TRAIN ONCE, REWIRE LATER: EXECUTION-GRAPH POLYMORPHIC TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer checkpoints are tied to the *execution graph* used during training. Compilers can optimize how a fixed graph is executed, but changing the graph itself typically requires training a new model. Even a minimal rewrite exposes this dependence: executing attention and the feed-forward layer in parallel raises the language-modeling loss of a 120M transformer from 1.137 to 1.681 bits per byte (BPB), despite preserving the same parameters and operators. We call this sensitivity to layerwise execution changes *execution-graph rigidity* and show that it is learned rather than architectural. We introduce Execution-graph Polymorphic Transformers (EPTs), which instead train a single set of shared weights across a reference execution graph and alternative layerwise programs, with a consistency objective that aligns their predictions across executions. For the controlled sequential-to-parallel rewrite, the all-parallel program costs at most 0.002 BPB from 1B to 7B; after 10B training tokens, downstream accuracy differs by at most 0.54 percentage points across four tasks. We then extend the same objective to six modes per layer: sequential, parallel, reverse, attention only, feed-forward network only, and skip. Measurement-guided search on one 1B checkpoint finds different programs for Blackwell, H100, and L40S, reaching 1.709×, 1.568×, and 1.697× prefill speedup. Programs selected under a +0.05 BPB search-window budget cost 0.050–0.052 BPB on disjoint validation contexts. EPTs therefore turn the execution graph from a fixed training choice into a budgeted compiler decision, allowing one training run to support different hardware deployment strategies.

1 INTRODUCTION

A Transformer checkpoint stores learned parameters, but inference also requires an *execution graph*, which specifies which attention and FFN operators are executed in each layer, the order in which they run, and whether one consumes the other's output. Current methods bind each checkpoint to one such execution graph. Deep learning compilers improve kernels, schedules, and parallelism, or apply transformations that preserve model semantics, but they ordinarily retain the operator relationships for which the weights were trained (Chen et al., 2018; Jia et al., 2019a; Yang et al., 2021; Unger et al., 2022; Shi et al., 2023). Hardware oriented architectures instead redesign these relationships to improve efficiency on particular systems, but the execution graph must be chosen before training and each design requires its own checkpoint (Wang & Komatsuzaki, 2021; Chowdhery et al., 2023; Prabhakar et al., 2024; Zhang et al., 2025). Adapting a model to new hardware therefore presents an undesirable choice: retain the trained execution graph and limit optimization to its implementation, or adopt a hardware oriented execution graph and train a new model. What is missing is a framework that preserves the predictive performance learned during training while allowing the execution graph to be selected at deployment for hardware specific inference.

The obstacle, however, is not simply the lack of such a deployment framework. Standard training makes the weights themselves dependent on their original execution graph. A 120M specialist trained with sequential blocks obtains 1.137 BPB under its original execution graph, but switching every block to parallel execution raises BPB to 1.681 and reduces matched-context agreement to 37.1%, even though all parameters and operators are retained. The same sensitivity appears for richer

Code and artifacts: `https://anonymous.4open.science/r/EPTs-67C5/`

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.

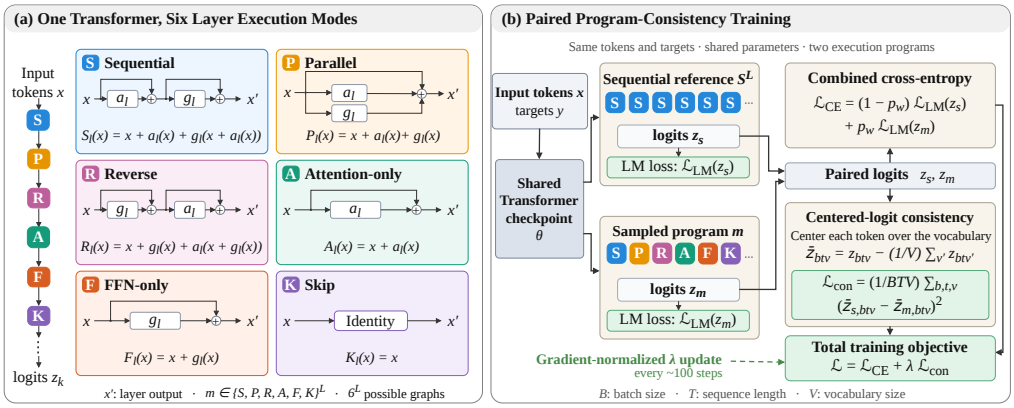

Figure 1: **Execution modes and program-consistency training.** **(a)** The six supported layer execution modes. **(b)** The same tokens are processed by an all-sequential reference and a sampled program. Language-model loss trains both paths, while centered-logit consistency aligns their predictions. The controller shown adjusts the consistency weight from gradient magnitudes.

execution graphs. Applying the programs selected for Blackwell, H100, and L40S to an ordinary 1B sequential specialist increases BPB by 1.369, 0.369, and 1.777, respectively. These failures show that weights specialize not only to the training data and objective, but also to the execution graph used during training. We refer to this learned dependence as *execution-graph rigidity*.

We ask whether this rigidity can instead be trained away, so that the execution graph becomes a deployment-time choice. Shared-weight elastic networks have previously exposed width, depth, or nested capacity as runtime choices (Yu et al., 2019; Yu & Huang, 2019; Cai et al., 2020; Hou et al., 2020; Devvrit et al., 2024; Cai et al., 2024); these methods vary how much computation a checkpoint performs. We additionally vary how the retained operators depend on one another. Our key object is a *family* of layerwise execution programs, where a program assigns one execution mode to each layer. We begin with sequential and parallel Attention–FFN blocks, add identity skipping, and finally expose six modes per layer: sequential, parallel, reverse ordering, attention only, FFN only, and skip. One checkpoint then supports up to 6^L programs, spanning dependency rewrites, removal of one branch, and removal of the complete block, all without changing the stored parameters.

Figure 1 illustrates the EPT program space and paired training objective. The same tokens are processed by an all-sequential reference and a sampled program. Both receive language-model supervision, while centered-logit consistency aligns their predictions; its weight controls the tradeoff between reference quality and robustness across programs. After training, single-layer probes and sampled programs estimate program costs relative to the sequential reference. At deployment, these estimates and target-hardware latency profiles guide a search under a specified quality budget, with direct BPB and latency measurements determining the final program.

Our contributions are as follows.

- We identify *execution-graph rigidity*: ordinary checkpoints are specialized to the graph they are trained with and degrade sharply under alternative graphs. It holds for both sequential and parallel specialists and extends from one dependency rewrite to six-mode programs.
- We introduce *EPT*s and show in matched controls that multi-program training removes most binary rewrite cost, while consistency raises agreement from 81.25% to 90.53% and halves ternary program cost. Binary robustness persists through 7B, 10B training tokens, and downstream evaluation.
- Extending EPTs to six modes enables hardware-specific deployment. One 1B checkpoint reaches 1.57–1.71× prefill speedup on three GPUs near +0.05 BPB, using a different measured program on each. The same programs cause much larger quality degradation when applied to a standard sequential checkpoint, showing that their success depends on the graph robustness learned during EPT training.

2 LEARNING EXECUTION-PROGRAM POLYMORPHISM

In this paper we treat execution as a deployment variable. An *execution mode* is one realization of a Transformer layer; an *execution program* specifies one mode for each layer; and executing that program instantiates its *execution graph*. We train alternative programs against a common sequential reference so that one set of weights remains predictive as the program changes.

2.1 AN EXECUTION-PROGRAM LANGUAGE

Let $a_l(x) = A_l(N_l(x))$ and $g_l(x) = F_l(N_l(x))$ denote the attention and FFN updates in layer l of a transformer. We use the following 6 modes for our programs:

$$\begin{aligned} \text{Sequential:} \quad & \mathrm{S}_l(x) = x + a_l(x) + g_l(x + a_l(x)), \\ \text{Parallel:} \quad & \mathrm{P}_l(x) = x + a_l(x) + g_l(x), \\ \text{Reverse:} \quad & \mathrm{R}_l(x) = x + g_l(x) + a_l(x + g_l(x)), \\ \text{Attn. Only:} \quad & \mathrm{A}_l(x) = x + a_l(x), \\ \text{FFN Only:} \quad & \mathrm{F}_l(x) = x + g_l(x), \\ \text{Skip:} \quad & \mathrm{K}_l(x) = x. \end{aligned} \tag{1}$$

Here N_l is the same pre-normalization operation for both branches: sequential and reverse renormalize after their first residual update, whereas parallel applies attention and the FFN to the same normalized input. The first pair is particularly important: sequential and parallel execution use exactly the same learned parameters and operators, and differ only in whether the FFN depends on the current attention update. Fixed parallel Attention–FFN blocks have appeared in GPT-J and PaLM (Wang & Komatsuzaki, 2021; Chowdhery et al., 2023); here they form one choice inside a shared-weight program family. Sequential and parallel therefore isolate execution-graph sensitivity from pruning or capacity reduction. Reverse changes operator order; attention-only and FFN-only remove one branch; and skip removes the corresponding layer from the graph, connecting the broader family to structured depth reduction (Fan et al., 2020; Elhoushi et al., 2024).

2.2 PAIRED PROGRAM-CONSISTENCY TRAINING

For a batch of B sequences of length T and vocabulary size V, let z_s and z_m denote logits from the sequential reference and sampled program m. For target tokens y, the next-token loss is

$$\mathcal{L}_{\mathrm{LM}}(z) = -\frac{1}{BT} \sum_{b,t} \log \mathrm{softmax}(z_{bt})_{y_{bt}} \tag{2}$$

and both paths receive language-model supervision:

$$\mathcal{L}_{\mathrm{CE}}(z_s, z_m) = (1 - p_w)\, \mathcal{L}_{\mathrm{LM}}(z_s) + p_w\, \mathcal{L}_{\mathrm{LM}}(z_m), \tag{3}$$

with $p_w = 0.5$ in all headline experiments.

To align their predictions on the same teacher-forced contexts, we center each token's logits over the vocabulary, $\bar{z}_{btv} = z_{btv} - V^{-1} \sum_{v'} z_{btv'}$, and minimize

$$\mathcal{L}_{\mathrm{con}}(z_s, z_m) = \frac{1}{BTV} \sum_{b,t,v} (\bar{z}_{s,btv} - \bar{z}_{m,btv})^2. \tag{4}$$

Centering removes shifts that leave softmax probabilities unchanged. The full objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}}(z_s, z_m) + \lambda\, \mathcal{L}_{\mathrm{con}}(z_s, z_m), \tag{5}$$

where λ is fixed or updated periodically so that the consistency gradient norm remains a fixed fraction of the language-model gradient norm (Appendix E). R-Drop and in-place distillation also align stochastic or shared-weight subnetworks (Liang et al., 2021; Yu & Huang, 2019); EPT instead aligns explicit execution programs, supervises both with language-model loss, and shares the reference weights. The headline six-mode checkpoints backpropagate the consistency loss through both z_s and z_m. The memory-efficient 430M binary checkpoint trained for 10B tokens stops the consistency gradient through z_s while retaining language-model gradients on both paths.

2.3 MEASURING PROGRAM QUALITY

Let $\mathrm{BPB}_\theta(m)$ denote the bits per byte obtained by checkpoint θ under program m. Bits per byte is the negative log-likelihood normalized by the number of input bytes, so lower values indicate better language-modeling quality and remain comparable across tokenizations. We use the all-sequential program S^L as the reference for each checkpoint. The program cost of executing program m is

$$\Delta\mathrm{BPB}_\theta(m) := \underbrace{\mathrm{BPB}_\theta(m)}_{\text{program } m} - \underbrace{\mathrm{BPB}_\theta(S^L)}_{\text{sequential reference}} . \tag{6}$$

This is a within-checkpoint comparison: the parameters are fixed and only the execution program changes. A value of zero indicates that m matches the reference quality, a positive value indicates degradation, and a negative value indicates that m improves BPB on the evaluation data. We report $\mathrm{BPB}_\theta(S^L)$ alongside this quantity because a small program cost does not by itself imply strong absolute model quality. When the checkpoint is clear from context, we omit the subscript θ and write $\mathrm{BPB}(S^L)$ and $\Delta\mathrm{BPB}(m)$.

We measure *matched-context agreement* as the fraction of teacher-forced token positions at which program m and the reference program S^L choose the same next-token argmax. Both programs receive the same input sequence, so this metric measures predictive agreement under matched contexts rather than agreement between free-running generations. We use "agreement" as shorthand for matched-context agreement after this definition.

We separately measure the *specialist gap* associated with training for execution-program polymorphism. Let θ_{EPT} be an EPT checkpoint and θ_{spec} a sequential specialist trained with the same architecture, training data, and token budget. The specialist gap is

$$\Delta_{\mathrm{spec}} := \underbrace{\mathrm{BPB}_{\theta_{\mathrm{EPT}}}(S^L)}_{\text{EPT checkpoint}} - \underbrace{\mathrm{BPB}_{\theta_{\mathrm{spec}}}(S^L)}_{\text{sequential specialist}} . \tag{7}$$

Unlike Eq. 6, this is a between-checkpoint comparison under the same execution program. A positive Δ_{spec} means that the EPT checkpoint has worse reference BPB than the specialist.

We report these metrics together because no single one establishes equivalence. Downstream tasks provide an additional behavioral check. Our evaluation separates native quality, program robustness, behavioral alignment, and deployment utility using reference BPB, $\Delta\mathrm{BPB}$, matched-context agreement, Δ_{spec}, downstream accuracy, and measured latency. We use measured $\Delta\mathrm{BPB}$ as the compiler feasibility criterion. Appendix H gives the family-specific cost models, and Appendix F derives the binary local defect.

2.4 DEPLOYMENT-TIME PROGRAM SELECTION

A hardware profiler measures each layer–mode choice on the target system and workload. We optimize measured latency, not FLOPs (Tan et al., 2019; Cai et al., 2019; Wang et al., 2020). The corresponding surrogate problem maximizes predicted latency benefit under a predicted program-cost budget:

$$\max_{m \in \mathcal{M}^L} \widehat{V}_{h,w}(m) \quad \text{s.t.} \quad \widehat{\Delta\mathrm{BPB}}(m) \le \varepsilon. \tag{8}$$

For six-mode programs, $\widehat{V}_{h,w}(m) = \sum_l v_{l,m_l}(h,w)$ and $\widehat{\Delta\mathrm{BPB}}(m) = \sum_l c_{l,m_l}$. Dynamic programming solves this discretized multiple-choice knapsack and produces candidate programs. Because the additive estimates omit whole-program interactions, the headline results use measurement-in-the-loop search: every acceptance decision and feasibility claim uses measured $\Delta\mathrm{BPB}(m)$ and latency. FlexFlow and Alpa search execution plans for a fixed model graph (Jia et al., 2019b; Zheng et al., 2022); here the program itself is also a deployment choice.

3 EXPERIMENTAL SETUP

3.1 MODELS AND TRAINING

We experiment with the following *mode-subsets*

$$\{\mathrm{S},\mathrm{P}\}^L \subset \{\mathrm{S},\mathrm{P},\mathrm{K}\}^L \subset \{\mathrm{S},\mathrm{P},\mathrm{R},\mathrm{A},\mathrm{F},\mathrm{K}\}^L. \tag{9}$$

The binary family changes one dependency without changing operators or capacity; the ternary family adds complete-block removal; and the six-mode family adds ordering changes and partial blocks (Appendix A, Table 4).

All checkpoints are trained from scratch on ClimbMix with an 8,192-token BPE vocabulary and evaluated on fixed ClimbMix validation contexts. The primary architecture uses RMS normalization, ReLU^2 FFNs, QK normalization, rotary embeddings, and alternating value embeddings. Binary experiments extend to 7B, ternary experiments to 3B, and six-mode experiments to 1B. We also train Llama-style six-mode checkpoints at 120M, 430M, and 1B. Appendix C gives checkpoint-specific weighting policies and provenance.

3.2 PROGRAM SAMPLING AND EVALUATION

The all-sequential program S^L is the reference in every family. Table 1 summarizes how the other programs are used during training and held-out evaluation.

Table 1: Training and held-out evaluation protocols. Modes are sampled independently across layers except in binary training, which evaluates both S^L and P^L on every step.

Family	Training protocol	Held-out evaluation protocol
Binary	Jointly train S^L and P^L, with $p_w = 0.5$.	Stratify mixed sequential/parallel programs by parallel-layer count $0, \ldots, L$.
Ternary	Sample $(\mathrm{S},\mathrm{P},\mathrm{K})$ with probabilities $(.5, .4, .1)$.	Sample 5,000 programs with probabilities $(.4, .4, .2)$.
Six-mode, primary (fixed weight)	Sample all six modes approximately uniformly.	Sample 3,000 programs uniformly over all six modes.
Six-mode, primary/Llama (GradNorm)	Sample $(\mathrm{S},\mathrm{P},\mathrm{R},\mathrm{A},\mathrm{F},\mathrm{K})$ with probabilities $(.30, .30, .15, .075, .075, .10)$.	Sample 3,000 programs uniformly over all six modes.

Held-out programs are sampled after training and evaluated on fixed, teacher-forced ClimbMix contexts. Because the families contain different modes and use different evaluation distributions, their aggregate metrics are interpreted within family rather than as a cross-family ranking. Predictor splits remain documented in Appendix B.

3.3 HARDWARE AND RUNTIME MEASUREMENT

Compiler results compare a program m with the all-sequential program S^L using the same EPT checkpoint: speedup is $\mathrm{Latency}(S^L)/\,\mathrm{Latency}(m)$, and directly measured $\Delta\mathrm{BPB}(m)$ determines feasibility. Specialist comparisons instead measure whether ordinary weights tolerate the same program; they are not latency baselines. Agreement is diagnostic and does not enter candidate acceptance.

All modes use shared attention and FFN modules in the same PyTorch stack, without mode-specific custom kernels. Headline latencies use a fixed batch of 8 sequences of length 1,024 in BF16, without KV caching, and therefore measure prefill-style execution. Appendix Table 10 reports pure-mode timing, and Appendix D reports training and search costs.

4 RESULTS: LEARNING AND DEPLOYING EXECUTION PROGRAMS

4.1 EXECUTION-GRAPH RIGIDITY IS LEARNED—AND CAN BE BROKEN

Changing one dependency breaks an ordinary Transformer. An ordinary 120M sequential specialist obtains 1.137 BPB natively but 1.681 BPB and only 37.1% agreement after an all-parallel rewrite. An ordinary specialist is therefore tied to its trained dependency graph even when the alternative graph uses the same operators and parameters.

The same operators can learn multiple dependency graphs. The binary family isolates consistency cleanly: sequential and parallel use the same operators and parameters, and the matched 120M runs differ only in the consistency term. Larger and longer-trained checkpoints test whether the resulting robustness persists with scale and training (Table 2).

Table 2: Binary evidence under the all-parallel rewrite. Every row retains all learned operators. The 3B and 7B checkpoints are short runs testing whether rewrite tolerance persists with scale, not absolute language-model scaling.

Scale	Checkpoint	Tokens	Consistency policy	Reference $\mathrm{BPB}(S^L)$	Program cost $\Delta\mathrm{BPB}(P^L)$	Matched-context agreement
120M	Sequential specialist	57M	none	1.1371	0.5437	37.11%
120M	Joint LM	57M	$\lambda = 0$	1.196	0.003	81.25%
120M	Paired consistency	57M	fixed $\lambda = 1$	1.195	0.001	90.53%
430M	Paired consistency	10B	fixed $\lambda = .1$, stop-gradient	0.8225	0.00019	92.24%
1B	Paired consistency	197M	fixed $\lambda = 1$	1.0470	0.00120	96.14%
3B	Paired consistency	98M	GradNorm, $\rho = .2$	1.1639	0.00124	98.10%
7B	Paired consistency	98M	fixed $\lambda = .1$, stop-gradient	1.2875	0.00174	95.56%

Joint language-model training supplies most of the binary BPB robustness: its all-parallel cost is already 0.003 BPB. Consistency changes this to 0.001 BPB and raises agreement from 81.25% to 90.53%. The larger rows test persistence, not a scaling law for native quality; their token budgets and policies differ. From 1B to 7B, the all-parallel program costs 0.00120–0.00174 BPB and retains 95.56–98.10% agreement. After 10B training tokens, the 430M checkpoint reaches 0.8225 BPB sequentially and 0.8227 in parallel: $\Delta\mathrm{BPB}(P^L) = +0.00019$ with 92.24% agreement.

Table 3: Downstream results for the 430M binary checkpoint after 10B training tokens. HellaSwag, PIQA, and ARC-Easy report normalized accuracy; WinoGrande reports accuracy. Differences are $P^L - S^L$ in percentage points.

Benchmark	S^L (%)	P^L (%)	Difference (pp)
HellaSwag	41.93	41.39	−0.54
PIQA	70.40	70.24	−0.16
WinoGrande	50.67	50.91	+0.24
ARC-Easy	54.38	54.50	+0.12

Neither program is consistently better across these tasks (Table 3). We treat the single-run differences as downstream parity evidence, not a statistical equivalence test. Additional binary evidence appears in Appendix M.2.

4.2 FROM DEPENDENCY REWRITES TO SIX-MODE PROGRAMS

The same objective extends to block removal. The ternary family adds a skip mode without changing the training objective. In a compute-matched 430M comparison, adding consistency reduces mean ΔBPB from 0.0328 to 0.016. Across 120M–3B checkpoints, ternary training reduces mean ΔBPB by 13–16$\times$ relative to binary-trained controls. The complete controls and sampling protocol appear in Appendix M.1.

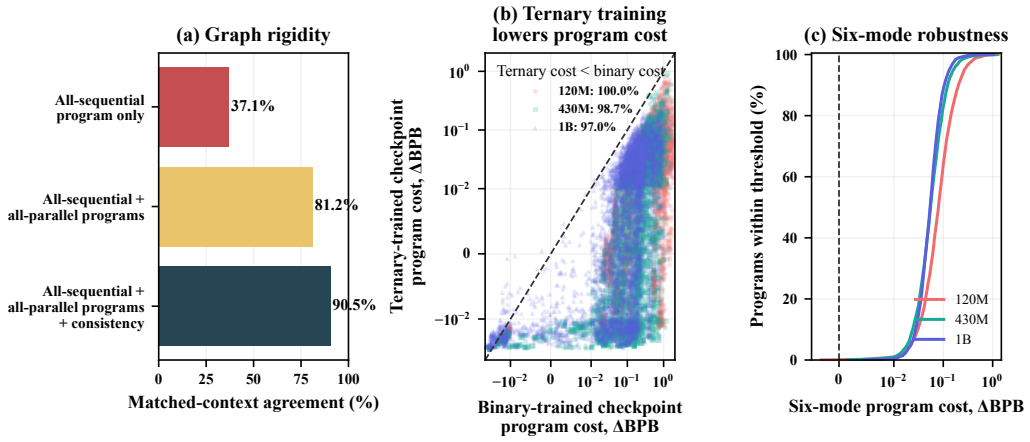

Figure 2: **Controlled evidence across the three program families.** **(a)** Binary language-model training improves agreement over an ordinary specialist, and consistency improves it further. **(b)** Paired evaluation of 5,000 ternary programs at 120M, 430M, and 1B. **(c)** Empirical CDFs under 3,000 uniformly sampled programs for the primary six-mode checkpoints. The families use different sampling distributions, so their raw costs are not directly comparable.

Six modes combine rewrites and removal. We next test one program family that combines dependency rewrites, operator reordering, partial blocks, and complete-block removal. An L-layer model has 6^L possible programs. Under 3,000 uniformly sampled programs, the 1B checkpoint has reference BPB 1.1177, mean $\Delta\text{BPB} = 0.0630$, 95th-percentile cost 0.1318, and a specialist gap of $\Delta_{\text{spec}} = 0.0367$ BPB. This is a severe stress test: on average, five of every six layers use a non-sequential mode. In a matched 500-program control, consistency reduces mean ΔBPB from 0.0711 to 0.0648, while agreement changes only from 76.11% to 76.31%. This is a modest cost improvement at this six-mode operating point. The larger causal effects are the binary agreement gain and the two-fold ternary cost reduction. The full three-scale results and cross-architecture analysis appear in Appendix Table 16 and Appendix K.

Figure 2 summarizes the progression: the controlled binary comparison isolates learned dependency robustness, ternary programs add complete-block removal, and six-mode programs combine rewrites with partial and complete removal.

Program costs retain useful structure. Figure 3 separates low-cost dependency rewrites from more consequential operator removal. Parallel and reverse are nearly neutral at the single-layer level, whereas attention-only, FFN-only, and skip have larger, checkpoint-dependent costs. Predictability is strongest for the binary family ($r = 0.971$); for six-mode programs, the controlled additive model reaches $r = 0.573$–0.721, while the richer archive surrogate reaches $r = 0.768$–0.888. These correlations guide candidate generation but do not certify feasibility, so selected programs are still validated by direct quality measurement.

4.3 Execution program becomes a deployment variable

Once a checkpoint supports many programs, deployment requires selecting one for the target hardware. Local cost models propose candidates, but direct quality and latency measurements determine every reported result; the prediction analysis appears in Appendix H.

Compiled programs expose a quality–latency frontier. Independent searches on the same 1B checkpoint select a different program for each GPU, reaching 1.57–1.71× speedup (Figure 4; Appendix Table 8). The programs satisfy the $+0.05$ ΔBPB constraint on the search windows; on disjoint validation windows they cost 0.0503–0.0516 BPB, slightly above the nominal threshold. Appendix M.4 shows the measured per-platform speed–quality landscapes, and Appendix I reports additional Blackwell points across scales.

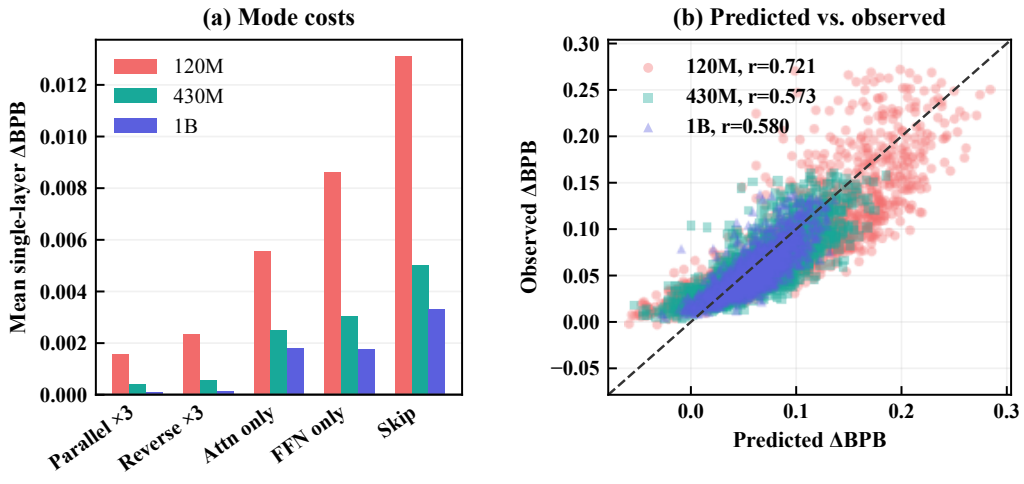

Figure 3: **Program costs retain useful structure.** **(a)** Mean single-layer costs separate dependency rewrites from operator-removal modes; parallel and reverse are multiplied by three for visibility. **(b)** Predicted versus observed ΔBPB for the controlled additive predictor on a disjoint test set.

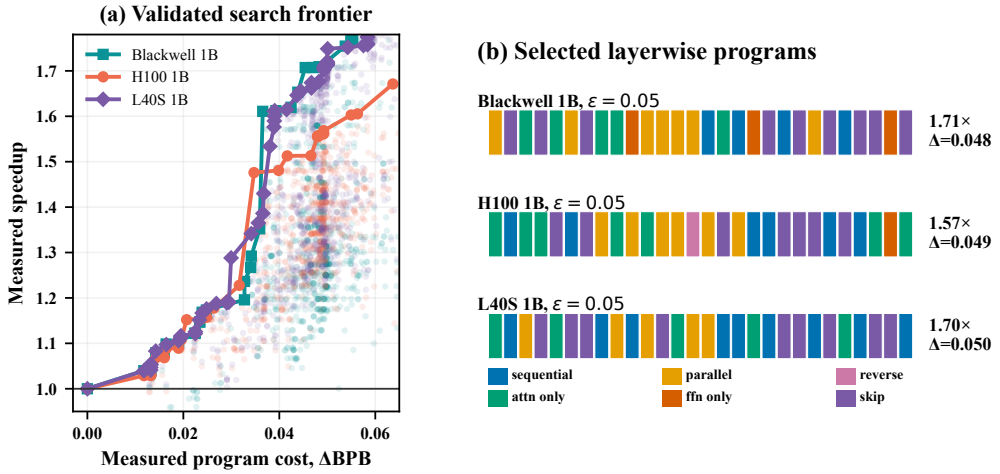

Figure 4: **Measured six-mode program selection on three GPUs.** **(a)** Quality–latency frontiers from measurement-in-the-loop search using the same 1B checkpoint; speedup is relative to each platform's all-sequential program. **(b)** The fastest measured program satisfying $\varepsilon = 0.05$ on the search windows. Program cost is relative to the checkpoint's all-sequential program.

The selected programs add about 0.05 BPB on the EPT but 1.3690, 0.3685, and 1.7767 BPB when forced onto independently trained sequential weights. These within-checkpoint costs isolate the EPT's learned tolerance; they are not absolute quality gaps between checkpoints.

At a nearby-latency Blackwell point, the compiled 430M EPT is 3.7% faster than a separately trained 316M dense specialist (34.99 versus 36.29 ms) but 0.0181 BPB worse. This single comparison illustrates EPT's amortization across hardware targets, not superiority to a model trained for one operating point.

Selected programs combine compute reduction and dependency rewrites. In matched 430M searches, restricting the program family to $\{\mathrm{S}, \mathrm{P}, \mathrm{R}\}$ yields only 1.003× speedup, confirming that skip and attention-only provide most standalone latency reduction. Yet the full six-mode family reaches 1.669×, versus 1.602× for $\{\mathrm{S}, \mathrm{A}, \mathrm{K}\}$, and every hardware-selected 1B program uses five to

seven parallel layers. Dependency rewrites are therefore complementary degrees of freedom in the mixed-program frontier rather than standalone speed levers (Appendix Tables 8–9).

5 Related work

Stochastic depth and LayerDrop train residual networks to tolerate omitted blocks, while Once-for-All, MatFormer, and Flextron amortize training across nested deployment configurations (Huang et al., 2016; Fan et al., 2020; Cai et al., 2020; Devvrit et al., 2024; Cai et al., 2024). CALM, LayerSkip, and Mixture-of-Depths adapt computation to an input or token (Schuster et al., 2022; Elhoushi et al., 2024; Raposo et al., 2024). EPT instead selects a static program for a target hardware workload before inference and treats depth as one choice inside a larger family that also changes dependencies and retained branches.

GPT-J and PaLM train fixed parallel Attention–FFN blocks (Wang & Komatsuzaki, 2021; Chowdhery et al., 2023). PAF, Simplifying Transformer Blocks, FAL, Kraken, and Ladder-Residual redesign dependencies for efficiency or communication overlap (Sonkar & Baraniuk, 2023; He & Hofmann, 2024; Kim et al., 2025; Prabhakar et al., 2024; Zhang et al., 2025). Puzzle searches distillation-derived architectures for a target hardware budget and emits a separate checkpoint per target (Bercovich et al., 2025). These methods select one architecture before training or adaptation. We train shared weights over a layerwise family and defer selection to deployment.

Feed-Forward Steering analyzes post-training sensitivity to sequential-to-parallel interventions (Mudarisov et al., 2026), and Transformer Layers as Painters finds partial post-hoc robustness to skipping, reordering, and parallelizing selected middle layers in frozen models (Sun et al., 2025). Our all-parallel control instead rewrites every layer, reducing an ordinary specialist to 37.1% agreement; we then study learned alignment, multi-layer composition, and compilation. The compiler complements quantization, KV-cache optimization, batching, kernel generation, and model parallelism rather than replacing them (Pope et al., 2023; Shoeybi et al., 2019; Kwon et al., 2023).

6 Limitations

The six-mode study reaches 1B parameters, and the headline checkpoint is trained on 98M tokens. We do not test whether six-mode robustness and the specialist gap persist after long-duration training. All checkpoints use ClimbMix, an 8,192-token vocabulary, and random initialization; other domains, larger vocabularies, and adaptation of pretrained checkpoints remain untested.

The reported speedups measure BF16 prefill-style execution on three GPUs. Cached autoregressive decoding, throughput, memory, and KV-cache compatibility remain open. The dense-specialist comparison covers one nearby-latency point, not a full frontier across scales and hardware targets. Uniform six-mode sampling is deliberately harsher than the deployment distribution and should not be read as representative serving behavior.

Cost predictors become less calibrated for aggressive correlated programs and do not reduce measurement need in our matched-budget test (Appendix J). We therefore use them only to propose candidates and directly measure all reported programs; Appendix L summarizes these reporting principles. Matched-context agreement is diagnostic, not a guarantee of identical free-running generation; downstream and generation evaluation of compiled six-mode programs remains future work.

7 Conclusion

A checkpoint need not be tied to one execution graph. Binary experiments show that EPT training preserves a dependency rewrite from 120M to 7B and after 10B training tokens; ternary experiments extend the result to block removal (Appendix M). With six modes per layer, one 1B checkpoint supports different measured deployment programs on Blackwell, H100, and L40S, reaching 1.57–1.71$\times$ prefill speedups near 0.05 BPB. Execution programs are an additional inference-optimization axis compatible with efficient attention, quantization, batching, KV-cache management, and model parallelism, while long-duration six-mode training and cached decoding remain open.

Reproducibility Statement

Appendix C records the training policy, token budget, and result artifact for each headline checkpoint. Appendices D–E report training cost, optimization, and compilation details. Appendix B and Appendices H–J document predictor splits, held-out errors, compiler diagnostics, and measurement-in-the-loop controls. Anonymous code, configurations, raw result artifacts, and exact selected execution programs are provided at the artifact link on page 1.

AI Use Disclosure

Generative AI tools were used in supporting roles during this work, including writing, debugging, and validating research code; literature search and summarization; manuscript preparation; and feedback on experimental methodology, follow-up experiments, and interpretation of results. For the manuscript, AI tools assisted with exploring paper structure and with drafting, revising, and improving the presentation of portions of the text.

AI-assisted code was executed, tested, and reviewed by the authors before being used for reported experiments. AI-assisted manuscript content and citations were similarly reviewed and revised. The authors made the final decisions regarding the research questions, experimental design, analysis, interpretation, scientific claims, and manuscript content, and take full responsibility for the final work.

References

Akhiad Bercovich, Tomer Ronen, Talor Abramovich, Nir Ailon, Nave Assaf, Mohammed Dabbah, et al. Puzzle: Distillation-based NAS for inference-optimized LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

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.

Ruisi Cai, Saurav Muralidharan, Greg Heinrich, Hongxu Yin, Zhangyang Wang, Jan Kautz, and Pavlo Molchanov. Flextron: Many-in-one flexible large language model. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: An automated end-to-end optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 578–594. USENIX Association, 2018.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Devvrit, Sneha Kudugunta, Aditya Kusupati, Tim Dettmers, Kaifeng Chen, Inderjit Dhillon, Yulia Tsvetkov, Hannaneh Hajishirzi, Sham Kakade, Ali Farhadi, and Prateek Jain. MatFormer: Nested transformer for elastic inference. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4461.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Annual Meeting of the Association for Computational Linguistics*, pp. 12622–12642, 2024.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020.

Bobby He and Thomas Hofmann. Simplifying transformer blocks. In *International Conference on Learning Representations*, 2024.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. DynaBERT: Dynamic BERT with adaptive width and depth. In *Advances in Neural Information Processing Systems*, 2020.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pp. 646–661, 2016.

Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. TASO: Optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 47–62, 2019a. doi: 10.1145/3341301.3359630.

Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. In *Proceedings of Machine Learning and Systems*, 2019b.

Gyudong Kim, Hyukju Na, Jin Kyu Kim, Hyunsung Jang, Jaemin Park, Jaegi Hwang, Namkoo Ha, Seungryong Kim, and Young Geun Kim. First attentions last: Better exploiting first attentions for efficient parallel training. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4711.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Xiaobo Liang, Lijun Wu, Juntao Li, Yue Wang, Qi Meng, Tao Qin, Wei Chen, Min Zhang, and Tie-Yan Liu. R-Drop: Regularized dropout for neural networks. In *Advances in Neural Information Processing Systems*, 2021.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL*, 2025.

Timur Mudarisov, Mikhail Burtsev, and Radu State. Feed-forward steering in transformer residual dynamics. *arXiv preprint arXiv:2608.02071*, 2026.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, volume 5, 2023.

Rohan Baskar Prabhakar, Hengrui Zhang, and David Wentzlaff. Kraken: Inherently parallel transformers for efficient multi-device inference. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0256.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, 2022.

Yining Shi, Zhi Yang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Ziming Miao, Yuxiao Guo, Fan Yang, and Lidong Zhou. Welder: Scheduling deep learning memory access via tile-graph. In *USENIX Symposium on Operating Systems Design and Implementation*, pp. 701–718, 2023.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Shashank Sonkar and Richard G. Baraniuk. Investigating the role of feed-forward networks in transformers using parallel attention and feed-forward net design. *arXiv preprint arXiv:2305.13297*, 2023.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(24):25219–25227, 2025.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, Xi Luo, Dheevatsa Mudigere, Jongsoo Park, Misha Smelyanskiy, and Alex Aiken. Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization. In *USENIX Symposium on Operating Systems Design and Implementation*, pp. 267–284, 2022.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 billion parameter autoregressive language model. Model and software release, 2021.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: Hardware-aware transformers for efficient natural language processing. In *Annual Meeting of the Association for Computational Linguistics*, pp. 7675–7688, 2020.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared LLaMA: Accelerating language model pre-training via structured pruning. In *International Conference on Learning Representations*, 2024.

Yichen Yang, Phitchaya Mangpo Phothilimthana, Yisu Remy Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. Equality saturation for tensor graph superoptimization. In *Proceedings of Machine Learning and Systems*, 2021.

Jiahui Yu and Thomas S. Huang. Universally slimmable networks and improved training techniques. In *IEEE/CVF International Conference on Computer Vision*, pp. 1803–1811, 2019.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas S. Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019.

Muru Zhang, Mayank Mishra, Zhongzhu Zhou, William Brandon, Jue Wang, Yoon Kim, Jonathan Ragan-Kelley, Shuaiwen Leon Song, Ben Athiwaratkun, and Tri Dao. Ladder-residual: Parallelism-aware architecture for accelerating large model inference with communication overlapping. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75779–75792. PMLR, 2025.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *USENIX Symposium on Operating Systems Design and Implementation*, pp. 559–578, 2022.

A NESTED EXECUTION FAMILIES

Table 4: The three program families share one training objective and progressively expand the structural intervention.

Family	Layer modes	Scientific role	Program space	Program-cost model
Binary	S, P	Isolate dependency rewrite	2^L	Exact defect + count adjustment
Ternary	S, P, K	Add complete-block removal	3^L	Calibrated layer/skip costs
Six-mode	S, P, R, A, F, K	Deployment language	6^L	Layer–mode additive costs

B PROGRAM-COST PREDICTOR SPLITS

Program-cost predictors use disjoint fitting and evaluation splits. The controlled additive model uses 1,500 fitting and 1,500 held-out programs. The archive surrogate uses an 80/20 split of 4,171, 3,898, and 3,580 programs at 120M, 430M, and 1B, respectively; all reported correlations use only the test split.

C HEADLINE CHECKPOINT PROVENANCE

Tables 5–7 record the policy and result artifact for every scale series used in the main binary, ternary, and six-mode claims. Paths are relative to the experiment repository. Token counts are computed from steps, batch size, and context length. The 430M binary EPT trained for 10B tokens uses a memory-efficient two-pass variant: the consistency gradient is stopped through z_s, while separate language-model gradients are retained for both programs. The headline six-mode checkpoints backpropagate the consistency loss through both z_s and z_m.

Table 5: Binary-checkpoint provenance. All checkpoints use the primary architecture.

Checkpoint type	Scale	Training protocol	Consistency policy	Tokens	Reference $\mathrm{BPB}(S^L)$	Result artifact
Specialist	120M	All sequential	none	57M	1.1371	`results/binary/120m_sequential_specialist_graphs.json`
Binary EPT	430M	Joint S^L/P^L; $p_w = .5$	gradient-normalized, cap 10	98M	1.0724	`results/binary/430m_gradnorm_graphs.json`
Binary EPT	430M	Joint S^L/P^L; $p_w = .5$	fixed $\lambda = .1$	10B	0.8225	`results/binary/430m_poly_10b_step076293_graphs.json`
Binary EPT	1B	Joint S^L/P^L; $p_w = .5$	fixed $\lambda = 1$	197M	1.0470	`results/binary/1b_poly_graphs.json`
Binary EPT	3B	Joint S^L/P^L; $p_w = .5$	gradient-normalized, $\rho = .2$	98M	1.1639	`results/binary/3b_gradnorm_v2_graphs.json`
Binary EPT	7B	Joint S^L/P^L; $p_w = .5$	fixed $\lambda = .1$, stop-gradient	98M	1.2875	`results/binary/7b_12k_poly_graphs.json`

Table 6: Ternary-checkpoint provenance. All checkpoints use the primary architecture and sample layer modes with $(\mathrm{S}, \mathrm{P}, \mathrm{K}) = (.5, .4, .1)$.

Scale	Consistency-weight policy	Tokens	Reference $\mathrm{BPB}(S^L)$	Result artifact
120M	fixed $\lambda = 1$	57M	1.1758	`results/ternary/ternary_consistent_120m_mask_eval.json`
430M	fixed $\lambda = 1$	98M	1.0985	`results/ternary/ternary_consistent_430m_mask_eval.json`
1B	fixed $\lambda = 1$	98M	1.1455	`results/ternary/ternary_consistent_1b_mask_eval.json`
3B	fixed $\lambda = .1$	98M	1.1531	`results/ternary/ternary_consistent_3b_mask_eval.json`

Table 7: Six-mode-checkpoint provenance. Sampling probabilities follow the mode order $(\mathrm{S}, \mathrm{P}, \mathrm{R}, \mathrm{A}, \mathrm{F}, \mathrm{K})$; "uniform" denotes the approximately uniform fixed-weight runs, and "weighted" denotes $(.30, .30, .15, .075, .075, .10)$.

Architecture	Scale	Sampling	Consistency-weight policy	Tokens	Reference $\mathrm{BPB}(S^L)$	Result artifact
Primary	120M	uniform	fixed $\lambda = 10$	57M	1.3905	`results/six_mode/6mode_masks_120m_cw10.json`
Primary	430M	uniform	fixed $\lambda = 10$	98M	1.1722	`results/six_mode/6mode_masks_430m_cw10.json`
Primary	120M	weighted	gradient-normalized	115M	1.1080	`results/six_mode/6mode_masks_120m_gradnorm_7k.json`
Primary	430M	weighted	gradient-normalized, cap 10	98M	1.1282	`results/six_mode/6mode_masks_430m_gradnorm.json`
Primary	1B	weighted	gradient-normalized, cap 10	98M	1.1177	`results/six_mode/6mode_masks_1b_gradnorm_cap10_24k.json`
Llama	120M	weighted	gradient-normalized, $\rho = .2$	57M	1.2193	`results/six_mode/6mode_masks_llama_120m_gradnorm.json`
Llama	430M	weighted	gradient-normalized, $\rho = .2$	98M	1.1205	`results/six_mode/6mode_masks_llama_430m_gradnorm.json`
Llama	1B	weighted	gradient-normalized, $\rho = .2$	98M	1.1045	`results/six_mode/6mode_masks_llama_1b_gradnorm.json`

D Training and deployment cost

Training throughput was recorded by the training loop as unique data tokens processed per elapsed second; the two execution paths used by paired training are not counted as additional data tokens. On one RTX PRO 6000 Blackwell Max-Q, the 1B full-gradient six-mode run sustained 6,886 tokens/s with batch size 4, whereas its sequential-specialist configuration sustained 15,766 tokens/s with batch size 8. Thus, at the logged configurations, six-mode training required 2.29× as much elapsed time per data token. The memory-efficient 430M binary run sustained 16,920 tokens/s with batch

size 128, versus 35,006 tokens/s with batch size 8 for the sequential specialist, corresponding to 2.07× as much elapsed time per data token. These are end-to-end throughput comparisons of the configurations used, including their different batch sizes and gradient-checkpointing policies; they do not isolate the cost of any single implementation choice. Peak training memory was not recorded and is therefore not reported. At deployment, each target-specific search used 185–303 program evaluations, corresponding to 126–232 unique programs; the per-platform accounting and rejection rates appear in Table 11.

For the nearby-latency comparison in Section 4.3, we interpolate a 316M dense architecture (20 layers, width 960, 15 attention heads) and train it for the same 12,000 steps as the 430M EPT. On Blackwell, the dense specialist measures 36.29 ms and 1.0957 BPB; the selected EPT program measures 34.99 ms and 1.1138 BPB. This is one nearby operating point, not a quality–latency frontier across scales or hardware.

E Optimization and compilation details

Matrix parameters are optimized with Muon using momentum 0.95 and five Newton–Schulz steps; embeddings and scalar parameters use AdamW with $\beta = (0.9, 0.95)$. Learning rates follow a cosine warmdown over the final 30% of training. The headline primary-architecture 1B six-mode run uses a batch of 4 sequences of length 1,024, BF16 parameters, gradient checkpointing, a matrix learning rate of 0.0015, an embedding learning rate of 0.008, and zero weight decay.

For gradient-normalized training, the consistency weight is updated as

$$\lambda = \mathrm{clip}\left(\rho \frac{\|\nabla \mathcal{L}_{\mathrm{CE}}\|_2}{\max(\|\nabla \mathcal{L}_{\mathrm{con}}\|_2, 10^{-8})}, 10^{-4}, \lambda_{\max}\right). \tag{10}$$

The headline 1B run uses $\rho = 0.2$ and $\lambda_{\max} = 10$; the gradient norms are recomputed every 100 steps on a diagnostic minibatch of up to two sequences. The six-mode multiple-choice knapsack uses 2,000 bins over the available program-cost budget. We cap simultaneous non-positive-cost choices because individually benign interventions can interact when concentrated. This discretization applies only to candidate generation: every reported deployment program is subsequently executed to measure its program cost and latency.

F Binary defect identity

Define the local defect with a fixed orientation: sequential output minus parallel output. From the mode definitions,

$$\delta_l(x) = \mathrm{S}_l(x) - \mathrm{P}_l(x) = g_l(x + a_l(x)) - g_l(x) = \int_0^1 J_{g_l}(x + t a_l(x)) a_l(x)\, dt. \tag{11}$$

The directed change made by switching from sequential to parallel is therefore $\mathrm{P}_l(x) - \mathrm{S}_l(x) = -\delta_l(x)$. The identity holds when g_l is absolutely continuous along the segment $x + t a_l(x)$. Removing the dependency subtracts exactly this hidden-state FFN response.

For the binary composition analysis, we additionally use symmetric KL between the next-token distributions of a program and the sequential reference:

$$D_{\mathrm{SKL}}(m) = \frac{1}{2}\left[D_{\mathrm{KL}}(p_{\mathrm{ref}} \| p_m) + D_{\mathrm{KL}}(p_m \| p_{\mathrm{ref}})\right].$$

This vector identity motivates, but is not identical to, the scalar single-layer cost $q_l = D_{\mathrm{SKL}}(e_l)$ of the program e_l that rewrites only layer l. Across binary programs, these costs follow the empirical law

$$\widehat{D}_{\mathrm{SKL}}(m) = \alpha_0 \left(\sum_l m_l q_l\right) |m|^{-\beta}, \tag{12}$$

where $m_l \in \{0, 1\}$ marks a parallel layer and $|m| = \sum_l m_l$. The target is excess symmetric KL relative to the reference program. This count-adjusted law reaches test-set Pearson $r = 0.971$; Appendix M.3 details the fitting protocol and the subadditivity summary.

G OPERATING-POINT-DEPENDENT TERNARY COMPOSITION

At the near-matched-quality checkpoint, a layerwise model reaches $r = 0.697$. At a more strongly aligned checkpoint, additive layer–mode costs reach $r = 0.884$ on the evaluated programs and $r = 0.736$ after disjoint refitting, while skip count alone is anti-correlated ($r = -0.576$). The inversion is checkpoint-specific: alignment changes the geometry of local removal costs, so ternary cost models are calibrated to the checkpoint they compile.

H PROGRAM-COST STRUCTURE AND PREDICTION

For ternary and six-mode programs, the first-order model predicts $\Delta\mathrm{BPB}$ using

$$\widehat{\Delta\mathrm{BPB}}(m) = \sum_{l=1}^{L} \sum_{q \neq \mathrm{S}} \mathbf{1}[m_l = q] c_{lq}. \tag{13}$$

Sequential is the zero reference and the implementation includes no intercept. On a well-aligned ternary checkpoint, layer–mode additive costs reach test-set $r = 0.736$ and outperform skip count alone (Appendix G).

On the gradient-normalized six-mode checkpoints, the ridge-regularized additive model obtains test-set Pearson correlations of 0.721, 0.573, and 0.580 at 120M, 430M, and 1B. Each fit uses 1,500 uniformly sampled programs for predictor training and a disjoint 1,500-program test set. A richer surrogate uses a mode-specific, pairwise-distance composition law as a prior and fits a gradient-boosted residual on the accumulated measurement archive. With an 80/20 train/test split, it reaches 0.888, 0.768, and 0.859 on 4,171, 3,898, and 3,580 measured programs, respectively. These archive results use different data and splits from the controlled additive fits and are not directly comparable. Appendix J reports absolute prediction errors and the measurement-budget ablation.

Parallel and reverse are close to neutral in the single-layer probes (Figure 3), while removing a branch or complete block is more consequential. The ordering of attention-only, FFN-only, and skip changes with scale, so the stable distinction is between dependency rewrites and operator removal rather than a universal ranking of removal modes. The fitted coefficients summarize the observed checkpoint; they are not causal effects of individual modes.

On Llama-style blocks, binary symmetric-KL composition reaches five-fold Pearson $r = 0.8174$ at 430M. The six-mode $\Delta\mathrm{BPB}$ fit is harder, with correlations from 0.588 to 0.625. Dependency rewrites therefore transfer more readily than operator removal.

Table 8: Programs selected independently from the same 1B EPT with $\varepsilon = 0.05$. Search and disjoint columns measure each program relative to the EPT's sequential execution. The specialist column instead measures the collapse caused by forcing the identical program onto independently trained sequential weights; it is not an absolute quality gap between the two checkpoints. Mode counts follow $(\mathrm{S}, \mathrm{P}, \mathrm{R}, \mathrm{A}, \mathrm{F}, \mathrm{K})$. Exact layerwise mode assignments will be included with the released code and result artifacts.

Hardware	Speedup	Search $\Delta\mathrm{BPB}(m)$	Disjoint $\Delta\mathrm{BPB}(m)$	Specialist forcing $\Delta\mathrm{BPB}_{\theta_{\mathrm{spec}}}(m)$	Mode counts
Blackwell	1.709×	0.04843	0.0512	1.3690	$(4, 7, 0, 5, 3, 9)$
H100	1.568×	0.04923	0.0503	0.3685	$(6, 6, 1, 7, 1, 7)$
L40S	1.697×	0.04999	0.0516	1.7767	$(9, 5, 0, 5, 0, 9)$

Table 9: Matched mode-set ablation on the 430M six-mode EPT on Blackwell. All searches use the same $\varepsilon = 0.05$ search-window budget.

Allowed modes	Speedup
$\{\mathrm{S}, \mathrm{P}, \mathrm{R}, \mathrm{A}, \mathrm{F}, \mathrm{K}\}$	1.669×
$\{\mathrm{S}, \mathrm{A}, \mathrm{K}\}$	1.602×
$\{\mathrm{S}, \mathrm{P}, \mathrm{R}\}$	1.003×

Table 10: Pure-program quality and runtime for the primary-architecture 1B six-mode EPT at 24k training steps. Each row assigns the same mode to all 28 layers and is measured directly without search. Runtime uses a fixed batch of 8 sequences of length 1024 in BF16; speedup is relative to the same platform's all-sequential program. Program cost is measured relative to the checkpoint's all-sequential program.

Pure program m	$\Delta\mathrm{BPB}(m)$	Blackwell speedup	H100 speedup	L40S speedup
S^L	0.0000	1.000×	1.000×	1.000×
P^L	0.0020	1.002×	1.006×	0.999×
K^L	2.5685	38.690×	37.930×	22.450×
R^L	0.0023	0.660×	0.640×	0.700×
A^L	1.2929	1.840×	1.650×	2.020×
F^L	1.0052	1.010×	1.010×	1.010×

Table 11: Accounting for the three independent 1B program searches. Feasibility counts and percentages use unique programs; repeated evaluations explain the larger evaluation totals. Search decisions use directly measured program cost and latency rather than predicted costs. Disjoint-window program costs are evaluated only after selection and are not used to choose the reported programs.

Platform	Evaluations	Unique programs	Within ε	Above ε	Disjoint $\Delta\mathrm{BPB}(m)$
Blackwell	303	232	142 (61.2%)	90 (38.8%)	0.0512
H100	185	143	84 (58.7%)	59 (41.3%)	0.0503
L40S	303	126	70 (55.6%)	56 (44.4%)	0.0516

I Additional Blackwell compiler diagnostics

The frontier in Table 12 uses separate Llama-style checkpoints across scales and complements the main three-platform result, which uses one primary-architecture 1B checkpoint. It exposes prediction residuals directly; these predicted costs guide candidate generation but do not certify feasibility.

Table 12: Additional predicted and measured compiler points on Blackwell for Llama-style six-mode checkpoints. Hardware and architecture are fixed across rows and therefore omitted from the table body.

Scale	Predicted budget ε	Speedup	Predicted $\widehat{\Delta\mathrm{BPB}}(m)$	Measured $\Delta\mathrm{BPB}(m)$	Matched-context agreement	Non-sequential mode counts
120M	0	0.996	−0.00489	0.00530	0.9214	1 A
120M	0.005	1.079	0.00451	0.00526	0.9185	1 K, 1 A
120M	0.020	1.258	0.01930	0.02891	0.8223	2 K, 2 A
430M	0	1.202	−0.00052	0.03279	0.8462	1 K, 5 A
430M	0.005	1.278	0.00476	0.03303	0.8447	2 K, 5 A
430M	0.020	1.401	0.01941	0.09719	0.7231	2 K, 7 A
1B	0	1.358	−0.00005	0.01163	0.8906	5 K, 4 A
1B	0.0005	1.406	0.00015	0.01115	0.8818	5 K, 5 A
1B	0.002	1.433	0.00134	0.05580	0.8340	4 K, 7 A
1B	0.005	1.480	0.00462	0.05197	0.8359	6 K, 5 A
1B	0.020	1.686	0.01886	0.04136	0.8306	9 K, 4 A

J Surrogate-guided search versus direct measurement

Table 13 complements correlation with absolute held-out error for the archive-trained six-mode predictor. RMSE is measured in the same within-checkpoint program-cost units, $\Delta\mathrm{BPB}(m)$, used by

the deployment budget. These errors motivate treating predictions as candidate proposals: feasibility is established by direct program measurement rather than by thresholding the predictor, so predictor precision or recall at $\varepsilon = 0.05$ is not used as a deployment guarantee.

Table 13: Held-out accuracy of the archive-trained six-mode program-cost predictor. Each checkpoint uses an 80/20 train/test split of its measurement archive.

Scale	Archive programs	Pearson r	RMSE (BPB)
120M	4,171	0.888	0.0420
430M	3,898	0.768	0.0407
1B	3,580	0.859	0.0174

The main text notes that the additive predictor proposes candidates rather than certifying feasibility. We tested the natural stronger claim directly: does using the predictor to *rank* candidates for measurement, rather than only to score them afterward, reduce the number of measurements needed to reach a given program? On a gradient-normalized 430M six-mode checkpoint at program-cost budget 0.05, we compared plain random-neighbor local search against the same search with a candidate-ranking filter (evaluate a larger neighbor pool with the fitted predictor, measure only its top-ranked subset) under matched measurement budgets across four seeds.

Table 14: Matched-measurement-budget comparison on the primary-architecture 430M gradient-normalized checkpoint at 12k training steps and $\varepsilon = 0.05$. Both variants continue from the same measured program for 15 steps with four measured neighbors per step; only the criterion for selecting which neighbors to measure differs, and every acceptance decision uses measured program cost.

Seed	Plain speedup	Predictor-ranked speedup
123	1.723×	1.719×
7	1.747×	1.718×
42	1.720×	1.717×
999	1.719×	1.717×

Plain measurement-in-the-loop search matched or exceeded the predictor-ranked variant in all four seeds (Table 14). We quantify why: within each search window, the ratio of the predictor's residual scale to its predicted spread across competitive candidates ($\kappa \approx 0.62$, with an independent pairwise version $\mathcal{R} \approx 1.38$) shows the predictor's local ranking resolution among near-optimal candidates is only marginal, with no trend across the search trajectory. A separate falsification test supports the same conclusion: a candidate mode-transition correction to the predictor, fit on one search trajectory, does not generalize under leave-one-anchor-out validation across 30 independently sampled anchors ($R^2 = -0.16$), ruling out a simple missing deterministic term. The predictor retains useful average-case and ordinal information—general prediction accuracy improves substantially with more training data—but this is a distinct quantity from having enough local resolution to prune the specific candidates that a measurement-based search would otherwise find. This is consistent with, and sharpens, the main text's limitation: the correct role of the composition law is to describe average program cost and propose candidates, not to replace direct measurement during search.

K Cross-architecture six-mode analysis

The Llama 120M six-mode checkpoint has reference BPB 1.2193, mean agreement 70.63%, mean program cost +0.1151 BPB, and fifth-percentile agreement 51.41% under 3,000 uniformly sampled programs. Increasing the number of parallel or reverse layers reduces mean program cost, whereas skip and FFN-only account for most of the additional cost. At six uses, mean program cost is approximately 0.045 BPB for parallel, 0.064 BPB for reverse, 0.313 BPB for skip, and 0.219 BPB for FFN-only; attention-only is comparatively flat in the observed range. The breakdown shows that dependency-rewrite robustness already transfers and pinpoints operator-removal alignment as the main opportunity for improving cross-architecture six-mode training.

L ADDITIONAL ABLATIONS AND REPORTING PRINCIPLES

Table 15 summarizes recurring questions, the supporting evidence, and the resulting reporting principles.

Table 15: Recurring questions, supporting evidence, and reporting implications.

Question	Evidence	Implication
Are ordinary checkpoints program-robust?	A 120M specialist has $\mathrm{BPB}(S^L) = 1.137$ and $\mathrm{BPB}(P^L) = 1.681$, with 37.1% matched-context agreement.	Program robustness must be learned explicitly.
Does binary composition transfer to Llama?	Five-fold count-adjusted Pearson is $r = 0.8174$ at 430M.	The structure transfers, with lower predictiveness than on the primary architecture.
How do consistency-weight policies compare?	Fixed consistency weights and gradient-normalized consistency weights both support execution-program polymorphism at different operating points.	The paired objective is shared, while consistency-weight policies select operating points with different tuning burdens.
How should ternary costs be modeled?	A strongly aligned checkpoint favors additive costs ($r = 0.884$), while simpler skip-count summaries depend on the operating point.	Cost models are calibrated to each checkpoint and program family.
How are compiled candidates verified?	Predictor accuracy is strongest near the reference and degrades for aggressive correlated programs.	Predicted costs guide search, followed by direct measurement of selected programs.
How broad is the hardware evidence?	Six-mode compiler outputs are measured on Blackwell across three scales and independently searched at 1B on Blackwell, H100, and L40S.	The selected program remains specific to its target hardware and workload.

M SUPPORTING TERNARY, BINARY, AND RUNTIME EVIDENCE

M.1 TERNARY CONTROLS AND THREE-FAMILY PROGRESSION

The ternary family tests whether alignment survives complete-block removal. Prior work obtains cheaper models through layer dropout, early exit, or post-training block removal (Fan et al., 2020; Elhoushi et al., 2024; Men et al., 2025; Xia et al., 2024); ternary EPTs instead retain many non-prefix skip patterns inside one checkpoint. Across 5,000 independently sampled programs at each of 120M, 430M, 1B, and 3B, the selected checkpoints obtain mean ΔBPB of 0.014–0.032 (Figure 5). At 430M, a binary-only checkpoint has mean $\Delta\text{BPB} = 0.244$; one-pass ternary augmentation reduces it to 0.0189; matched-FLOP augmentation gives 0.0520; and the compute-matched two-pass LM control gives 0.0328. Adding consistency to the same two-pass setup reduces the mean to 0.016. Across all four scales, paired ternary training reduces mean ΔBPB by 13–16$\times$ relative to binary-trained controls.

At 430M, the quality-oriented checkpoint has reference BPB 1.099 and mean $\Delta\text{BPB} = 0.0156$. A Llama-style 430M checkpoint reaches 84.45% agreement at reference BPB 1.1160. A more strongly aligned primary checkpoint raises mean agreement to 94.1% but worsens reference BPB to 1.450, illustrating why agreement must be reported with absolute model quality.

Table 16: Primary-architecture gradient-normalized six-mode checkpoints under 3,000 uniformly sampled programs. Specialist BPB uses a sequential model with the same architecture, training data, and token budget.

Scale	Reference $\text{BPB}(S^L)$	Specialist BPB	Specialist gap Δ_{spec} (BPB)	Mean $\Delta\text{BPB}(m)$	95th pct. $\Delta\text{BPB}(m)$
120M	1.1080	1.0861	+0.0219	0.1049	0.2671
430M	1.1282	1.0853	+0.0429	0.0699	0.1604
1B	1.1177	1.0810	+0.0367	0.0630	0.1318

For the six-mode consistency control reported in Section 4.2, we compare matched 430M runs with $\lambda = 0$ and $\lambda = 1$ at step 12,000. Both are evaluated on the same 500 programs under the same sampling protocol. Mean agreement is 76.11% without consistency and 76.31% with consistency; the corresponding mean ΔBPB values are 0.0711 and 0.0648.

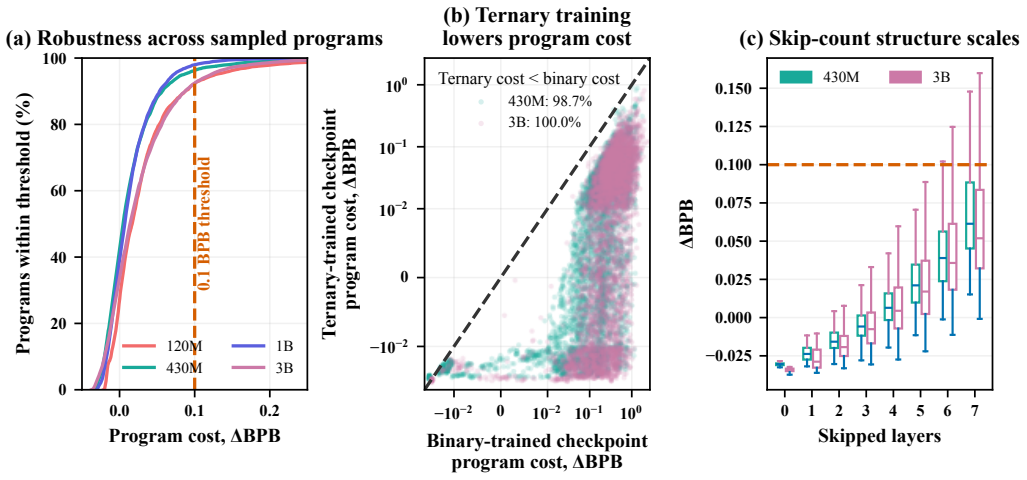

Figure 5: Ternary robustness across scale. These checkpoints use different values of λ, and program cost is measured relative to each checkpoint's sequential reference. The figure therefore evaluates within-checkpoint program robustness rather than absolute model quality.

M.2 Additional binary evidence

At 12k training steps, the primary 430M binary EPT has reference BPB 1.0724, $\Delta\mathrm{BPB}(P^L) = 0.00051$, and 98.39% agreement. An independently trained Llama-style 1B checkpoint obtains reference BPB 1.0035, $\Delta\mathrm{BPB}(P^L) = 0.00171$, and 91.16% agreement.

A parallel specialist is also brittle in the reverse direction, moving from 1.147 to 1.310 BPB when executed sequentially. This sensitivity is consistent with post-training analyses of sequential-to-parallel interventions (Mudarisov et al., 2026); the controlled results in the main text show that robustness to the intervention can instead be learned.

M.3 Binary composition mechanism

Figure 6 uses $|m|$ for the number of parallel layers. "Few-to-many" fits on programs with at most 10 parallel layers and tests on programs with more than 10; "many-to-few" reverses those sets; and five-fold CV denotes five-fold cross-validation. We summarize subadditivity with the endpoint ratio

$$\eta = \frac{D_{\mathrm{SKL}}(P^L)}{\sum_{l=1}^{L} D_{\mathrm{SKL}}(e_l)}, \tag{14}$$

where P^L is the all-parallel program and e_l rewrites only layer l. Thus $\eta < 1$ indicates that the all-parallel program costs less than the sum of the single-layer costs.

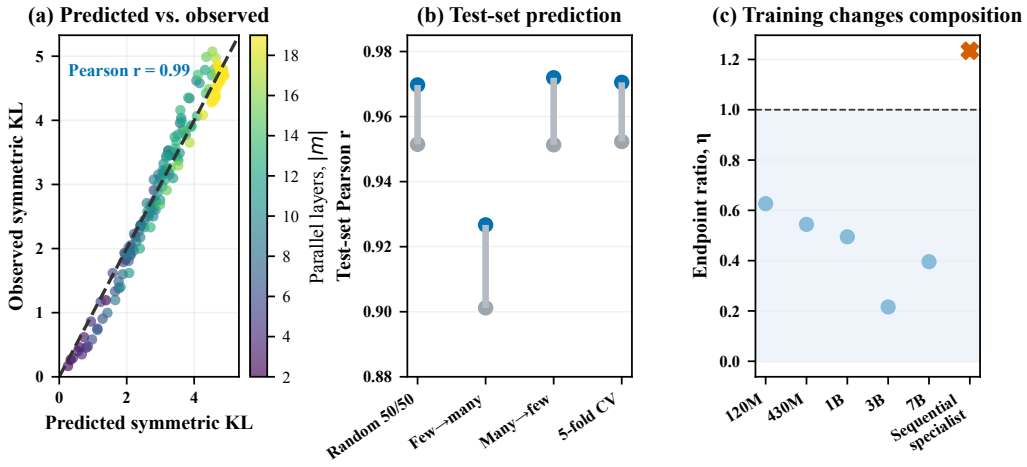

Figure 6: Binary composition evidence. The count-adjusted law gives the strongest composition result for binary rewrites; ternary and six-mode families use separately calibrated additive models.

M.4 WORKLOAD-DEPENDENT RUNTIME BEHAVIOR

Figure 7 shows the measured speed–quality landscapes for the three platforms.

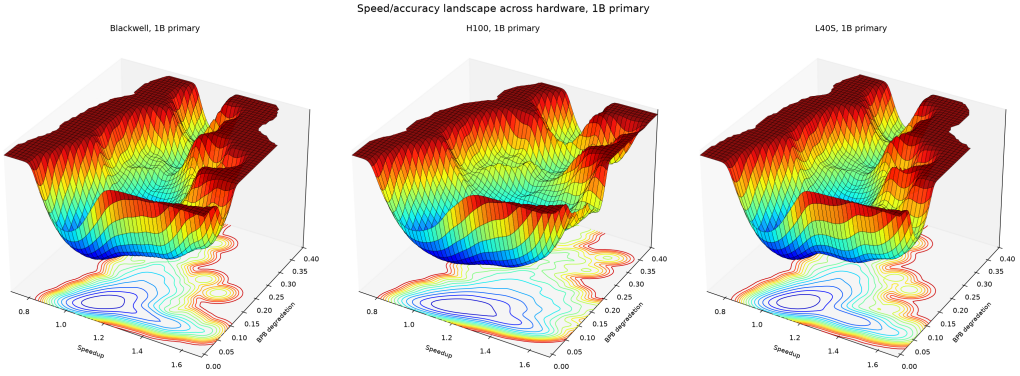

Figure 7: Empirical speed–quality landscapes for measured programs from the primary-architecture 1B six-mode checkpoint on Blackwell, H100, and L40S, from left to right. The horizontal axes show speedup relative to each platform's all-sequential program and within-checkpoint program cost $\Delta\mathrm{BPB}(m)$; surface height visualizes program density rather than predicted quality or an optimization objective. These landscapes characterize the measured program family and do not constitute an absolute-quality comparison against separately trained models.