tobias Katsch, Nimit Sohoni, Arjun Desai, and Albert Gu. Llamba: Scaling Distilled Recurrent Models for Efficient Language Processing, 2025a. URL https://arxiv.org/abs/2502.14458.

Aviv Bick, Kevin Y. Li, Eric P. Xing, J. Zico Kolter, and Albert Gu. Transformers to SSMs: Distilling Quadratic Knowledge to Subquadratic Models, 2025b. URL https://arxiv.org/abs/2408.10189.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, et al. Training Verifiers to Solve Math Word Problems, 2021. URL https://arxiv.org/abs/2110.14168.

D. de Fontaine. Cluster Approach to Order-Disorder Transformations in Alloys. In *Solid State Physics*, volume 47, pp. 33–176. Elsevier, 1994. URL https://doi.org/10.1016/S0081-1947(08)60639-6.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, et al. The Language Model Evaluation Harness, July 2024. URL https://zenodo.org/records/12608602.

Daniel Goldstein, Eric Alcaide, Janna Lu, and Eugene Cheah. RADLADS: Rapid Attention Distillation to Linear Attention Decoders at Scale, 2026. URL https://arxiv.org/abs/2505.03005.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Yuxian Gu, Qinghao Hu, Shang Yang, Haocheng Xi, Junyu Chen, et al. Jet-Nemotron: Efficient Language Model with Post Neural Architecture Search, 2025. URL https://arxiv.org/abs/2508.15884.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, et al. OpenThoughts: Data Recipes for Reasoning Models, 2025. URL https://arxiv.org/abs/2506.04178.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, et al. Single Path One-Shot Neural Architecture Search with Uniform Sampling. In *Computer Vision – ECCV 2020*, pp. 544–560, Cham, 2020. Springer International Publishing. URL https://doi.org/10.1007/978-3-030-58517-4_32.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, et al. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, et al. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/be83ab3ecd0db773eb2dc1b0a17836a1-Paper-round2.pdf.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, et al. RULER: What's the Real Context Size of Your Long-Context Language Models?, 2024. URL https://arxiv.org/abs/2404.06654.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization*, pp. 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. URL https://doi.org/10.1007/978-3-642-25566-3_40.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, et al. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code, 2024. URL https://arxiv.org/abs/2403.07974.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, December 1998. URL https://doi.org/10.1023/A:1008306431147.

Greg Kamradt. Needle In A Haystack. Software repository, 2023. URL https://github.com/gkamradt/needle-in-a-haystack.

Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/f33ba15effa5c10e873bf3842afb46a6-Paper.pdf.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, et al. Kimi Linear: An Expressive, Efficient Attention Architecture, 2025. URL https://arxiv.org/abs/2510.26692.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, et al. Efficient Memory Management for Large Language Model Serving with PagedAttention, 2023. URL https://arxiv.org/abs/2309.06180.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, et al. Jamba: A Hybrid Transformer-Mamba Language Model, 2024. URL https://arxiv.org/abs/2403.19887.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. OpenCeres: When Open Information Extraction Meets the Semi-Structured Web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3047–3056, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. URL https://aclanthology.org/N19-1309/.

David J. C. MacKay. Bayesian Interpolation. *Neural Computation*, 4(3):415–447, May 1992. URL https://doi.org/10.1162/neco.1992.4.3.415.

Sathwik Tejaswi Madhusudhan, Sagar Davasam, and Torsten Scholak. Apriel-1.6-15b-Thinker: Cost-efficient Frontier Multimodal Performance. Hugging Face blog, December 2025. URL https://huggingface.co/blog/ServiceNow-AI/apriel-1p6-15b-thinker.

William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, et al. Olmo Hybrid: From Theory to Practice and Back, 2026. URL https://arxiv.org/abs/2604.03444.

NVIDIA. NeMo Skills: A project to improve skills of large language models. Software repository, 2024. URL https://github.com/NVIDIA-NeMo/Skills.

NVIDIA, Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, et al. Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models, 2025a. URL https://arxiv.org/abs/2504.03624.

NVIDIA, Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, et al. Nemotron 3 Nano: Open, Efficient Mixture-of-Experts Hybrid Mamba-Transformer Model for Agentic Reasoning, 2025b. URL https://arxiv.org/abs/2512.20848.

Oleksiy Ostapenko, Luke Kumar, Raymond Li, Denis Kocetkov, Joel Lamy-Poirier, et al. Apriel-H1: Towards Efficient Enterprise Reasoning Models, 2025. URL https://arxiv.org/abs/2511.02651.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, et al. RWKV: Reinventing RNNs for the Transformer Era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14048–14077, Singapore, December 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.findings-emnlp.936/.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4095–4104. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/pham18a.html.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, et al. Humanity's Last Exam, 2026. URL https://arxiv.org/abs/2501.14249.

Shruthan Radhakrishna, Aman Tiwari, Aanjaneya Shukla, Masoud Hashemi, Rishabh Maheshwary, et al. Apriel-1.5-15b-Thinker, 2025. URL https://arxiv.org/abs/2510.01141.

Negin Raoof, Etash Kumar Guha, Ryan Marten, Jean Mercat, Eric Frankel, et al. Evalchemy, June 2025. URL https://github.com/mlfoundations/evalchemy.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, et al. GPQA: A Graduate-Level Google-Proof Q&A Benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

J.M. Sanchez, F. Ducastelle, and D. Gratias. Generalized cluster description of multicomponent systems. *Physica A: Statistical Mechanics and its Applications*, 128(1-2):334–350, November 1984. URL https://doi.org/10.1016/0378-4371(84)90096-7.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, et al. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision, 2024. URL https://arxiv.org/abs/2407.08608.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1): 148–175, January 2016. URL https://doi.org/10.1109/JPROC.2015.2494218.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, et al. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568:127063, February 2024. URL https://doi.org/10.1016/j.neucom.2023.127063.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967. URL https://doi.org/10.1109/TIT.1967.1054010.

Junxiong Wang, Daniele Paliotta, Avner May, Alexander M. Rush, and Tri Dao. The Mamba in the Llama: Distilling and Accelerating Hybrid Models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62432–62457. Curran Associates, Inc., 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/723933067ad315269b620bc0d2c05cba-Paper-Conference.pdf.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, et al. MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark, 2024b. URL https://arxiv.org/abs/2406.01574.

Colin White, Willie Neiswanger, and Yash Savani. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10293–10301, May 2021. URL https://doi.org/10.1609/aaai.v35i12.17233.

Bernard Widrow and Marcian E. Hoff, Jr. Adaptive Switching Circuits. In *IRE WESCON Convention Record*, volume 4, pp. 96–104, August 1960. URL https://isl.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2 Technical Report, 2024a. URL https://arxiv.org/abs/2407.10671.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing Linear Transformers with the Delta Rule over Sequence Length. In *Advances in Neural Information Processing Systems*, volume 37, pp. 115491–115522. Curran Associates, Inc., 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d13a3eae72366e61dfdc7eea82eeb685-Paper-Conference.pdf.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, et al. Instruction-Following Evaluation for Large Language Models, 2023. URL https://arxiv.org/abs/2311.07911.

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, et al. Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance, 2025. URL https://arxiv.org/abs/2507.22448.

A Additional Tables

Table 2: Training data composition: token proportions by source category. We use the `Distillation` dataset for all the distillation runs and some ablations. We use `SFT` dataset for all post-distillation fine-tuning runs unless stated otherwise. Some earlier training runs used a starred `SFT`* version, which excludes the tool use subset.

Category	**Distillation (%)**
Reasoning traces & SFT	29.3
Code	10.0
Math & STEM	5.0
Web / encyclopedic text	6.7
Image (multimodal)	49.0
Category	**SFT (%)**
SFT: Code	36.1
SFT: Math & STEM	38.7
SFT: Chat, generic reasoning & IF	11.3
SFT: Tool use	12.0
SFT: Safety, robustness & content moderation	2.0

Table 3: Evaluation results across campaigns and dev. tasks.

Campaign	Config	@32k	AIME'24	AIME'25	MATH-500	GSM8K	MMLU	FDA	SWDE	RULER	NIAH	Math	All
S1: Distil. 69B	all-attention	1.0×	90.0	86.7	85.0	80.7	64.7	81.3	91.0	63.4	99.8	85.6	80.4
	Idealized|Lklhd–6	6.2×	76.7	60.0	83.4	76.7	63.0	79.4	90.2	64.4	27.2	74.2	74.2
	Idealized|All–6	6.1×	63.3	63.3	84.2	96.1	59.6	80.9	87.8	78.7	26.6	76.7	76.7
	Idealized|All–7	4.3×	76.7	56.7	85.4	95.0	63.5	81.6	90.4	67.1	29.8	78.4	77.0
	Idealized|All–18	2.0×	86.7	76.7	85.6	95.9	64.2	81.4	89.5	56.9	82.0	86.2	79.6
	Reg|Lklhd–10	10.7×	56.7	40.0	88.4	86.9	56.8	59.7	78.0	36.7	25.9	68.0	62.9
	Reg|Lklhd–13	6.9×	66.7	56.7	89.0	92.3	63.9	51.0	77.0	55.4	28.4	76.2	69.0
	Reg|Lklhd–18	4.8×	63.3	63.3	90.4	95.3	67.5	78.5	89.1	64.6	36.6	78.1	76.5
	Reg|Lklhd–26	2.9×	80.0	93.3	90.8	94.5	69.1	77.8	88.6	66.1	67.6	89.6	82.5
S2: S1 → SFT(1) 40B Single Placement	Idealized|Lklhd–6	6.2×	93.3	86.7	93.0	90.5	55.3	71.1	85.8	64.9	23.0	90.9	80.1
S2: S1 → SFT(2) 60B Targeted 8 placements	all-attention	1.0×	93.3	86.7	91.8	92.3	49.8	78.3	89.5	79.4	100.0	91.0	82.6
	Idealized|All–6	6.1×	83.3	80.0	92.2	91.7	55.1	75.6	87.4	61.9	30.8	86.8	78.4
	Idealized|All–18	2.0×	90.0	86.7	92.0	92.1	58.9	78.0	86.6	67.1	78.8	90.2	81.4
	Idealized|Lklhd–6	6.2×	83.3	76.7	92.4	92.3	57.4	76.9	88.9	66.1	25.8	86.2	79.3
	Reg|Lklhd–10	10.7×	76.7	66.7	90.6	90.8	55.6	65.2	82.9	48.6	27.4	81.2	72.1
	Reg|Lklhd–13	6.9×	76.7	73.3	90.4	91.2	62.5	68.6	85.1	57.0	28.0	82.9	75.6
	Reg|Lklhd–18	4.8×	86.7	76.7	92.4	91.7	58.6	81.7	89.8	60.5	57.0	86.8	79.7
	Reg|Lklhd–26	2.9×	86.7	83.3	92.0	91.1	60.7	79.9	88.2	74.4	98.8	88.3	82.0
Falcon-H1R 7B	Hybrid	4.6×	66.7	70.0	90.0	87.6	68.1	63.6	89.6	72.0	92.4	78.6	75.9
Nemotron-Nano 12B v2	Hybrid	5.9×	60.0	60.0	83.6	94.3	78.2	82.5	92.1	65.2	58.0	74.5	77.0
OLMo-Hybrid-Think 7B	Hybrid	2.5×	66.7	46.7	87.8	89.3	69.4	78.5	89.5	65.8	75.4	72.6	74.2
Nemotron-3-Nano 30B	Hybrid MoE	4.1×	93.3	76.7	89.8	96.3	76.8	74.7	91.7	64.9	97.8	89.0	83.0
Apriel-H1 15B	Hybrid	2.0×	80.0	53.3	92.8	95.4	64.4	73.2	88.6	71.0	85.6	80.4	77.3
Qwen-3.5 27B	Hybrid MoE	0.5×	90.0	93.3	89.8	97.1	87.1	84.4	90.4	54.4	95.4	92.6	85.8

Table 4: Full benchmark results for the `S2: S1 → SFT(2) 60B Targeted 8` across all placements and tasks. For one placement we also include numbers from the `S1: Distil.` stage. We also include speedups @32k and @16k generation sequence lengths, as described in Appendix H.

Config	@32k	@16k	AIME'24	AIME'25	MATH-500	GSM8K	FDA	SWDE	RULER	Tau2	MMLU-Pro	AIME(NV)	GPQA	HLE	LCB	IFEval	All
Reg|Lklhd–10	10.69×	4.2×	76.7	66.7	90.6	90.8	65.2	82.9	48.6	23.4	68.2	24.4	52.5	4.5	50.2	56.2	57.2
Reg|Lklhd–13	6.9×	2.7×	76.7	73.3	90.4	91.2	68.6	85.1	57.0	28.6	69.3	26.7	61.2	5.5	52.6	57.1	60.2
Idealized|Lklhd–6	6.2×	2.4×	83.3	76.7	92.4	92.3	76.9	88.9	66.1	40.4	73.6	62.2	65.0	6.1	57.0	54.9	66.8
Idealized|All–6	6.13×	2.5×	83.3	80.0	92.2	91.7	75.6	87.4	61.9	34.2	73.3	56.7	61.0	5.9	55.9	55.3	65.3
Reg|Lklhd–18	4.76×	2.2×	86.7	76.7	92.4	91.7	81.7	89.8	60.5	46.2	76.3	74.4	68.7	6.6	64.8	59.3	69.7
Reg|Lklhd–26	2.85×	1.5×	86.7	83.3	92.0	91.1	79.9	88.2	74.4	30.7	76.2	80.0	70.7	10.0	69.2	63.1	71.1
Idealized|All–18	1.99×	1.1×	90.0	86.7	92.0	92.1	78.0	86.6	67.1	52.6	76.3	82.2	68.2	6.9	67.3	58.8	71.8
all-attention	1.0×	1.0×	93.3	86.7	91.8	92.3	78.3	89.5	79.4	56.7	76.8	82.7	72.0	8.2	68.6	63.1	74.2
S1: Distil. Idealized|All–6	6.13×	2.5×	63.3	63.3	84.2	96.1	80.9	87.8	78.7	12.3	74.5	43.7	57.9	5.0	44.4	59.9	60.8
Apriel-1.6	1.0×	1.0×	90.0	83.3	86.2	83.6	84.5	90.2	69.1	62.6	78.8	81.7	70.7	8.8	78.8	66.5	73.9
OLMo-Hybrid-Think 7B	2.51×	2.1×	66.7	46.7	87.8	89.3	78.5	89.5	65.8	9.7	65.7	55.2	47.0	5.1	48.6	30.0	56.1
Nemotron-Nano 12B v2	5.85×	4.3×	60.0	60.0	83.6	94.3	82.5	92.1	65.2	9.1	77.2	76.2	64.5	4.2	70.9	34.2	62.4
Falcon-H1R 7B	4.61×	3.4×	66.7	70.0	90.0	87.6	63.6	89.6	72.0	11.1	72.5	78.3	61.3	11.1	74.3	60.5	64.9
Nemotron-3-Nano 30B	4.09×	2.8×	93.3	76.7	89.8	96.3	74.7	91.7	64.9	31.3	78.3	89.1	73.0	10.6	75.3	71.5	72.6
Qwen-3.5 27B	0.55×	0.5×	90.0	93.3	89.8	97.1	84.4	90.4	54.4	93.6	86.1	83.7	85.5	18.0	85.5	77.8	80.7
Apriel-H1 15B	1.97×	1.9×	80.0	53.3	92.8	95.4	73.2	88.6	71.0	4.7	67.5	48.7	52.3	5.7	54.0	29.8	58.4

B Architecture Details

B.1 Teacher model

Apriel 1.6 (Madhusudhan et al., 2025) is a 15B-parameter multimodal transformer derived from Pixtral-12B via depth upscaling (increasing the number of layers) (Radhakrishna et al., 2025): 48 decoder layers with grouped-query attention (32 query / 8 KV heads, d_h=128), hidden dimension d=5120, SiLU-gated FFN of width 14 336, vocabulary of 131 072 tokens, and a context window of 262 400 positions. A Pixtral vision encoder and learned projector handle image inputs.

B.2 Full Attention (FA)

Standard grouped-query attention (Ainslie et al., 2023) with rotary position embeddings (Su et al., 2024). The query, key, and value projections follow the teacher's configuration. FA computes exact pairwise token interactions with $O(n^2)$ time and $O(n)$ KV cache per layer.

At decode step t, the KV-read operational intensity per layer is approximately $h_q/(2\,h_{kv}\,b)$, where h_q and h_{kv} are the numbers of query and KV heads and b is the bytes per element. For typical GQA configurations on current hardware, this is orders of magnitude below the compute–memory ridge point, placing FA decode deep in bandwidth-bound territory. The decode latency contribution of each FA layer grows linearly with context length: $T_{\text{decode}}(t) \sim 2t\,h_{kv}\,d_h\,b\,/\,B_{\text{HBM}}$.

B.3 Sliding Window Attention (SWA)

Attention restricted to a fixed local window of $w = 4096$ tokens. Each token attends only to the w most recent positions, yielding $O(nw)$ time and $O(w)$ KV cache per layer. The query, key, value, and output projection weights are identical to FA; only the attention mask changes.

B.4 Gated DeltaNet (GDN)

Gated DeltaNet (Yang et al., 2025) combines a scalar gating factor $\alpha_t \in (0, 1)$ for adaptive global forgetting with the delta rule (Widrow & Hoff, 1960) for targeted memory updates. The recurrence is:

$$\mathbf{S}_t = \alpha_t \left(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top\right) \mathbf{S}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top$$

where $\mathbf{k}_t, \mathbf{v}_t$ are the key and value vectors, β_t is a scalar learning rate, and the projection $(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top)$ erases the component of $\mathbf{S}_{t-1}$ along $\mathbf{k}_t$ before writing the new association. The scalar gate α_t enables rapid global forgetting when context shifts.

GDN uses a fused input projection producing Q, K, V, and a modulation vector $\mathbf{z}$ in a single matrix multiply, plus a separate projection for the gating parameters (β, α). A short causal 1D convolution with SiLU activation is applied jointly to Q, K, V before the recurrence. The gating computation produces $\beta = \sigma(\beta_{\text{raw}})$ and $g = -\exp(A_{\log}) \cdot \text{softplus}(\alpha + \text{dt_bias})$, combining learned and data-dependent decay. After the chunkwise recurrence, outputs pass through gated RMS normalization (with $\mathbf{z}$ as the gate) and then the output projection.

For Super Apriel Apriel, GDN is configured with 32 value heads, 8 key heads, and $d_k = d_v = 128$, giving a state matrix $\mathbf{S} \in \mathbb{R}^{128\times128}$ per key head. Queries and keys are L_2-normalized. Training uses a chunkwise parallel algorithm based on the WY representation (Yang et al., 2024b).

B.5 Kimi Delta Attention (KDA)

KDA (Kimi Team et al., 2025) extends GDN with channel-wise (vector-valued) gating. Where GDN applies a single scalar gate uniformly across all dimensions, KDA uses a per-dimension gate vector $\boldsymbol{\alpha}_t \in (0, 1)^{d_k}$:

$$\mathbf{S}_t = \left(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top\right) \text{diag}(\boldsymbol{\alpha}_t)\, \mathbf{S}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top$$

Different state dimensions can decay at different rates: some maintain long-lived factual associations while others rapidly update working memory.

KDA uses separate Q, K, V projections, each followed by its own causal 1D convolution with SiLU activation. The per-dimension gate $\boldsymbol{\alpha}_t$ is produced by a low-rank factorization: $\mathbb{R}^d \to \mathbb{R}^{d_h} \to \mathbb{R}^{n_h d_h}$,

followed by the fused gate computation. A separate output gate with the same low-rank structure modulates the recurrence output before the final projection. For SUPER APRIEL Apriel, KDA uses 32 heads with $d_h = 128$. The efficient chunkwise training algorithm uses DPLR transition matrices (Kimi Team et al., 2025).

B.6 MIXER INITIALIZATION (DIL/KIL)

GDN and KDA mixers are initialized from the teacher's own attention weights, without requiring a separate distillation checkpoint.

Delta-in-the-Llama (DIL) — GDN initialization. The teacher's Q, K, V projections are mapped into GDN's fused input structure via cyclic head tiling: for GDN key head g, query rows come from teacher Q head $g \bmod h_q^{\text{src}}$ and key rows from teacher KV head $g \bmod h_{kv}^{\text{src}}$. The output projection is copied from the teacher. Gate projections (β, α) and modulation vector $\mathbf{z}$ are initialized to zeros; the causal convolution uses a scaled identity (0.5 on the last kernel tap); decay $A_{\log} = \log(0.1)$.

Kimi-in-the-Llama (KIL) — KDA initialization. Q, K, V projections are transferred individually (not fused) with cyclic tiling when the target head count exceeds the source. The output projection is copied directly. The per-dimension decay gate and output gate (both low-rank) use Kaiming normal initialization; per-head β is randomly initialized; causal convolutions use scaled identities; $A_{\log}$ and dt_bias follow DIL.

B.7 0.5B DEVELOPMENT SUPERNET

Architecture and initialization. The development supernet is derived from Qwen2-0.5B (Yang et al., 2024a): 24 layers, d=896, 14 query / 2 KV heads (d_h=64), vocabulary of 151 936 tokens, text-only (no vision encoder). Qwen2 has no architectural relationship to the Apriel lineage; it was chosen as a widely available small GQA transformer for fast ablation turnaround. The supernet equips every layer with the same four mixer types as SUPER APRIEL Apriel (FA, SWA, GDN, KDA), using the general initialization described in Section 2.1: FA and SWA weights come from Qwen2's own attention, and GDN/KDA projections are initialized via DIL/KIL (Appendix B.6) from those same weights. Unlike the 15B supernet, there is no prior distillation stage—all four mixers start from the same source checkpoint and are trained simultaneously. This model serves as a controlled proxy: small enough for fast iteration across multiple training regimes, large enough for the landscape dynamics to be meaningful.

Training. The supernet is trained in two stages. First, Qwen2-0.5B is fine-tuned for 18 000 steps (learning rate 5×10^{-4}, sequence length 8 192, batch size 128) with standard next-token prediction on a reasoning dataset (Guha et al., 2025) with 1M high-quality examples covering math, science, code, and puzzles. Both the teacher (full-attention variant) and the student (supernet with all four mixer types) are then initialized from this checkpoint and the student is distilled for 60 000 steps (learning rate 10^{-4}, constant schedule, 100 warmup steps) using per-block activation distillation (λ_{act}=1.0), reverse KL (λ_{RKL}=0.9), and forward KL (λ_{FKL}=0.1); the standard LM loss is disabled. At each step, one mixer per layer is sampled uniformly at random. The full hyperparameter set is in Table 9.

C SURROGATE SEARCH DETAILS

C.1 FEATURE INDICATOR EQUATIONS

Order 1 (self-energy). For each layer i and type $m \in \mathcal{M}$:

$$\phi_{i,m}^{(1)} = \mathbb{1}[x_i = m]$$

These $L \times |\mathcal{M}|$ indicators capture per-layer preferences. The $|\mathcal{M}|$ allocation counts $n_m = \sum_i \phi_{i,m}^{(1)}$ are included as explicit features.

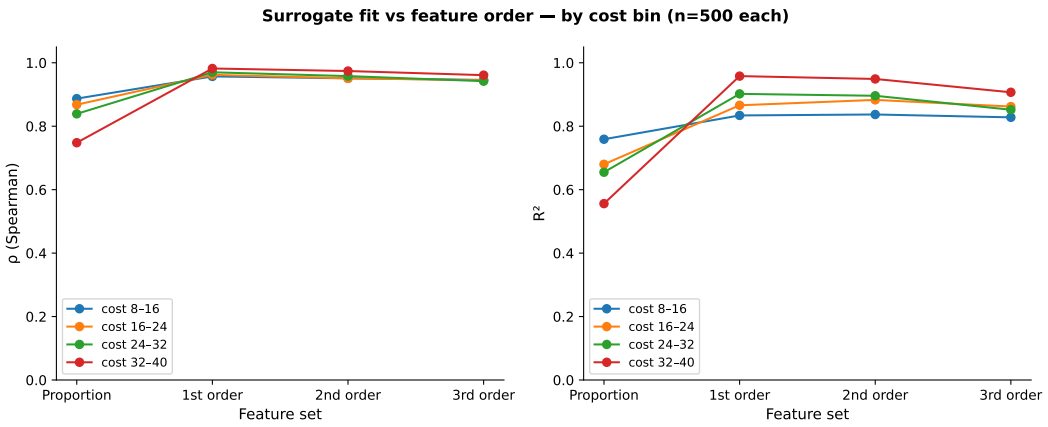


Figure 7: Surrogate fit quality as a function of feature expansion order, stratified by placement cost. Each line corresponds to a cost bin (width 8, n=500 per bin, sampled from a 10k exploration run); placement cost is computed using `Regression` cost model. Metrics are computed via 5-fold cross-validated Ridge regression. Left: Spearman ρ; right: R^2. Moving from proportion features (4 features) to 1st-order features yields the largest gain across all cost levels; higher-order terms provide diminishing returns. Growing ρ and R^2 justify the MRF decomposition for the surrogate based placement search.

Order 2 (pairwise interactions). For each pair of layers (i, j) with $|i - j| \leq r_{\max}$ and each pair of types (m, m'):

$$\phi^{(2)}_{ij,mm'} = \mathbb{1}[x_i = m \wedge x_j = m']$$

Order 3 (triplet interactions). For contiguous triplets $(i, i{+}1, i{+}2)$ and type triples (m, m', m''):

$$\phi^{(3)}_{i,mm'm''} = \mathbb{1}[x_i = m \wedge x_{i+1} = m' \wedge x_{i+2} = m'']$$

Feature counts are in Table 5.

Table 5: Feature counts at each expansion order ($L = 48$, $|\mathcal{M}| = 4$).

Order	Features	Cumulative
1 (unary)	$L\|\mathcal{M}\| + \|\mathcal{M}\|$	196
2 (range 1)	$(L{-}1)\|\mathcal{M}\|^2$	948
2 (range 2)	$(L{-}2)\|\mathcal{M}\|^2$	1,684
2 (range 3)	$(L{-}3)\|\mathcal{M}\|^2$	2,404
3 (triplets)	$(L{-}2)\|\mathcal{M}\|^3$	5,348

C.2 Bayesian Linear Surrogate

The surrogate is a Bayesian linear model with a Normal-Inverse-Gamma (NIG) conjugate prior on the weight vector $\mathbf{w}$ and noise variance σ^2:

$$\mathbf{w} \mid \sigma^2 \sim \mathcal{N}(\mathbf{0},\, \sigma^2 \alpha^{-1} \mathbf{I}), \qquad \sigma^2 \sim \text{InvGamma}(a_0, b_0)$$

Given data $(\mathbf{X}, \mathbf{y})$ where $\mathbf{X}$ is the cluster-expansion feature matrix, the posterior is available in closed form:

$$\boldsymbol{\mu}_n = \mathbf{V}_n \mathbf{X}^\top \mathbf{y}, \qquad \mathbf{V}_n = (\mathbf{X}^\top \mathbf{X} + \alpha \mathbf{I})^{-1}$$

The predictive distribution for a new placement $\mathbf{x}^*$ is Student-t with scale:

$$\hat{\sigma}(\mathbf{x}^*) = \sqrt{c\,(1 + \boldsymbol{\phi}(\mathbf{x}^*)^\top \mathbf{V}_n\, \boldsymbol{\phi}(\mathbf{x}^*))}$$

where $c = b_n/a_n$. The variance grows with distance from the training data, providing calibrated uncertainty without resampling.

The log marginal likelihood,

$$\log p(\mathbf{y} \mid \alpha, a_0, b_0) = -\tfrac{n}{2}\log 2\pi + \tfrac{d}{2}\log\alpha - \tfrac{1}{2}\log|\mathbf{A}| + a_0\log b_0 - a_n\log b_n + \log\Gamma(a_n) - \log\Gamma(a_0)$$

where $\mathbf{A} = \mathbf{X}^\top\mathbf{X} + \alpha\mathbf{I}$, is used for expansion-order selection (Section C.5).

The posterior-mean coefficients $\boldsymbol{\mu}_n$ decompose as MRF potentials—unary $V_i(m)$, pairwise $V_{ij}(m, m')$, and (when the expansion includes order 3) triplet $V_{ijk}(m, m', m'')$—which admit the exact Viterbi optimization described next.

C.3 VITERBI DP

A placement's cost depends only on its *allocation* $\mathbf{a} = (n_1, \ldots, n_{|\mathcal{M}|})$—the number of layers assigned to each mixer type, $n_m = \sum_i \mathbb{1}[x_i{=}m]$—so enforcing a cost budget requires tracking $\mathbf{a}$ alongside the chain state. We therefore index the cost-constrained Viterbi DP by the tuple $(i,\ m,\ \mathbf{a})$: the current layer i, the mixer $m = x_i$ assigned at that layer (sufficient to evaluate the range-1 pairwise potential $V_{i,i+1}$), and the running allocation $\mathbf{a}$ accumulated over layers $1, \ldots, i$. Let $B[i, m, \mathbf{a}]$ denote the best partial score of any prefix ending in state $(i, m, \mathbf{a})$; the recurrence is

$$B[i, m, \mathbf{a}] = V_i(m) + \max_{m'} \Big\{ B[i{-}1,\ m',\ \mathbf{a} - \mathbf{e}_m] + V_{i-1,i}(m', m) \Big\},$$

where $\mathbf{e}_m$ is the unit vector for mixer m (each step increments coordinate m of $\mathbf{a}$), and states whose allocation exceeds the budget are pruned. For $|\mathcal{M}| = 4$, $L = 48$ with pairwise range 1, this yields approximately 80,000 reachable states at the final layer, and the forward pass completes in seconds on CPU. Each feasible allocation corresponds to a distinct terminal $\mathbf{a}$; the optimal placement per allocation is recovered by backtracking. All cost-feasible allocations—up to $\binom{L+|\mathcal{M}|-1}{|\mathcal{M}|-1}$ in the unconstrained case (20,825 for $L{=}48$, $|\mathcal{M}|{=}4$)—are solved simultaneously in a single pass. For interaction range $r > 1$, the state retains the last r mixer assignments $(x_{i-r+1}, \ldots, x_i)$ in place of the single mixer m.

C.4 RISK-BUCKET CANDIDATE SELECTION

Candidate placements for supernet evaluation are generated from the posterior mean and selected using a risk-bucket scheme that controls the explore–exploit balance.

Candidate generation. Extract MRF potentials from $\boldsymbol{\mu}_n$, run the all-allocations Viterbi pass, and keep the top M placements per allocation to obtain an allocation-diverse candidate pool.

Scoring and selection. Each candidate is scored under two buckets that combine the posterior predictive mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$:

- **Safe** (sign $= -1$): score $= \hat{\mu} - \beta\,\hat{\sigma}$. Penalizes uncertainty, selecting placements the model is confident are good (exploitation).
- **Upside** (sign $= +1$): score $= \hat{\mu} + \beta\,\hat{\sigma}$. Rewards uncertainty, selecting placements whose true quality could exceed the prediction (exploration).

A configurable quota (e.g., 70% safe / 30% upside) apportions the per-round evaluation budget across buckets. Each bucket independently ranks its eligible candidates and selects the top k by its scoring rule; ties are broken by $\hat{\mu}$. A mean-floor filter optionally restricts each bucket to candidates above a specified quantile of $\hat{\mu}$ in the pool.

Refinement loop. The selected placements are evaluated against the real supernet, added to the training set, and the surrogate is refit. Defaults: 4 refinement rounds, 500 evaluations per round. The candidate pool targets 1000 placements, distributed as $\lceil 1000 \,/\, |\text{feasible allocations}| \rceil$ per allocation to maintain diversity across cost levels.

Algorithm 1 Exploration: adaptive random sampling

Require: cost model c, N, K, `min_samples`, `batch_size`, `max_samples`, patience P
Ensure: dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$, feature config ϕ^*

Enumerate all feasible allocations $\mathcal{A}$
Compute cost-uniform weights $w_a \propto |\{a' \in \mathcal{A} : c(a') = c(a)\}|^{-1}$ for each $a \in \mathcal{A}$ ▷ c is integer-linear in $\boldsymbol{\alpha}$, so cost levels are discrete and equality is exact
$\mathcal{D} \leftarrow \emptyset$
// Initial batch
for $i = 1, \ldots,$ `min_samples` **do**
    Sample allocation $a \sim \mathcal{A}$ with weights w; sample random permutation $x \sim a$
    Evaluate $y \leftarrow f(x)$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$
end for
// Feature expansion selection via log marginal likelihood
$\phi^* \leftarrow \arg\max_{(\text{order},r):\, d(\text{order},r) \le 2|\mathcal{D}|} \log p(\mathbf{y} \mid \mathbf{X}(\phi), \theta)$
Fit Bayesian Ridge surrogate θ_{prev} on $\mathcal{D}$ with features ϕ^*
quality_history $\leftarrow [\,]$
// Adaptive loop
while $|\mathcal{D}| <$ `max_samples` **do**
    for $i = 1, \ldots,$ `batch_size` **do**
        Sample $a \sim \mathcal{A}$ with weights w; sample $x \sim a$
        $y \leftarrow f(x)$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$
    end for
    Re-select ϕ^* via log marginal likelihood if d budget allows
    $q \leftarrow \frac{1}{|\mathcal{D}_{\text{new}}|} \sum_{(x,y) \in \mathcal{D}_{\text{new}}} \log p(y \mid x, \theta_{\text{prev}})$
    Append q to *quality_history*; refit θ_{prev} on full $\mathcal{D}$
    if no improvement $> \epsilon$ in last P entries of *quality_history* **then**
        break
    end if
end while
return $\mathcal{D}$, ϕ^*

C.5 Adaptive Exploration Protocol

1. Sample $N_0 = 1000$ diverse feasible placements using cost-weighted allocation sampling.
2. Evaluate all placements against the supernet.
3. Select the expansion order by comparing log marginal likelihoods across all feasible (order, range) combinations (subject to the feature guard below). The configuration with the highest evidence is selected.
4. Refit the Bayesian surrogate at the selected expansion, run risk-bucket refinement, add evaluations, and repeat expansion selection.

The expansion selection is deterministic (no CV splits, no RNG) and is re-run each iteration as the training set grows; if a higher order or longer range gains the most evidence, the surrogate is upgraded.

Feature guard. The feature guard enforces a 2:1 sample-to-feature ratio. At 1000 samples: unary features (196) are within budget; pairwise range-1 features bring the total to 948 (within budget); triplet features (5348 cumulative) are blocked until $\sim$2500 samples are collected.

D Placement Landscape Dynamics

Ablations on 0.5B development model investigating how placement rankings evolve during training.

Algorithm 2 ViterbiDP: exact cost-constrained placement optimisation

Require: MRF potentials θ, feasible allocations $\mathcal{A}_{\text{feas}}$, memory m, top-n per allocation
Ensure: candidate pool $\mathcal{P}$: up to n placements per allocation

$\mathcal{F} \leftarrow \{\mathbf{c}(a) : a \in \mathcal{A}_{\text{feas}}\}$ ▷ feasible count tuples
$c_k^{\max} \leftarrow \max_{a \in \mathcal{A}_{\text{feas}}} a_k$ for each mixer k
Initialise $V[0]$, *back*$[0] \leftarrow \emptyset$
for $j = 0, \ldots, K-1$ **do** ▷ base case: layer 0
  $\mathbf{h} \leftarrow (j)$; $\mathbf{c} \leftarrow \mathbf{e}_j$
  $V[0][(\mathbf{h}, \mathbf{c})] \leftarrow V_0(j)$
end for
for $i = 1, \ldots, N-1$ **do** ▷ standard 1-best forward pass
  for each state $(\mathbf{h}, \mathbf{c})$ with score s in $V[i-1]$ **do**
    for $j = 0, \ldots, K-1$ **do**
      $\mathbf{c}' \leftarrow \mathbf{c} + \mathbf{e}_j$
      if $\mathbf{c}'_k > c_k^{\max}$ for any k **then continue**
      end if
      $\mathbf{h}' \leftarrow$ *last-*m*-of*$(\mathbf{h} \| j)$
      $s' \leftarrow s + V_i(j) + \Delta(\mathbf{h}, j, i)$ ▷ unary + interaction terms
      if $s' > V[i][(\mathbf{h}', \mathbf{c}')]$ **then**
        $V[i][(\mathbf{h}', \mathbf{c}')] \leftarrow s'$; *back*$[i][(\mathbf{h}', \mathbf{c}')] \leftarrow (\mathbf{h}, \mathbf{c})$
      end if
    end for
  end for
end for
// Extract top-n per allocation — no k-best extension needed
// Each distinct terminal history $\mathbf{h}$ with the same counts $\mathbf{c}$
// encodes a unique 1-best path, yielding up to K^m candidates per allocation.
$\mathcal{P} \leftarrow \emptyset$
for each feasible count tuple $\mathbf{c} \in \mathcal{F}$ **do**
  $\mathcal{T} \leftarrow \{(\mathbf{h}, V[N-1][(\mathbf{h}, \mathbf{c})]) : (\mathbf{h}, \mathbf{c}) \in V[N-1]\}$
  Sort $\mathcal{T}$ by score descending; take top n
  for each $(\mathbf{h}, s)$ in top-n of $\mathcal{T}$ **do**
    Backtrack from $(\mathbf{h}, \mathbf{c})$ through *back*$[\cdot]$ to recover placement x
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{x\}$
  end for
end for
return $\mathcal{P}$

E COST MODEL

To enable cost-constrained placement search, we use a linear cost model that predicts inference throughput from the mixer composition of a placement. We model the inverse throughput (latency per token) as a sum of per-layer contributions:

$$\frac{1}{\text{throughput}} = \sum_{m \in \mathcal{M}} c_m \cdot n_m, \tag{3}$$

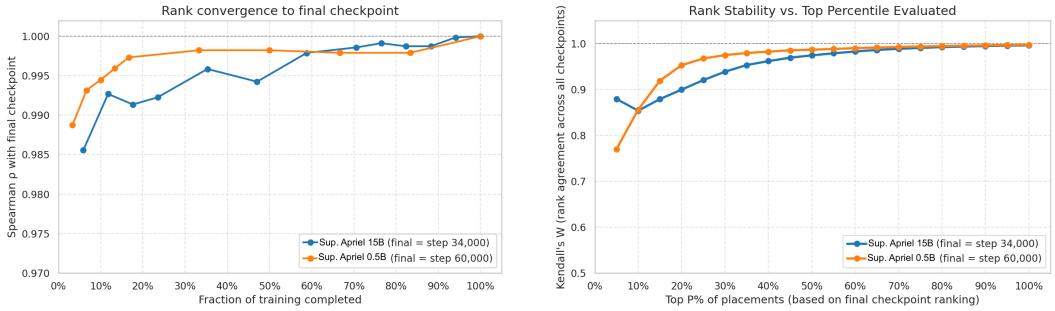
where n_m is the number of layers assigned to mixer type m and c_m is the per-layer cost coefficient. Normalizing by the attention coefficient c_{FA} yields the relative costs reported in Table 6.

Idealized cost. The simplest cost estimate uses pure placements only: $c_X = \frac{\text{throughput}_{\text{FA}}}{\text{throughput}_X}$. This yields relative costs of 0.44 for SWA (w=4096), 0.16 for KDA, and 0.10 for GDN (Table 6, **Idealized***). A measurement with a slightly different vLLM modeling file and SWA window size w=2048 gave lower costs (SWA: 0.16, KDA: 0.11, GDN: 0.09); the SWA difference is largely explained by the halved window size (Table 6, **Idealized**). *We note that the later Idealized model is the one we used to obtain results in our preliminary experiments reported in the main body of this report.*

Algorithm 3 Exploitation: Bayesian surrogate refinement

Require: dataset $\mathcal{D}$, feature config ϕ^*, cost budget C, `num_rounds` R, `evals_per_round` E, top-n candidates per round, risk buckets B
Ensure: best placement x^*

Enumerate $\mathcal{A}_C \leftarrow \{a \in \mathcal{A} : c(a) \le C\}$
Fit Bayesian Ridge surrogate $\boldsymbol{\mu}_n, \mathbf{V}_n$ on $\mathcal{D}$ with ϕ^* ▷ optimise prior precision α via Type-II ML
for round $= 1, \ldots, R$ **do**
    // Generate candidates via Viterbi on posterior mean MRF
    $\mathcal{C} \leftarrow \textsc{ViterbiDP}(\boldsymbol{\mu}_n, \mathcal{A}_C, n)$
    // Compute posterior predictive statistics
    for each $x \in \mathcal{C}$ **do**
        $\mu(x) \leftarrow \boldsymbol{\mu}_n^\top \phi(x)$ ▷ posterior predictive mean
        $\sigma^2(x) \leftarrow \frac{b_n}{a_n}\big(1 + \phi(x)^\top \mathbf{V}_n\, \phi(x)\big)$ ▷ posterior predictive variance
    end for
    // Apportion E slots across buckets (largest-remainder method)
    $k_b \leftarrow \lfloor q_b \cdot E \rfloor$ for each $b \in B$ ▷ floor allocations
    $r_b \leftarrow q_b \cdot E - k_b$ for each $b \in B$ ▷ fractional remainders
    Distribute $E - \sum_b k_b$ extra slots to the buckets with the largest r_b
    // Select candidates bucket by bucket
    $\mathcal{C}_E \leftarrow \emptyset$
    for each bucket $b \in B$ with sign s_b, beta β_b, floor quantile τ_b **do**
        $\mathcal{C}_b \leftarrow \{x \in \mathcal{C} \setminus \mathcal{C}_E : \mu(x) \ge \text{quantile}_{\tau_b}(\mu)\}$
        $\mathcal{C}_E \leftarrow \mathcal{C}_E \cup \text{top-}k_b\big(\mathcal{C}_b,\ \mu(x) + s_b\, \beta_b\, \sigma(x)\big)$
    end for
    // Evaluate and update
    for each $x \in \mathcal{C}_E$ **do**
        $y \leftarrow f(x)$; $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$
    end for
    Refit $\boldsymbol{\mu}_n, \mathbf{V}_n$ on $\mathcal{D}$
end for
return $x^* \leftarrow \arg\max\{y : (x, y) \in \mathcal{D},\ c(x) \le C\}$



(a) Spearman ρ vs. final checkpoint.

(b) Kendall's W by percentile tier.

Figure 8: Rank stability under stochastic distillation. **(a)** Spearman ρ between each intermediate checkpoint and the final checkpoint (15B: step 34K; 0.5B: step 60K). Rankings exceed $\rho > 0.98$ from the earliest checkpoint at both scales. **(b)** Kendall's W (multi-rater concordance across all checkpoints) as a function of the percentile tier. Concordance drops monotonically toward the frontier: top-5% shows W=0.77 (0.5B) and W=0.88 (15B).

Regression fit. We benchmark 123 placements of a 48-layer Super Apriel 1.6 (15B) on a single H100 GPU with 80GB memory. Unless stated otherwise, we use `vLLM v0.14.0rc1` with FlashInfer attention backend, `VLLM_USE_FLASHINFER_SAMPLER`=1,

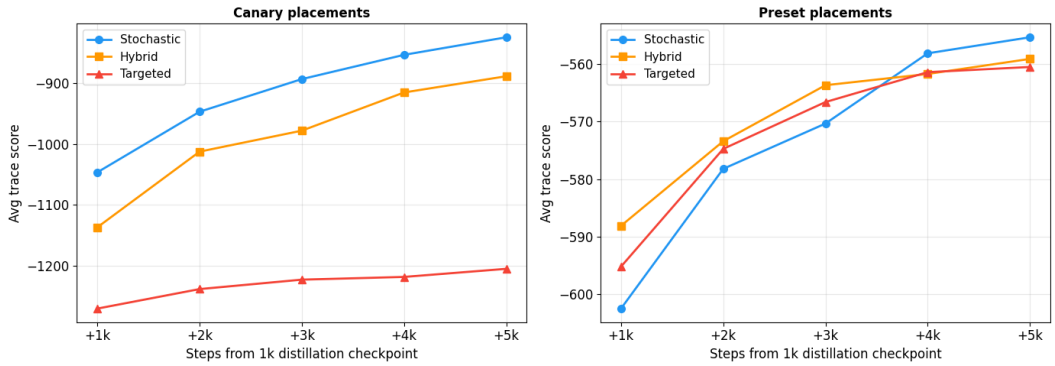


Figure 9: Placement scores during continued distillation of the 0.5B model under three training regimes: stochastic (blue), hybrid (orange), and targeted (red). **Left:** Scores on random canary placements (not included in targeted training). **Right:** Scores on preset placements (the deployment candidates). Stochastic training improves all placements uniformly; targeted training gives presets an initial advantage but stochastic catches up.

Table 6: Relative per-layer cost of each mixer (attention = 1.00), measured on H100 with 15B-parameter Apriel 1.6 hybrids, sequence length 16k. **Idealized*** column — cost for mixer X is simply: $c_X = \frac{\text{throughput}_{\text{att}}}{\text{throughput}_X}$ where both throughputs are measured on pure placements — all 48 layers set to the same mixer type. This assumes each layer contributes independently and equally to total latency, with no cross-mixer overhead (w – window size for SWA); **Idealized** – uses a slightly different vLLM setup and w=2048, details below. **Regression (clean)** = linear fit of $1/\text{throughput}$ on mixer counts from 84 mixed placements (each mixer ≥ 3 layers); **Regression (all)** — same fit on all 123 placements including singleton placements with mixer counts ≤ 3. See below for a discussion of the outlier effect induced by singleton mixer in VLM (unless stated otherwise, we use vLLM v0.14.0rc1).

Mixer	**Idealized** (w=2048)	**Idealized*** (w=4096)	**Regression** (clean, w=4096)	**Regression** (all, w=4096)
FA	1.00	1.00	1.00	1.00
SWA	0.16	0.44	0.48	0.50
KDA	0.11	0.16	0.21	0.36
GDN	0.09	0.10	0.14	0.28
R^2	—	—	0.977	0.576
N	4	4	84	123

`gpu_mem_utilization`=0.8[3], and `FULL_AND_PIECEWISE` CUDA graph mode. Each placement is evaluated on 500 requests with output length 16k tokens and input length of 1 token using `vllm bench throughput`[4] script. The placements span the full range from pure (all 48 layers of a single mixer) to uniformly mixed compositions across all four mixer types (FA, SWA, KDA, GDN). We use `surgery` mode for this benchmarking, meaning that for each run we only load the placement in question to avoid memory waste (as opposed to loading all 4^{48} placements).

We fit the linear model in Eq. 3 via ordinary least squares on $1/\text{throughput}$. When fit on all 123 placements, the model achieves $R^2 = 0.576$ with a mean absolute latency prediction error of 35.9%. This high error is driven by the singleton placements, i.e. placements with a small number of mixers of a single type. Excluding placements where any mixer type has fewer than 3 layers (retaining 84 of 123 placements) yields a substantially better fit: $R^2 = 0.977$, mean error 7.6%, with relative costs of 0.48 (SWA), 0.21 (KDA), and 0.14 (GDN). These regression-based costs are consistently higher

[3]Higher `gpu_mem_utilization` resulted in OOM errors for some placement presumably due to memory fragmentation.

[4]https://docs.vllm.ai/en/v0.14.0/cli/bench/throughput/

than the idealized estimates, reflecting the overhead of mixing heterogeneous mixer types in vLLM's unified KV cache allocator.

Outlier placements. As discussed above, including singleton placements with strong minority token mixers leads to significantly lower R^2 of the linear fit. In Fig. 10 we plot linear fits induced by all tested placements (left) as well as with strong minority placements (aka singletons) excluded (right). To better understand the nature of the outlier placements, we grouped them into three categories and annotated some of them in the left plot in Fig. 10. **This annotation reveals that all outliers are singletons but not all singletons are outliers.** More specifically, we observe that: adding a minority of a slower mixer into a fast majority seem to cause disproportionate overhead (attention or SW into GDN/KDA). Adding a minority of a faster mixer into a slow majority does not seem to have the same effect.

Based on this finding, here we simply restrict candidates from the refinement phase (Section 2.3) to placements that include 0 or at least 3 of each mixer type. This prevents the search from selecting minority-placement candidates, which would not fit the cost model.

We attribute this to the current vLLM design, where the slowest mixer type determines memory allocation (KV cache pages, block tables) and scheduling constraints for the whole model. The per-layer latency of token mixers should in principle combine linearly. Currently, vLLM's hybrid model support (unified page size, recurrent state management, CUDA graph handling) introduces overhead that breaks this assumption for some configurations. *As vLLM's hybrid model support matures — which is likely given the trend toward hybrid architectures — the linear cost model should become more accurate, making placement optimization via simple cost budgets even more accurate.* Unless stated otherwise, in this work we rely on the coefficients obtained either from the idealized model or the regression fit with all singletons excluded (Fig. 10 right).

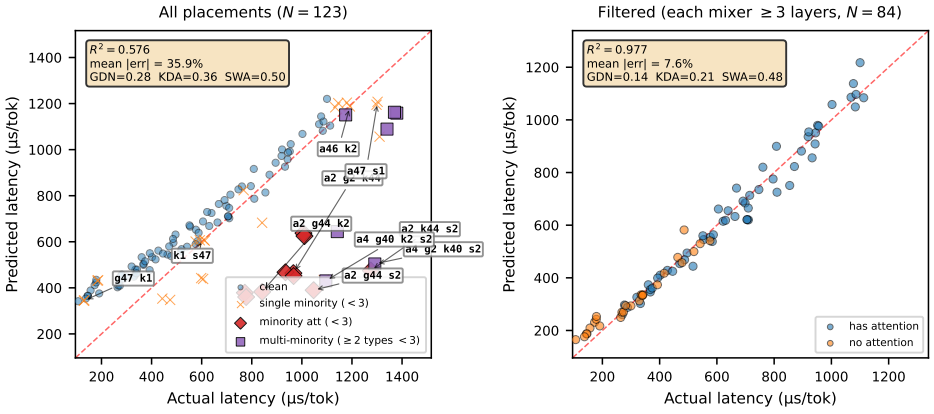


Figure 10: Predicted vs. actual per-token latency on 15B Super Apriel under the linear cost model $1/\text{throughput} = \sum_m c_m \cdot n_m$. **Left:** fit on all 123 placements ($R^2 = 0.576$); we observe large prediction error mostly on layouts with strong minority mixers, see discussion in the main text. **Right:** excluding placements where any mixer has fewer than 3 layers yields a tight fit ($R^2 = 0.977$, mean error 7.6%) on the 84 remaining mixed placements. We use these filtered regression coefficients as the cost model for placement search.

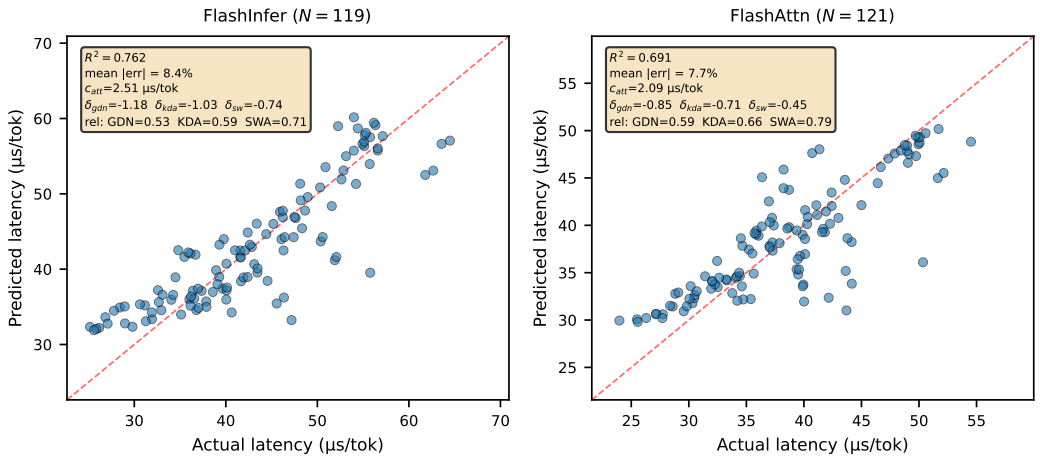


Figure 11: Linear cost model fit for 0.5B supernet. Each point represents a distinct layer placement evaluated at 16k sequence length. The model decomposes per-token latency as $1/\text{throughput} = \sum_i c_i \cdot n_i$, where c_i is the per-layer cost and n_i the layer count for mixer type $i \in$ att, gdn, kda, sw. Coefficients are reported as deltas relative to the attention baseline ($\delta_i = c_i - c_{\text{att}}$) and as relative costs (c_i/c_{att}). **Left**: FlashInfer attention backend. **Right**: FlashAttn attention backend. For 0.5B model we do not exclude outliers and directly use the coefficients induced by the FlashAttn backend.

F Speculative Decoding

The supernet enables speculative decoding (Leviathan et al., 2023) without a separate draft model: use the `all-FA` placement as verifier and any efficient placement as draft. Because draft and verifier share most parameters (FFNs, embeddings, normalization), acceptance rates remain high across placements.

Given generated traces from the target placement p (`all-FA`), one can estimate the acceptance rate of any draft placement q as follows. Given a prefill $\boldsymbol{x}$, recall that the speculative decoding algorithm generates γ tokens from q, denoted by $\boldsymbol{x}_{1:\gamma} = (x_1, \ldots, x_\gamma)$, and verifies them with a single forward pass of p. Each token is accepted with probability $\min(1, p(x_i \mid \boldsymbol{x}, \boldsymbol{x}_{1:i-1})/q(x_i \mid \boldsymbol{x}, \boldsymbol{x}_{1:i-1}))$. If a token is rejected for some position i, we discard tokens i to γ, and generate the final token from the adjusted distribution $\text{norm}\left(\max(0, p(\cdot \mid \boldsymbol{x}, \boldsymbol{x}_{<i}) - q(\cdot \mid \boldsymbol{x}, \boldsymbol{x}_{<i}))\right)$, where norm normalizes the distribution to sum to 1. This procedure comprises a single "step" of speculative decoding. We repeat these steps until we generate an EOS token or reach maximum length.

Let us define the expected step acceptance rate, denoted by $\bar{n}$, as the expected number of tokens generated in a single step of speculative decoding, and let n be the stochastic counterpart. Further, let A_i denote the event where the ith token is accepted. Then

$$\bar{n} = \sum_{i=1}^{\gamma} \mathbb{P}[n \geq i] = \sum_{i=1}^{\gamma} \mathbb{P}\left[\cap_{j=1}^{i} A_j\right].$$

Additionally,

$$\begin{aligned}
\mathbb{P}\left[\cap_{j=1}^{i} A_j\right] &= \mathbb{E}_{\boldsymbol{x}\sim p, \boldsymbol{x}_{1:i}\sim q(\cdot|\boldsymbol{x})}\left[\mathbb{P}\left[\cap_{j=1}^{i} A_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1}\right]\right] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p, \boldsymbol{x}_{1:i}\sim q(\cdot|\boldsymbol{x})}\left[\prod_{j=1}^{i} \min\left(1, \frac{p(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}{q(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p, \boldsymbol{x}_{1:i}\sim p(\cdot|\boldsymbol{x})}\left[\frac{q(\boldsymbol{x}_{1:i} \mid \boldsymbol{x})}{p(\boldsymbol{x}_{1:i} \mid \boldsymbol{x})}\prod_{j=1}^{i} \min\left(1, \frac{p(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}{q(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}\right)\right] \\
&= \mathbb{E}_{\boldsymbol{x}_{1:i}, \boldsymbol{x}\sim p(\cdot|\boldsymbol{x})}\left[\prod_{j=1}^{i} \min\left(\frac{q(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}{p(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}, 1\right)\right].
\end{aligned}$$

Therefore, if for every step we collect

$$\hat{n}(\boldsymbol{x}, \boldsymbol{x}_{1:\gamma}) = \sum_{i=1}^{\gamma}\prod_{j=1}^{i} \min\left(\frac{q(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}{p(x_j \mid \boldsymbol{x}, \boldsymbol{x}_{1:j-1})}, 1\right),$$

then we have an unbiased estimator of $\bar{n}$ since $\bar{n} = \mathbb{E}_{\boldsymbol{x}, \boldsymbol{x}_{1:\gamma}\sim p}[\hat{n}(\boldsymbol{x}, \boldsymbol{x}_{1:\gamma})]$.

Importantly, $\bar{n}$ can be estimated from precomputed target traces without generating new tokens—only log-probabilities under the draft and target are needed. The final estimated acceptance rate is given by $a := \bar{n}/\gamma \in [0, 1]$.

To find the optimal draft placement, we fit a Bayesian linear surrogate to the acceptance rate landscape (analogous to Section 2.3) and search over placements via DP. Fig. 12 shows the resulting speedup landscape averaged over four math benchmarks. The cheapest draft (all-GDN) achieves the highest end-to-end speedup, indicating that acceptance rates remain sufficiently high across the full cost range.

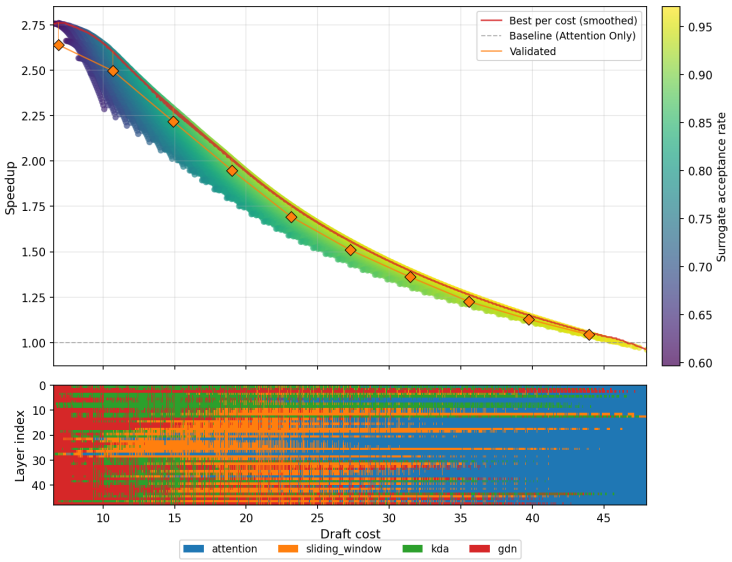
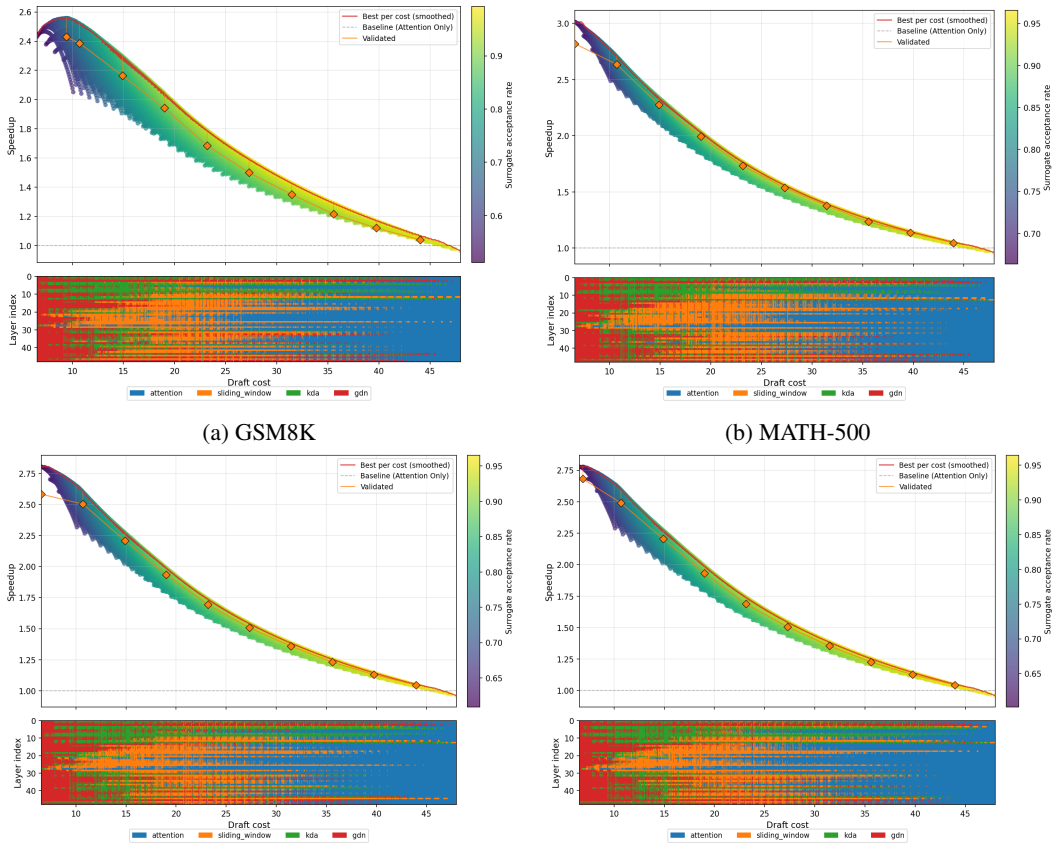


Figure 12: Speculative decoding speedup for the 15B supernet, averaged over four math benchmarks (GSM8K, MATH-500, AIME24, AIME25). Each point represents a candidate draft placement; the x-axis shows draft placement cost and the y-axis shows the net decoding speedup after accounting for the acceptance rate. The cheapest draft (all-GDN) achieves the highest speedup, indicating that acceptance rates remain high across the full cost range.

G INFERENCE

The supernet is served via a vLLM plugin with two serving modes:



(a) GSM8K (b) MATH-500

(c) AIME24 (d) AIME25

Figure 13: Per-task speculative decoding acceptance rate landscapes for the 15B supernet. The target model is the `all-FA` placement; each point is a candidate draft placement scored by the surrogate. The pattern is consistent across tasks: cheaper drafts yield higher speedups despite lower acceptance rates.

Single-preset mode. Only the weights of a single selected mixer placement are loaded and executed using CUDA graphs (`FULL_AND_PIECEWISE`). Inactive mixer weights are entirely offloaded to CPU, so the GPU memory footprint and decode throughput match those of a dedicated single-architecture model. Unless stated otherwise, this mode is used for throughput benchmarking and cost model calibration.

Heterogeneous state. FA and SWA layers maintain standard KV caches (growing with sequence length for FA, bounded at $w = 4096$ for SWA). GDN layers maintain a fixed-size recurrent state $\mathbf{S} \in \mathbb{R}^{h_k \times d_k \times d_v}$ plus a short convolution buffer. KDA layers maintain a similar recurrent state plus three convolution buffers (one per projection stream). A unified page-size architecture pads all per-layer state allocations to a common page size, so that the paged memory manager handles heterogeneous layers transparently.

Prefill and decode. GDN and KDA support dual execution paths. Prefill uses chunkwise parallel algorithms from the FLA (Flash Linear Attention) library for hardware-efficient tensor-core utilization. Decode uses fused recurrent kernels that update the state in-place with incremental convolution updates.

Supernet mode and preset switching. The full supernet is loaded into memory, enabling placement switching at serving time. Switching presets does not require reloading from disk; only the mixer blocks change while shared parameters (FFNs, embeddings, normalization) remain in place. A switch

transfers the required mixer weights between CPU and GPU and recaptures CUDA graphs, typically taking 5–15 s depending on how many layers change mixer type.

A per-request placement routing solution is under development, extending vLLM to let a single loaded checkpoint serve requests with different placements selected on a per-request basis. The scheduler enforces same-placement batching: all requests in a given scheduling round share the same placement, with mismatched requests deferred to the next iteration. During startup, CUDA graphs are pre-captured for allowed placements.

Throughput considerations. Serving multiple placements concurrently requires keeping all registered mixer weights on GPU, reducing KV cache capacity and maximum batch size. Placements with lower mixer overlap across layers yield lower throughput overall as more mixer weights occupy GPU memory.

Supernet-mode throughput. Table 7 reports decode throughput in supernet mode, where more than one placement is kept GPU-resident so that placements can be switched per request. We load two placements at a time—always including `all-FA`—and report absolute throughput together with the speedup relative to `all-FA` served in single-preset mode (926 tok/s). Even with the memory overhead of a second resident placement, serving the efficient preset retains a 1.36–1.80× speedup over single-preset `all-FA`. Consistent with the tradeoff noted above, keeping multiple mixer sets resident reduces KV-cache capacity and batch size, so per-placement throughput in supernet mode is somewhat below the corresponding single-preset figure. These runs use the `FLASH_ATTN` attention backend and same-placement batching; remaining settings follow Table 8.

Table 7: Supernet-mode decode throughput on a single H100 GPU (`TP=1`, `bfloat16`, `load_format=dummy`, prefix caching disabled, `FLASH_ATTN` backend, `FULL_AND_PIECEWISE` CUDA graphs, 1000 prompts, input length 1). Two placements are kept resident at once, always including `all-FA`; throughput is measured when serving the efficient placement, and speedup is relative to `all-FA` in single-preset mode.

Resident placements	Throughput (tok/s)	Speedup vs. single-preset all-FA
all-FA (single-preset)	926	—
all-FA + Idealized|All–18	1660	1.79×
all-FA + Idealized|All–7	1667	1.80×
all-FA + Idealized|All–6	1255	1.36×
all-FA + Idealized|Lklhd–6	1257	1.36×

H THROUGHPUT EVALUATION

Table 8: Throughput evaluation setup for all models. We run all benchmarking on a single H100 GPU. All experiments use `TP=1`, `bfloat16` precision and `load format=dummy` with prefix caching disabled. CUDA graph mode is `FULL_AND_PIECEWISE` and vLLM `v0.14.0 rc.1` unless stated otherwise. Some external models were evaluated with vLLM `v0.18`, which may auto-adjust some settings like cuda-graph mode. Final evaluations use 1000 prompts at 16k sequence length and 500 prompts at 32k sequence length. We use `vllm bench throughput`[5] script for these evaluations. We aim to compare all models in identical setup. We lower the `gpu_mem_utilization` to prevent OOM errors for some models (potentially caused by memory fragmentation in heterogeneous hybrids).

Model	Size	vLLM Version	Attention Backend	GPU Util.	Max Len	Input Len	Output Len	Num Prompts
Supernet Models								
Super Apriel 0.5B (all placements)	0.5B	v0.14.0 rc1	FlashInfer	0.9	65536	1	16000	100
Super Apriel 15B (multiple placements)	15B	v0.14.0 rc1	FlashInfer	0.8	65536	1	16000	500†
						1	16000	1000
						1	32000	500
Baseline Models (Fixed Architecture)								
Apriel-H-1-15B Thinker-SFT	15B	v0.14.0 rc1	FlashInfer	0.8	65536	1	16000	1000
						1	32000	500
OLMo-Hybrid Instruct-SFT	7B	v0.18	auto-select*	0.8	32768	1	16000	1000
						1	32000	500
Qwen3.5-27B	27B	v0.18	auto-select*	0.8	65536	1	16000	1000
						1	32000	500
Falcon-H1R-7B	7B	v0.18	auto-select*	0.8	65536	1	16000	1000
						1	32000	500
NVIDIA Nemotron Nano-12B-v2	12B	v0.18	auto-select*	0.8	65536	1	16000	1000
						1	32000	500
NVIDIA Nemotron 3-Nano-30B	30B	v0.18	auto-select*	0.8	65536	1	16000	1000
						1	32000	500

Limitations. Similar to OLMo-Hybrid (Merrill et al., 2026), we disable CUDA graphs by using `eager-mode` for benchmark evaluation of all linear mixers (both our and external models) to avoid numerical instabilities in current kernel implementations. Throughput measurements, by contrast, were conducted with CUDA graphs enabled. Additionally, for external hybrids we set `--disable-cascade-attn` and `--mamba-ssm-cache-dtype float32`. CUDA graph optimizations for hybrid architectures remain an active area of kernel development, and we expect performance discrepancies between modes to diminish as the ecosystem matures.

Inference engines such as vLLM evolve rapidly, and measured speedups—even accuracy numbers—may depend on the specific implementation and execution mode. Unless stated otherwise, we used `vLLM v0.14.0rc1` for throughput benchmarking and cost model calibration.

Table 9: Distillation hyperparameters for the 15B SUPER APRIEL supernet and the 0.5B development supernet. All runs use AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1), bfloat16 mixed precision, and only mixer weights are trainable.

	Substage 1 (1.5 teacher)		Substage 2 (1.6 teacher)		0.5B Dev
Parameter	**Distill**	**Anneal**	**Distill**	**Anneal**	
Learning rate	10^{-5} (const.)	$10^{-5} \to 0$ (linear)	10^{-5} (const.)	$10^{-5} \to 0$ (linear)	10^{-4} (const.)
Warmup steps	1 000	0	1 000	0	100
Iterations	95 332	5 000	25 090	10 000	60 000
Tokens	187B	10B	49B	20B	∼63B
GPUs (H100)	192	160	120	120	64
Wall-clock (hours)	270	17	100	33	∼11
ZeRO stage	3	3	2	2	3
Seq. length	16 384	16 384	16 384	16 384	8 192
Batch (sequences)	120	120	120	120	128
λ_{act}	0.5	0.5	0.5	0.5	1.0
λ_{RKL}	1.0	1.0	0.9	0.9	0.9
λ_{FKL}	—	—	0.1	0.1	0.1

I TRAINING DETAILS

I.1 DISTILLATION HYPERPARAMETERS

Apriel 1.5 provenance. Supernet development began while Apriel 1.6 was still in training, so Stage 1 distilled from the then-current Apriel 1.5 teacher (Radhakrishna et al., 2025). When 1.6 became available, the trained GDN and KDA mixer weights were transplanted into a new supernet whose shared parameters (FFNs, embeddings, norms, vision encoder) come from the 1.6 checkpoint. Because 1.6 is a post-trained continuation of 1.5 with the same decoder architecture, the transplant required no adaptation and Stage 2 training resumed immediately.

I.2 SFT HYPERPARAMETERS

SFT is initialized from the distillation endpoint `S1: Distil.` and trains with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1), a learning rate of 10^{-5} with 1 000 warmup steps followed by constant schedule, and bfloat16 mixed precision. Each step processes 120 sequences of 32 768 tokens (∼3.9M tokens). Training uses 120 H100 GPUs with ZeRO-2 parallelism and 8-way tensor parallelism, achieving ∼1 690 tokens/s/GPU.

J EVALUATION SUITE

Dev benchmarks We use `LM eval-harness` (Gao et al., 2024) formulations of the following tasks. **MMLU** (Hendrycks et al., 2021a): 57-subject multiple-choice knowledge benchmark (∼14K problems), scored by log-likelihood ranking over answer options. **GSM8K** (Cobbe et al., 2021): grade-school math word problems (1,319 test), generative with exact-match on the final numeric answer. **MATH500** (Hendrycks et al., 2021b): 500-problem subset of competition mathematics, generative. **AIME 2024 / AIME 2025**: competition-level mathematical reasoning from the American Invitational Mathematics Examination, generative. We use `answer-in-text` filter for both AIME and Math500 benchmarks here, since we are interested mostly in the model's ability to preserve reasoning capabilities after distillation, rather than their ability to adhere to a specific output format. We include a more common `evalchemy` (Raoof et al., 2025) version in the unseen set. **FDA** (Arora et al., 2025): zero-shot key–value extraction from FDA drug-label PDFs. **SWDE** (Arora et al., 2025; Lockard et al., 2019): zero-shot relation extraction from semi-structured web pages. **NIAH** (Kamradt, 2023) and **RULER** (Hsieh et al., 2024): synthetic long-context retrieval and reasoning. Specifically, we use the `niah_single_2` (16384 context) and `ruler_qa_squad` (8192 and 16384 context) variants of the RULER benchmark.

[5] https://docs.vllm.ai/en/v0.14.0/cli/bench/throughput/

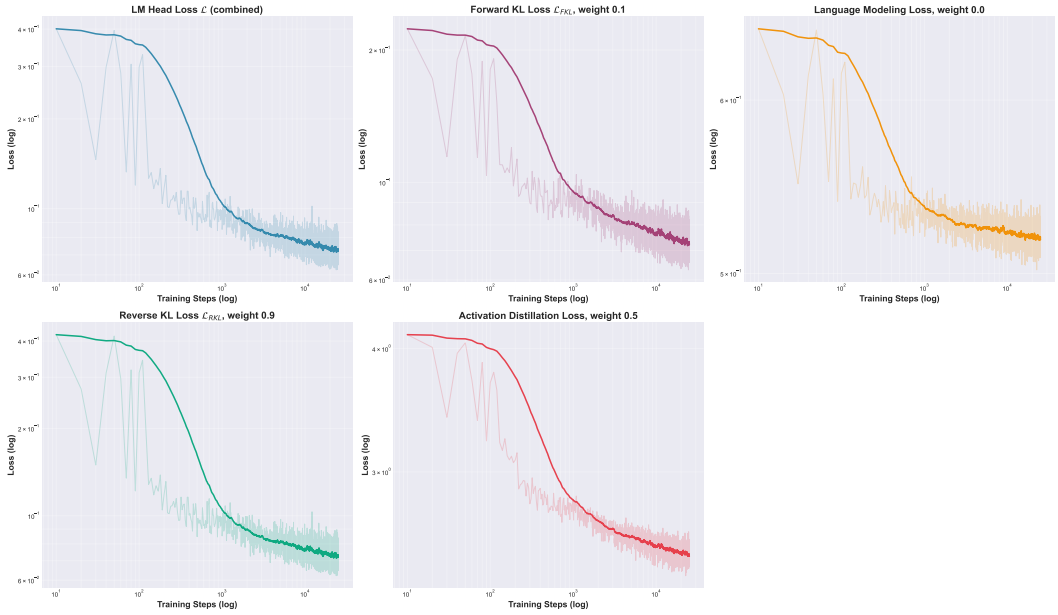


Figure 14: Training losses for stage 1 distillation from Apriel 1.6 (i.e. substage 2) performed for 25k steps with subsequent learning rate annealing phase (not shown here). We use batch size of 120 and sequence length of 16,384 in this stage. Language modeling loss is tracked but not used in training (see weight = 0.0). The losses are plotted without scaling; the scaling coefficient weights λ used during training are in the titles of each plot.

Unseen Benchmarks These benchmarks are only evaluated on final models and are not used for any type of decision-making during training (akin to a held-out test set), ensuring unbiased assessment of generalization. **MMLU-Pro** (Wang et al., 2024b): Enhanced version of MMLU with more challenging questions and increased answer choices (10 options vs 4), reducing guessing probability and requiring deeper reasoning. **GPQA** (Rein et al., 2023): Graduate-level science questions (physics, chemistry, biology) written and validated by domain experts with PhD-level knowledge, testing advanced scientific reasoning. **HLE (Humanity's Last Exam)** (Phan et al., 2026): A comprehensive evaluation benchmark designed to test advanced AI capabilities across extremely challenging problems spanning mathematics, science, and reasoning that push the boundaries of current models. **LCB** (Jain et al., 2024): LiveCodeBench, a contamination-free code generation benchmark with problems collected after model training cutoff dates, ensuring true zero-shot evaluation. **τ^2-Bench** (Barres et al., 2025): A tool-augmented, dual-control benchmark for conversational agents that evaluates multi-step reasoning, coordination, and user-guidance in shared, dynamic environments. **IFEval** (Zhou et al., 2023): Instruction-following evaluation measuring adherence to explicit constraints (format, length, content requirements) in generated responses. We also evaluate on **AIME (NV)** (NVIDIA, 2024) – a version of AIME 2025 problems with Nvidia's extraction protocol using the `evalchemy` filter, which enforces strict output formatting compared to our `answer-in-text` variant used in dev benchmarks. *All evaluations are conducted in a fully generative setting, where models must produce free-form outputs.*

Table 10: Per-benchmark evaluation details.

Benchmark	Scoring	Shots	Filter / Extraction	Notes
Dev Benchmarks				
MMLU	log-liklh	5	—	10 problems/subject
GSM8K	gen	5	`flexible-extract`	greedy decoding
MATH-500	gen	0	`answer-in-text`	greedy decoding
AIME '24	gen	0	`answer-in-text`	greedy decoding
AIME '25	gen	0	`answer-in-text`	greedy decoding
FDA	gen	5	exact-match	greedy decoding
SWDE	gen	5	exact-match	greedy decoding
NIAH	gen	0	exact-match	16k context, greedy decoding
RULER-QA	gen	0	exact-match	8k & 16k context, greedy decoding
Unseen Benchmarks				
MMLU-Pro	gen	5	exact-match	—
GPQA Diamond	gen	0	exact-match	—
HLE	gen	0	exact-match	—
LCB	gen	0	pass@1	3 seeds average
τ^2-Bench	gen	—	GPT-4 user simulator	3 seeds average
IFEval	gen	0	constraint adherence	3 seeds average
AIME (NV)	gen	0	NeMo-Skills (NVIDIA, 2024)	AIME25 with `evalchemy` prompt, 10 seeds average