POSITION: PRETRAINED LLMS SHOULD MOVE FROM LINEAR TO GRAPH-STRUCTURED EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) typically execute their layers in a fixed linear sequence. Post-training methods modify this structure through skipping, looping, reordering, or parallel execution, offering different capability and resource benefits without training a new LLM from scratch. However, these operations are often studied separately. This position paper argues that research on post-training LLM restructuring should systematically explore how multiple structural operations can be combined to pursue their different benefits within the same LLM. We introduce model execution graphs to realize this position, jointly specifying which pretrained layers execute, how often, and how they are connected. We also develop an evaluation framework that assesses multiple capabilities and resource costs and examines whether composition provides benefits beyond applying the operations separately. Experiments across pretrained LLMs show that combined operations can achieve simultaneous resource savings, but these savings can accompany task-performance losses and do not necessarily confer an advantage over simpler alternatives. The same graph can also affect task performance differently across LLMs. Finally, we outline future research directions for designing and evaluating model execution graphs. Code and data are available in the anonymous repository.

1 INTRODUCTION

Large language models (LLMs) are generally pretrained as a fixed sequence of Transformer layers and retain this sequence when deployed (Vaswani et al., 2017; Qwen Team, 2024; Jiang et al., 2023). Recent work has explored whether this serial structure can be modified after pretraining. Starting from an existing LLM, layer-skipping methods bypass selected layers (Men et al., 2024; Gromov et al., 2025; Csordás et al., 2025), loop-based methods repeatedly apply an individual layer or a group of layers (Chen et al., 2026), and layer-parallel methods execute multiple layers in parallel rather than strictly one after another (Sun et al., 2025). Together, these studies show that restructuring a pretrained LLM can yield gains in efficiency or model performance. More broadly, these results suggest that pretrained layers can be used to restructure a model with an alternative execution structure without retraining it from scratch.

Existing work, however, typically studies one type of structural change at a time. Skipping, looping, and parallel execution are developed as separate alternatives, and an edited model is usually constructed using only one of them. This separation leaves a larger set of possible structures unexplored. A single edited model could instead combine these operations within its execution—for example, skipping some layers, reusing others, and executing others in parallel. **Position: Pretrained LLMs should be restructured as execution graphs that combine multiple structural operations.**

We make this position concrete by restructuring a pretrained LLM as a **model execution graph**, in which multiple structural operations jointly determine how the model is executed. Each node invokes a pretrained Transformer layer once, and each directed edge carries the resulting hidden state to a downstream node. Modifying the nodes and edges determines which layers are executed, how often and in what order they are executed, and where the computation branches and merges. The model performs inference by executing this graph. Existing methods correspond to graphs restricted to limited types of structural change, whereas model execution graphs can combine multiple operations within the same model.

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.

Combining multiple structural operations also changes how edited models should be evaluated. Existing studies often emphasize the capability or efficiency measure their method is designed to improve, which can hide costs elsewhere; for example, faster inference may be accompanied by weaker reasoning. A systematic evaluation must therefore assess the resulting model across a broad set of dimensions. It must also determine whether combining operations produces gains beyond applying the same operations separately. We develop an evaluation framework that addresses both requirements and apply the same model execution graphs across multiple pretrained LLMs.

In summary, we develop and support this position in four ways. (1) We introduce model execution graphs for combining multiple structural operations within a pretrained LLM. (2) We develop a systematic evaluation framework that assesses the resulting models across a broad set of evaluation dimensions and quantifies the additional gains from combining structural operations. (3) We evaluate several model execution graphs across multiple pretrained LLMs, demonstrating that combining operations can yield gains beyond applying each operation alone. (4) We discuss credible alternative views and suggest directions for future research on post-training model restructuring.

2 Why Combine Structural Operations?

Structural operations alter which pretrained layers are invoked, how often they are invoked, and which invocations must occur in sequence. Skipping removes calls, looping repeats calls or shares weights, reordering changes their serial order, and parallel execution removes selected serial ordering constraints. Because these changes pursue different benefits and expose different costs, their consequences must be assessed both separately and in combination.

2.1 Structural Operations

The schematics below show how each operation changes a short serial layer chain.

Skip. Skipping removes the invocation of a selected pretrained layer and reconnects the preceding layer directly to the next retained layer. In the schematic, the path $\alpha \to \beta \to \gamma$ becomes $\alpha \to \gamma$. ShortGPT selects layers using hidden-state similarity (Men et al., 2024), while contiguous depth-pruning methods identify removable regions and then adapt the shortened model (Gromov et al., 2025). Skipping reduces active depth, but it can remove transformations needed by later layers, and different skipped positions produce different changes in model performance (Csordás et al., 2025).

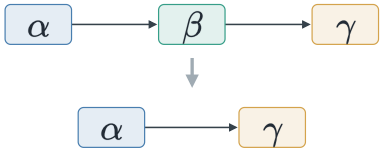


Figure 1: Skip a layer.

Loop. Looping invokes a pretrained layer or layer group more than once. In the schematic, β is executed twice in total before its output is passed to γ; the loop denotes finite repetition rather than indefinite cycling. Additional loop iterations can improve task performance without updating the checkpoint (Chen et al., 2026), but require additional computation and may increase latency. Looping can also replace several distinct layers with repeated calls to a smaller retained group, thereby reducing parameter storage through weight sharing (Bae et al., 2025).

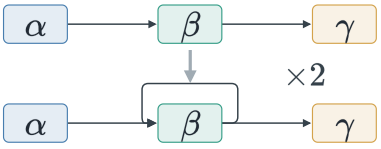


Figure 2: Repeat a layer.

Reorder. Reordering changes the serial order of pretrained layers and therefore the hidden states they receive. The schematic changes $\alpha \to \beta \to \gamma \to \delta$ into $\alpha \to \gamma \to \beta \to \delta$ while retaining the same layer parameters. Prior work evaluates adjacent swaps, reversal, and randomized middle-layer orders (Sun et al., 2025). These operations preserve the number of layer calls and the length of the serial path, but can disrupt interfaces learned under the pretrained order.

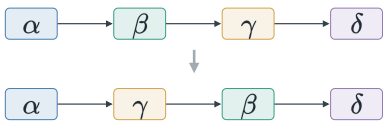


Figure 3: Reorder layers.

Parallelize. Parallel execution removes selected serial ordering constraints by sending the same hidden state from α to β and γ, then merging both results before δ. Prior work evaluates parallel middle layers whose outputs are combined by averaging (Sun et al., 2025). This structure can shorten the longest serial chain without reducing layer calls, although an actual latency reduction requires a concurrent implementation and suitable hardware. Section 3 defines the numerical merge rule used in our graphs.

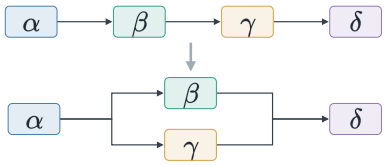
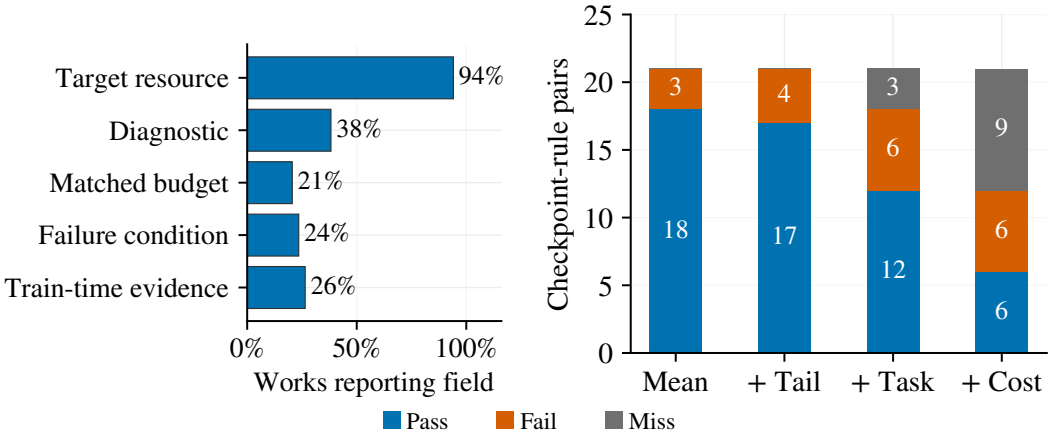


Figure 4: Parallelize layers.

2.2 Limitations of a Single Operation

No individual operation improves every relevant outcome. Skipping reduces active computation but can delete transformations needed by later layers. Looping can add processing or reduce stored weights, but extra invocations increase computation and shared-layer replacement changes the hidden-state interfaces inherited from pretraining. Reordering changes those interfaces without reducing the number of layer calls. Parallel execution shortens the longest serial chain, but provides no realized speedup when its branches are executed sequentially. Each operation therefore brings both an intended benefit and an operation-specific cost.



(a) Reporting-field coverage.

(b) Decisions after successive measurements.

Figure 5: Reporting coverage and model-selection results under different evaluation requirements.

Prior skipping work primarily motivates removing layers through lower active depth and inference cost (Men et al., 2024; Gromov et al., 2025). Across the selected 34-work corpus, target resources are reported in 94% of works, whereas matched budgets and explicit failure conditions appear in 21% and 24%, respectively (Figure 5a; Appendix E). This reporting gap motivates our controlled comparison of why intended benefit alone is insufficient for deciding whether to adopt a skip. We apply seven selection rules to each of Qwen2.5-1.5B, Mistral-7B, and Gemma2-2B, yielding 21 checkpoint–rule entries that each skip two layers; different rules may select the same pair. Five of these rules appear in Figure 7, while the other two are random selection and a final-hidden-state cosine control. Eighteen entries satisfy a mean per-sequence negative-log-likelihood increase of at most 1.0 nat per scored token. Requiring in addition a 95th-percentile per-sequence increase of at most 1.0 leaves 17 entries; allowing at most a 0.05 downstream-accuracy drop leaves 12; and requiring median prefill- and decode-time ratios no greater than 1.0 leaves 6 (Figure 5b). An entry meets a requirement set only when every required measurement is present and passes. Vermilion means that at least one measured requirement fails and the decision rule selects the original model; gray means that a required measurement was not collected. Average loss alone therefore cannot settle whether a skip meets all stated requirements.

The tail, task, and serving checks expose costs that mean loss alone misses; they neither show that every skip is harmful nor demonstrate that composition is superior. Before expanding from single-

operation candidates to a larger space of combined operations, Section 3 therefore asks how clearly harmful candidates can be screened.

2.3 Combining Multiple Operations

The different trade-offs provide a reason to combine operations: one operation may offset a cost introduced by another. Skipping can compensate for additional calls introduced by looping, and parallel execution can shorten the longest serial chain while repetition adds processing elsewhere. Prior work has evaluated skipping, reordering, and parallel execution as alternative structures (Sun et al., 2025), and Program-of-Layers permits input-specific programs that both skip and loop pretrained layers (Li et al., 2026). The open question is therefore how to specify, select, and evaluate combined operations as complete models.

Composition is not automatically better than a single operation. One edit can produce a different quality or resource change after another edit alters the hidden state it receives, and a simpler graph may achieve the same deployment objective. LLM restructuring should therefore determine the operations, their locations, and their connections jointly, while retaining the original chain and single-operation graphs as alternatives. The next section provides the representation and design principles needed to make those choices explicit.

3 From Linear Layer Sequences to Model Execution Graphs

We represent a reconstructed model as a *model execution graph*. This representation specifies the computation produced by one or more structural operations. After defining that representation, this section addresses three design questions: which candidates can be screened before an expensive search, which graph serves the deployment objective, and which graph is appropriate for the particular pretrained model.

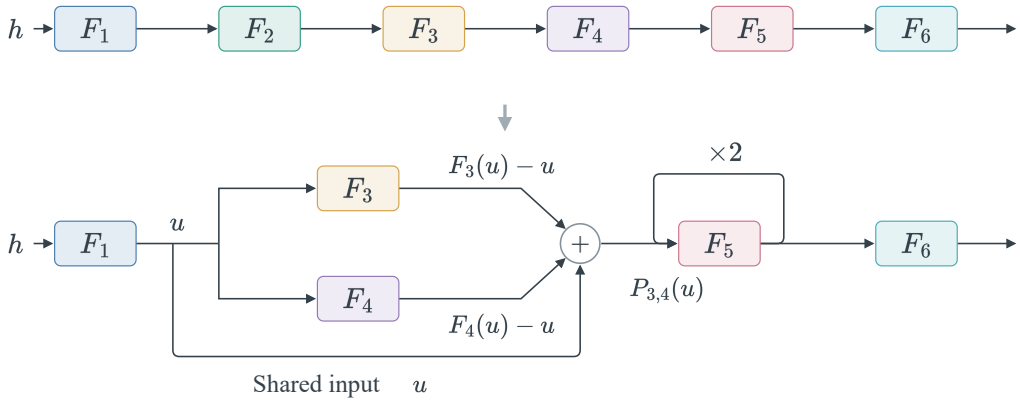


Figure 6: A six-layer execution graph combining skipping, branch–merge, and repetition.

3.1 Model Execution Graphs

Let $F_1, \ldots, F_L$ denote the Transformer layers at their pretrained positions. The original model invokes each layer once in sequence. A reconstructed model instead executes a directed graph $G = (V, E)$. Each node in V invokes one pretrained layer once, and each edge in E carries the resulting hidden state to a downstream invocation that receives it. Several nodes may refer to the same F_i, so the graph distinguishes parameter storage from the number of layer calls.

Skipping removes nodes and reconnects the remaining path, repetition adds invocations that share parameters, and reordering changes the edges that determine execution order. Parallel execution introduces branches that receive the same state and requires a merge rule before a downstream invocation. Together, node-to-layer assignments, edges, parameter references, and merge rules define the inference computation. A finite repetition is represented by unfolding its calls into a finite acyclic schedule.

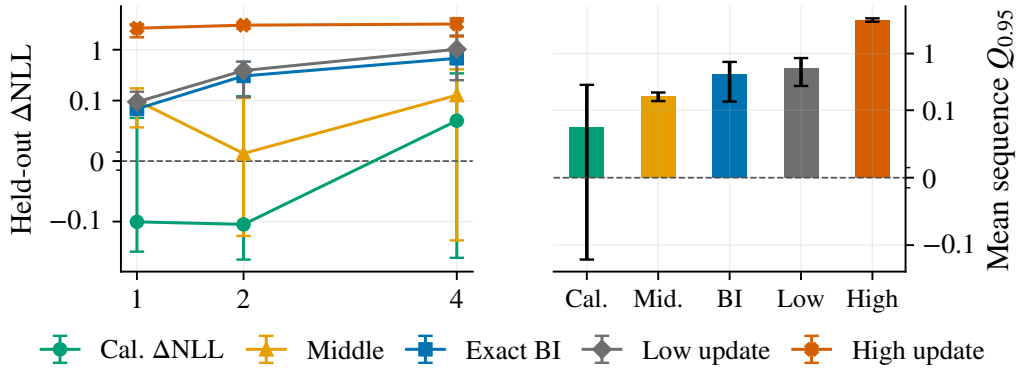
Every graph must declare a shape-compatible merge rule for each branch. The branch–merge graphs evaluated here have exactly two branches. For layers F_a and F_b acting on a hidden state h produced by the preceding graph node, the serial computation is $F_b(F_a(h))$, whereas our binary merge returns the hidden state $P_{a,b}(h)$ after both layers receive the same h and their residual updates are added:

$$P_{a,b}(h) = h + \big(F_a(h) - h\big) + \big(F_b(h) - h\big).$$

Figure 6 shows how that merge combines with skipping and repetition in a six-layer example; Appendix B gives the full computation and structural counts.

3.2 Narrowing the Search Space

Even one operation creates a large candidate set: skipping exactly m layers from an L-layer chain already gives $\binom{L}{m}$ location sets before considering repetition, reordering, branching, or combinations. Exhaustively adapting every graph is therefore impractical. Calibration-side evidence can first exclude clearly damaging candidates while held-out outcomes remain reserved for evaluation.



(a) Mean held-out damage (b) Tail damage at $k = 2$

Figure 7: Mean and upper-tail loss increases under different layer-skipping selection rules.

Figure 7 focuses on five of the seven rules used above: calibration loss chooses the smallest one-layer calibration loss increase; Exact Block Influence is the ShortGPT score (Men et al., 2024); low and high update choose the smallest and largest norms of a layer's residual update relative to its output norm on calibration text; and middle is a depth-based control. Across three WikiText partitions for each of Qwen2.5-1.5B, Mistral-7B, and Gemma2-2B, high-update selection causes markedly greater mean and tail damage, whereas the ordering of the other selectors from lowest to highest damage changes across checkpoints and edit budgets. The high-update contrast also appears after transferring selected candidates to TinyStories and a Pile subset (Appendix J, Table 35). This supports operation-specific harmful-candidate screening, not a universal selector, and the search-cost reduction itself remains unmeasured.

3.3 Design Based on Demand

Filtering removes clearly harmful candidates, but the remaining space still contains graphs with different quality–resource trade-offs. The deployment requirement determines which operations are relevant. Replacing distinct layers with shared repetitions can reduce stored weights if the replaced weights are no longer resident; skipping reduces layer calls; and concurrent branches shorten the longest serial chain only when the implementation and hardware execute them in parallel. Extra repetition instead adds computation in pursuit of task-performance gains. Operations, locations, and connections should therefore be chosen jointly for the stated demand, while retaining the original chain and simpler edits as alternatives. Each candidate must also specify its merge, cache, weight-residency, and adaptation rules so that the proposed graph denotes an executable model with a well-defined resource profile.

3.4 Design Based on Model

A shared deployment demand does not imply that one graph is best for every pretrained model. The same non-skip selection rules exchange rank across checkpoints and operation types (Appendix J, Figure 14). Even when a repetition-only graph retains every original layer and repeats the same two selected layers, accuracy on the multiple-choice ARC-Challenge benchmark (Clark et al., 2018) decreases on Qwen2.5-1.5B but increases on Qwen2.5-7B (Appendix C). Graph selection must therefore use evidence from the target checkpoint rather than assuming that an operation's effect transfers.

These model-specific comparisons identify candidates; Section 4 next asks whether their task quality and resource use make them worth adopting.

4 Evaluating Model Execution Graphs

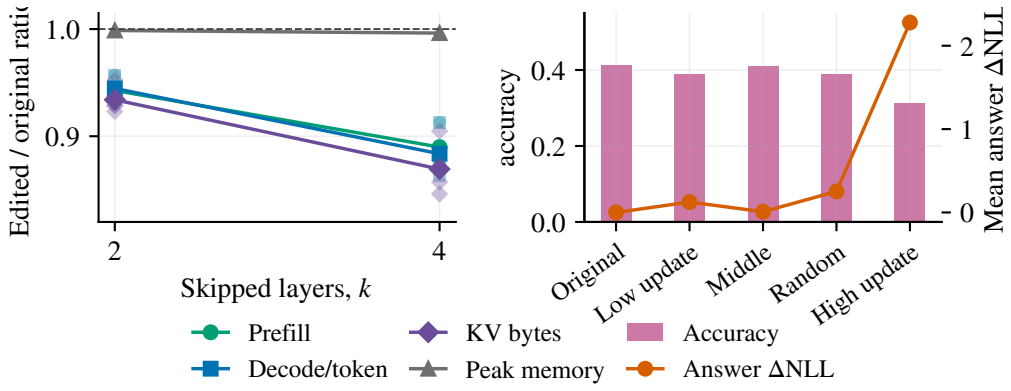
Deciding whether to adopt an executable graph requires measurements that match its deployment objective and comparisons that isolate any benefit of composition. Language-model evaluation already distinguishes task success, predictive quality, reliability, and efficiency (Liang et al., 2023; Biderman et al., 2024). Model execution graphs additionally require comparisons with the original chain, their constituent operations, and feasible simpler graphs. This section therefore proceeds in three steps: choose measurements that match the deployment requirement, test whether composition adds value beyond the original and simpler graphs, and separate candidate selection from final evaluation.

4.1 Evaluation Metrics

Task performance. When an application requires the model to answer a particular class of questions correctly, evaluation must measure success on that task rather than infer it from language-model loss. Multiple-choice settings such as ARC and MMLU therefore use answer accuracy (Clark et al., 2018; Hendrycks et al., 2021), while generative settings may require a declared answer-extraction and exact-match rule, as in GSM8K (Cobbe et al., 2021). We report each task before any aggregate so that improvement on one capability does not hide regression on another.

Language-modeling quality. When the deployment requires broad next-token prediction, or when a task score does not reveal how restructuring changes the model's probability distribution, evaluation should measure predictive quality on shared text. Negative log-likelihood and perplexity are established measures for this purpose (Merity et al., 2017; Kaplan et al., 2020). We use mean per-token NLL, written $\ell(G, x)$ for graph G on sequence x, to compare the same scored tokens before and after editing; lower values indicate better predictive quality.

Resource use. When deployment is limited by device capacity, evaluation should measure the memory that determines whether the model and its workload fit: resident weights, key–value (KV) cache, temporary activations, and peak allocation. Here, a skipped layer's weights remain resident when its parameters stay loaded in device memory even though the layer is not invoked. PagedAttention evaluates cache management through memory utilization and serving throughput (Kwon et al., 2023), while FlashAttention demonstrates the importance of memory traffic (Dao et al., 2022). When deployment is instead limited by responsiveness or serving capacity, evaluation should measure prefill or time to first token, per-token decode latency, and request or token throughput (Zhong et al., 2024; Kwon et al., 2023). These are all resource measures, but they answer different constraints and require matched hardware, precision, batch size, sequence lengths, and cache policy. In our measurements, keeping skipped weights resident reduces prefill, decode, and KV allocation but changes peak allocation little; removing them enlarges the cache pool and improves throughput (Appendix J, Tables 29 and 30). Figure 8 therefore reports memory, speed, and task quality separately.



(a) Cache-enabled serving cost (b) Downstream task behavior

Figure 8: Serving resource ratios and task performance under layer skipping.

Other outcomes. When a deployment decision requires confidence quality, behavior under perturbation, equitable performance, or safety-relevant behavior, evaluation should add calibration, robustness, fairness, bias, toxicity, or another application-specific reliability measure (Liang et al., 2023). Figure 5 demonstrates the narrower point established here: among 21 two-layer skip candidates, the supported count falls from 18 under mean loss alone to 17 with tail loss, 12 with downstream tasks, and 6 with serving cost. The selected 34-work audit also finds intended resources reported more consistently than matched budgets and failure conditions. These results motivate choosing metrics from deployment requirements rather than imposing one universal list.

A quality–cost example. The same graph can move quality and computation in opposite directions. A reproduction of the released Training-Free Looped Transformer recipe (Chen et al., 2026) on Qwen3-4B-Base produces slightly higher task-aggregate point estimates while increasing the number of Transformer-layer invocations in the complete looped forward computation from 36 to 56 and elapsed evaluation time from 829.3 to 1206.8 seconds (Appendix J, Table 27). The reproduction does not estimate run-to-run uncertainty, so it establishes a descriptive quality-for-computation trade rather than a reliable quality improvement or a resource saving.

4.2 Evaluating the Benefits of Composition

Let G_0 denote the original serial graph and G a reconstructed graph. The metrics above assess G as a complete model; the following quantities ask whether combining operations adds value beyond the original graph and simpler alternatives.

Quality change and tail damage. For a held-out set of text sequences D_{text}, define the per-sequence negative log-likelihood change $\delta_G(x) = \ell(G, x) - \ell(G_0, x)$. Because each $\ell(G, x)$ is a mean over the scored tokens in sequence x, $\delta_G(x)$ remains measured in nats per scored token. We then give every sequence equal weight in $\Delta(G) = |D_{\text{text}}|^{-1} \sum_{x \in D_{\text{text}}} \delta_G(x)$. Positive values indicate deterioration. Because improvements on some sequences can offset severe losses on others, we also define $Q_{0.95}(G) = \text{Quantile}_{0.95}\{\delta_G(x) : x \in D_{\text{text}}\}$, the loss-change threshold exceeded by approximately five percent of sequences. Mean change, tail change, and task accuracy answer different questions.

Loss interactions. Let $\mathcal{A}(G)$ be a fixed set of edits whose joint application to G_0 produces G, and let $G^{(e)}$ be the graph obtained by applying only edit $e \in \mathcal{A}(G)$ to G_0. We define $I(G) = \Delta(G) - \sum_{e \in \mathcal{A}(G)} \Delta(G^{(e)})$. A positive value means joint damage exceeds the sum of individual damage, while a negative value means less-than-additive damage. Neither sign alone establishes improvement over G_0 or over a single-operation graph.

Joint resource savings. Let a resource metric m map a graph to a positive measured value under a fixed workload and measurement protocol. Define $r_m(G) = m(G)/m(G_0)$; a ratio below one indicates a reduction in that resource. For two resource metrics m_1 and m_2 selected from the deployment requirements, simultaneous savings require $r_{m_1}(G) < 1$ and $r_{m_2}(G) < 1$. Constituent-only

graphs must still be reported to show how much of each saving survives composition. Appendix C illustrates the distinction: the tested mixture reduces memory and latency relative to G_0, but incurs substantial quality loss and does not outperform direct skipping at a similar memory level.

Performance under a resource budget. Let $q(G)$ be a task-performance measure where larger is better and $c(G)$ a declared scalar cost where smaller is better. A cost-reduction objective with tolerance $\epsilon \geq 0$ requires $q(G) \geq q(G_0) - \epsilon$ and $c(G) < c(G_0)$. Let $\mathcal{B}_{\text{single}}$ contain G_0 and every evaluated single-operation alternative. For a budget b under which at least one graph in $\mathcal{B}_{\text{single}}$ is feasible, $C(G; b) = q(G) - \max_{G' \in \mathcal{B}_{\text{single}},\, c(G') \leq b} q(G')$ compares a feasible G with the best evaluated baseline under the same budget. A positive value is specific to that task, cost, budget, baseline set, and adaptation allowance; it is not global optimality over untested graphs.

When several dimensions matter simultaneously, the evaluated Pareto frontier contains candidates for which no other tested graph is at least as good on every reported dimension and strictly better on one. This frontier preserves measured trade-offs, but remains limited to the tested candidate set.

4.3 Model Selection and Evaluation Protocol

Candidate graphs and baselines use matched selection and adaptation allowances. Adaptation, when permitted, uses training data D_{train}; candidate selection uses disjoint calibration data D_{cal} and resource measurements under the target workload; final task, mean-loss, tail, and composition results use held-out data D_{test} that do not guide selection. Every resource comparison uses the same workload for G and G_0, reports relevant uncertainty, and retains the original and simpler graphs as possible outcomes. Missing required evidence leaves the deployment decision unresolved rather than counting as success. Appendix B supplies the full evaluation record, edit vocabulary, calibration-only scores, and adaptation accounting.

5 Future Research Directions

The alternatives above make progress depend on finding reconstructions whose benefits justify their quality and preparation costs. Comparable benchmarks would make candidate trade-offs easier to assess, and objective-driven selection would use that evidence to choose useful graphs. Reliability tests must then examine consequential behavior beyond average quality, while explanations of information flow could improve predictions of where edits succeed or fail. These are research directions beyond the present evidence, not claims that the necessary methods already exist.

5.1 Build Comparable Benchmarks for Execution Graphs

Comparable benchmarks are needed to determine whether an apparent advantage comes from the graph or from different evaluation conditions. Benchmark developers should evaluate single operations and mixtures on the same pretrained checkpoints, with fixed calibration and test partitions for candidate selection and final comparison. Capability tests should include language modeling and downstream behavior; resource tests should cover the hardware and workloads relevant to the claimed benefit.

Such benchmarks should report capability gains and losses alongside measured cost changes. Comparisons with constituent-only alternatives should state the search and adaptation allowances, so extra tuning or omitted measurements are not mistaken for a benefit of composition. The evaluation principles in Section 4 provide a starting point for this community effort.

5.2 Select Graphs for Particular Models and Objectives

Comparable evaluation supplies a basis for choosing candidates; search methods must turn that evidence and a deployment objective into an executable graph. Researchers can study higher task accuracy within a latency budget or lower memory use within a quality tolerance, specifying the selected layers, parameter reuse, merge rules, and adaptive post-training protocol. The quality cost observed in resource composition makes adaptation part of this choice: which adaptation objectives and training budgets best preserve quality at a given resource profile?

Search methods must account for up-front costs as well as the benefits expected over deployment. Screening candidates using the harmful-edit patterns in Section 3 offers one possible way to reduce these costs. Future work should test whether using that information reduces search expense without discarding valuable candidates, and whether the resulting search procedure transfers to new checkpoints. The aim is to find useful graphs efficiently, beyond merely expanding the set of representable graphs.

5.3 Study Reliability After Editing

A graph selected for average quality and cost can still change consequential behaviors. Reliability studies should therefore compare rare failures, output formatting, factual consistency, and behavior on unfamiliar inputs before and after reconstruction. Comparing these outcomes again after adaptive post-training would distinguish changes that persist from harmful changes that adaptation mitigates. This evaluates a different requirement from the average-quality objective used to select the graph.

For instruction-tuned models, refusal behavior and safety-relevant tasks are further instances of this requirement. They need dedicated evaluations because the loss and task tests reported here do not measure these behaviors. Including those outcomes would make the deployment decision reflect the consequences of editing, not only its average score.

5.4 Explain Changes in Information Flow

The changes in candidate preference across checkpoints leave a further question: what about the computation makes an edit succeed or fail? Mechanistic studies can investigate which representations change after skipping or repetition, and which serial dependencies branching removes. Researchers can intervene on a specified intermediate state and measure the resulting change in task behavior, testing whether a proposed dependency contributes to performance rather than merely correlating with a layer score.

To contribute to graph selection, an explanation should predict success or failure on a held-out model or task, rather than only redescribe a measured result. Testing those predictions would establish whether the explanation improves on trial-and-error search. An explanation supported by evidence of mismatched layer interfaces would also motivate a different test: pretraining models under alternative graphs to determine whether learning those interfaces improves their trade-offs. That question requires new training evidence and is not answered by post-training reconstruction.

6 Conclusion

We advocate reconstructing pretrained models as execution graphs by composing post-training structural edits and evaluating the adapted model across its intended capabilities and resources. The resource-composition example shows how shared-weight replacement and skipping can bring together memory and execution savings. The useful question is whether a combination offers the resource profile a deployment needs, not whether it beats each constituent on every metric.

The same example also shows why resource savings cannot decide deployment on their own: quality decreases, and simple skipping remains competitive. Together with the variation across models and selection rules, this evidence supports evaluating the complete adapted graph against its objective and feasible simpler alternatives. Execution graphs provide a common description of the candidate computation; capability and resource measurements determine whether that candidate is worth using.

AI USE STATEMENT

We used generative AI tools to assist with literature organization, experimental and figure-generation code, data analysis, and manuscript drafting and revision. The authors reviewed all AI-assisted text and code, verified reported measurements against saved result files, and compiled and visually inspected the final paper. We take responsibility for the complete content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

REFERENCES

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.20672.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, et al. Lessons from the trenches on reproducible evaluation of language models, 2024. URL https://arxiv.org/abs/2405.14782.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI*, 2020. URL https://arxiv.org/abs/1911.11641.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers, 2026. URL https://arxiv.org/abs/2605.23872.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Róbert Csordás, Christopher D. Manning, and Christopher Potts. Do language models use their depth efficiently?, 2025. URL https://arxiv.org/abs/2505.13898.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2025. URL https://arxiv.org/abs/2403.17887.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*, 2021. URL https://arxiv.org/abs/2009.03300.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

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 *SOSP*, 2023.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? learning program-of-layers in LLMs, 2026. URL https://arxiv.org/abs/2606.06574v2. Version 2.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2211.09110.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *ACL*, 2022. URL https://arxiv.org/abs/2109.07958.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect, 2024. URL https://arxiv.org/abs/2403.03853.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017. URL https://arxiv.org/abs/1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018. URL https://arxiv.org/abs/1809.02789.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*, 2016. URL https://arxiv.org/abs/1606.06031.

Qwen Team. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI*, 2020. URL https://arxiv.org/abs/1907.10641.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters, 2025. URL https://arxiv.org/abs/2407.09298.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *NAACL*, 2019. URL https://arxiv.org/abs/1811.00937.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, 2017. URL https://arxiv.org/abs/1707.06209.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *ACL*, 2019. URL https://arxiv.org/abs/1905.07830.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *OSDI*, 2024. URL https://arxiv.org/abs/2401.09670.

A Alternative Views and Scope

The experiments support considering composed graphs, but do not make composition the preferred choice for every application. Adopting one requires a reason to depart from the pretrained chain, a benefit beyond an adequate single operation, and enough deployment value to justify preparation costs. The alternatives below examine each of these conditions and specify when retaining a simpler model is the better decision.

A.1 The Pretrained Order May Be Essential

A pretrained layer has learned to process states produced by a particular sequence of earlier layers. Reordering or branching changes that input and can damage task performance; experiments on layer order report larger degradation on some reasoning tasks than on other benchmarks (Sun et al., 2025). Preserving the chain is therefore a strong default when the intended task depends on computations disrupted by editing.

This concern makes quality preservation a condition for adopting an edited graph, not an assumption of the framework. Adaptation to the edited computation is one way to address the changed interfaces, but the resource-composition case in Appendix C still incurs a quality cost alongside its memory and time savings. Whether a different adaptation objective, budget, or set of updated parameters can better preserve quality at those resource levels remains an empirical question. Until an edited candidate meets the application's requirements, retaining the original chain is a valid outcome of evaluation.

A.2 A Single Operation May Already Be Enough

If layer skipping meets an application's quality and latency targets, a more elaborate graph may offer little value. Combining operations introduces more design choices and implementation work, and may require additional adaptation. A useful combination must provide a benefit worth those costs.

This alternative is part of the proposed candidate space, not a case the framework must exclude. As specified in Section 4.2, compare a mixture with feasible constituent-only alternatives under a stated budget. Prefer the simpler model when the mixture adds no meaningful benefit. Composition remains worth investigating when a single operation leaves an unmet resource or capability requirement that another operation might address.

A.3 Search and Evaluation Could Cost More Than They Save

Even a useful edited graph may be uneconomical to find and prepare. Expanding the candidate space, applying post-training repair, and developing specialized execution kernels all require resources; for a briefly used model, these up-front costs may exceed the per-request savings. The same proportionality applies to evaluation: a narrowly targeted deployment may need only a focused task suite and the latency measurement relevant to its objective.

Search and evaluation should therefore scale with the intended deployment. Reuse candidate evidence and test simple baselines before investing in more complex alternatives. The harmful-skip results in Section 3 suggest screening as another possible way to limit search, although its effect on total search cost remains unmeasured. Focused deployment claims can use focused tests; broader capability or resource claims need correspondingly broader evidence. The research directions below address how to improve these choices through comparable evaluation and less costly graph selection.

B DepthGraphAudit Evaluation Specification

This appendix specifies how to turn the evaluation principles in Section 4 into reproducible comparisons. It first separates the research questions and required evaluation fields, then defines the static edits and calibration scores, and finally explains the accounting used in the interface-repair diagnostics. Study-specific model, data, and execution settings appear in the corresponding result appendices.

B.1 Four Related Research Questions

The graph representation separates four questions that require different evidence. **Module localization** asks whether a layer or another unit is especially sensitive to intervention. **Interface compatibility** asks whether a hidden state remains usable after a substitution or reconnection. **Edit performance** asks whether a specified skip, repeat, swap, split, route, or merge improves the intended quality–resource trade-off. **Pretraining under an alternative graph** asks whether that training structure produces a competitive model under matched quality, data, compute, parameter, and serving budgets. The empirical studies in this paper address the first three questions by reconstructing existing pretrained checkpoints, with or without adaptive post-training. They do not evaluate pretraining under the alternative graph.

B.2 The Seven Evaluation Fields

Once the research question is fixed, the evaluation record must identify the change, the intended benefit, and the evidence that could favor or reject it. The following seven fields make those choices explicit before held-out testing.

1. **Measured object:** Which node, edge, or hidden-state interface changes?
2. **Graph edit:** Which topology operation is proposed?
3. **Quality and resource:** Which behavior must be retained, and which cost is expected to improve?
4. **Selection score:** Which calibration-only evidence selects the candidate?
5. **Matched comparison:** Which model and budget define the comparison?
6. **Failure condition:** Which prespecified result favors the original model?
7. **Execution and repair:** Why is the graph executable, and what adaptation budget is allowed?

The quality and resource fields are selected for the stated question rather than filled from a universal metric list. Quality may use negative log-likelihood, divergence, accuracy, generation success, or a task-specific score. Cost may use stored or active parameters, floating-point operations, latency, key–value cache allocation, peak memory, repair tokens, or wall-clock time. Reporting these dimensions separately preserves the trade-off the evaluation is intended to resolve.

B.3 Static Edit Vocabulary

To compare edits, the evaluation record must specify the computation each edit changes. Here a block or module is a Transformer layer, and the original model retains its serial layer dependencies. *Repeat* executes a module again. *Skip* removes a block from the active path. *Swap* exchanges the execution order of an adjacent pair. *Branch–merge* evaluates two residual updates from a shared incoming state and merges them. *Interface repair* trains only a declared reconnection map while keeping checkpoint parameters frozen. Dynamic routing and pretraining under an alternative graph are separate settings.

For the pairwise operations, let i index a candidate adjacent pair, A_i and B_i be its first and second block mappings, and h_i be the hidden state produced by the unchanged prefix. These block mappings are distinct from the edit sets labeled in the main text. Swap replaces the original serial computation $B_i(A_i(h_i))$ with $A_i(B_i(h_i))$. Branch–merge instead produces the merged state $\hat{z}_i$ from residual updates Δ_{A_i} and Δ_{B_i}:

$$\hat{z}_i = h_i + \Delta_{A_i} + \Delta_{B_i}, \qquad \Delta_{A_i} = A_i(h_i) - h_i, \quad \Delta_{B_i} = B_i(h_i) - h_i.$$

These equations define the edited computation. The branch–merge implementation evaluated here executes both branches sequentially.

Six-layer composition example. Figure 6 illustrates how several static operations change one execution graph. It skips F_2, sends $u = F_1(h)$ to both F_3 and F_4, adds their residual updates to u, and invokes F_5 twice before F_6. Its computation is

$$F_6(F_5(F_5(P_{3,4}(F_1(h))))) .$$

The example uses five distinct pretrained layers and six layer calls, with at most five sequential layer calls on an input-to-output path. These counts describe graph structure, not measured quality or latency.

B.4 Calibration-Only Selection Scores

Candidate-selection scores order eligible edits before their joint held-out evaluation. Researchers compute the scores on calibration sequences D_{cal}; the test set does not contribute to this ordering. For a single-block candidate, let i index its original layer position, M_i denote its original mapping, $h_i(x)$ its input state for sequence x, and $\tau > 0$ a small denominator floor. For skipping, the relative-update score for block i is

$$d_i^{\text{cal}} = \frac{1}{|D_{\text{cal}}|} \sum_x \frac{\|M_i(h_i(x)) - h_i(x)\|_2}{\max(\|M_i(h_i(x))\|_2, \tau)}.$$

The sum ranges over calibration sequences. Low-update blocks provide plausible candidates; high-update blocks define the control intended to expose harmful skipping, whose actual damage is measured on held-out data.

A second local score, ShortGPT Block Influence (BI), measures the angular change between a block's input and output (Men et al., 2024). Let $\mathcal{T}_{\text{cal}}$ contain the non-padding sequence–token pairs (x, t) in the calibration data, and let the superscript (t) select the hidden vector at token position t. Exact Block Influence is

$$b_i^{\text{cal}} = 1 - \frac{1}{|\mathcal{T}_{\text{cal}}|} \sum_{(x,t)\in\mathcal{T}_{\text{cal}}} \cos\Big(h_i^{(t)}(x), M_i(h_i(x))^{(t)}\Big).$$

For non-skip edits, the low/high update labels rank operation-specific local discrepancies rather than the skip relative-update score. Repetition compares the additional update from the second application with the first application's update. Swapping and branch–merge compare the edited output with the original serial output, normalized by the serial residual update. These scores use local outputs before the unchanged suffix runs.

To measure change beyond the edited region, suffix hidden-state cosine compares the final hidden states after subsequent layers have processed the edit. It ranks candidates by one minus the mean tokenwise cosine similarity between the original and edited final hidden states, after the unchanged suffix has run and before final normalization. Smaller drift is preferred. Unlike Block Influence, it compares the complete paths rather than a block's own input and output.

Hidden-state differences are not the only basis for selection. Direct calibration damage evaluates each edit alone and ranks it by the change in mean calibration NLL, with smaller changes preferred. Depth and random controls test whether diagnostic information improves on simpler choices: middle-depth selection ranks start positions by distance from the midpoint of the eligible range, with ties favoring the earlier position, and random selection uses the recorded partition seed.

Once a rule has ordered individual candidates, joint non-skip configurations take them in ranked order while excluding overlapping layer intervals. The chosen edits are then applied together and the resulting graph is evaluated on independently partitioned held-out sequences. This final step tests the complete graph rather than reusing individual-candidate scores as its outcome.

B.5 Reporting Repair

Repair comparisons need to distinguish the effect of changing the graph from the additional training used to recover quality. Interface repair produces a new model by training a reconnection map under an explicit adaptation budget. Its record should specify the reconnection point, interface class, trainable parameter count, processed hidden-token states, wall-clock time, mean and tail held-out damage, and downstream performance. The best values over the held-out repair grid describe the quality recovered within that grid; they are not used to select a deployment model.

B.6 Recovery in the Earlier Interface Diagnostics

To interpret the repair record, the interface-only studies express the loss recovered relative to the damage present before repair. Let Δ_A^0 and Δ_A^B denote loss damage before and after adaptation with

budget B. Their recovery measure is

$$R_A(B) = \frac{\Delta_A^0 - \Delta_A^B}{\max(\Delta_A^0, \tau)}$$

for positive initial damage, where $\tau > 0$ stabilizes the denominator. Above the floor, a value of one indicates recovery to the unedited reference. These budgets count processed hidden-token states and include repeated exposure during optimization. This diagnostic is distinct from the joint resource savings evaluated after common LoRA post-training.

The recovery measure makes partial improvement visible without treating repair as a guarantee. Some edits recover part of their lost quality, but the gain is not monotonic in budget, and repeat and swap repairs retain residual damage (Figure 15; Table 31). Recovery must therefore be interpreted alongside the remaining damage and adaptation cost when choosing a deployment trade-off.

C Resource Composition After Common Adaptation

This appendix specifies the graphs, adaptation, and measurements used for the composition-specific comparison in Section 4.2. We first evaluate Qwen2.5-1.5B and then transfer the selected invocation programs unchanged to Qwen2.5-7B (Qwen Team, 2024), with common LoRA post-training (Hu et al., 2021). A complementary mixed-operation comparison asks whether mixtures improve quality at the original layer-call budget. The sharing and skipping patterns are established design ingredients; the experiments test their consequences in complete models.

C.1 Execution Graphs and Adaptation

The graph programs separate weight reuse from skipped execution so that their individual and combined costs can be compared. Layer indices in this appendix start at zero. Pattern A replaces four distinct consecutive layers by three repetitions of the first two layers in that group. On 1.5B, we consider start positions 8 and 12. Pattern B skips layers 18, 20, 22, and 24, outside both sharing regions. A+B applies both patterns. The selected sharing group starts at 12, chosen by adapted calibration loss among layouts with the same resource counts.

The controls isolate alternative uses of the same resources. Skip-2 removes layers 22 and 24 and matches A+B's call count. Skip-6 removes B's layers and the two distinct layers omitted by A, matching A+B's resident-layer count. Repeat-only retains every original layer and adds one further call to each of A's first two layers, matching A's call count. A, A+B, Skip-6, and Repeat-only each have a variant at both candidate sharing locations; Linear, B, and Skip-2 do not depend on that location. In total, the primary 1.5B data-order seed adapts eleven candidate graphs; fixing the selected layout gives seven final comparisons.

The candidate graphs are compared after the same adaptation allowance, including the linear reference. Each receives 131,072 input tokens, with rank-eight LoRA of scaling factor 16 and zero dropout on query and value projections. AdamW uses learning rate 2×10^{-4}, no weight decay, gradient clipping at one, and eight sequences of length 256 per update. The data pool contains 1,024 WikiText training sequences, followed by 32 separate calibration sequences. The primary data-order seed is 7; seed 17 repeats the fixed selected comparisons, while LoRA initialization remains fixed at seed 317.

Quality is measured after layout selection, using 256 held-out WikiText sequences and 512 ARC-Challenge questions. Neither evaluation set selects a graph. For ARC-Challenge, the predicted answer has the largest conditional log-likelihood divided by its character count. This length-normalized scoring rule is fixed for every graph and both seeds; it is not raw summed-log-likelihood accuracy.

For the 7B transfer, all seven programs use the same zero-indexed layer positions selected on 1.5B; no target-model reselection is allowed. Both checkpoints have 28 layers. Training, calibration, and test token sequences and the multiple-choice examples are identical, verified by their content hashes. The target model receives the same adaptation protocol for both data-order seeds.

C.2 Actual Weight Residency and Cached Execution

Resource measurements must reflect the graph actually deployed, including weight residency and per-invocation cache state. Each measurement loads the selected graph in a separate process on one NVIDIA L40. Unused layers are omitted before transfer to the device. Repeated invocations share layer weights and LoRA parameters but receive distinct cache slots. The base weights use BF16 and the unmerged LoRA parameters remain in float32, preserving the computation used in quality evaluation. We verify that deployment matches the trained invocation program and that cached decoding matches full-sequence computation in an FP32 check.

All models process the same input tokens and the same fixed continuation; generation does not stop early. Only last-position output logits are computed for serving. Each workload uses three warm-up runs and seven measured runs, with two rounds in opposite graph orders. The transferred model's serving comparison uses batch sizes one and four at context length 2048, with 32 fixed continuation tokens and the same opposite-order rounds. The L40 retains small background trading processes, so these are controlled measurements on shared hardware. Peak memory refers to PyTorch allocated memory; driver-level process residency is also recorded. Cache memory counts key and value tensors for every invocation.

Table 1 preserves the individual rounds for the initial 1.5B workloads, while Tables 2–4 report the corresponding quality and transfer comparisons. The long-context decode ratios vary substantially between rounds, so a representative median is not evidence of a uniform speedup across workloads and repetitions. The predeclared stronger deployment screen required both peak-memory and decode-time ratios to be at most 0.97 in each round. On 1.5B, A+B does not meet that screen, although its representative-workload measurements show the joint resource-saving direction discussed in the main text. This distinction separates an illustrative composition result from a stronger deployment claim.

Table 1: Resource ratios for Qwen2.5-1.5B by workload and measurement round. Lower is better. Memory ratios agree across the two rounds; prefill and decode ratios are retained separately. All ratios use the linear model measured in the corresponding round.

Graph	Batch	Context	r_M	$r_{P,1}$	$r_{P,2}$	$r_{T,1}$	$r_{T,2}$
Linear	1	128	1.000	1.000	1.000	1.000	1.000
Linear	1	2048	1.000	1.000	1.000	1.000	1.000
Linear	4	2048	1.000	1.000	1.000	1.000	1.000
Linear	4	8192	1.000	1.000	1.000	1.000	1.000
A	1	128	0.940	1.112	1.074	1.108	1.076
A	1	2048	0.945	1.078	1.076	1.091	1.037
A	4	2048	0.956	1.076	1.077	1.100	1.044
A	4	8192	0.981	1.078	1.077	1.117	0.653
B	1	128	0.880	0.871	0.856	0.872	0.862
B	1	2048	0.885	0.854	0.852	0.860	0.848
B	4	2048	0.895	0.860	0.856	0.922	0.842
B	4	8192	0.918	0.862	0.863	0.871	0.521
A+B	1	128	0.821	1.012	0.929	1.008	0.926
A+B	1	2048	0.829	0.934	0.922	0.988	0.909
A+B	4	2048	0.852	0.928	0.930	0.989	0.914
A+B	4	8192	0.899	0.931	0.933	0.988	0.603
Skip-2	1	128	0.940	0.941	0.940	0.941	0.949
Skip-2	1	2048	0.942	0.926	0.926	0.926	0.909
Skip-2	4	2048	0.948	0.931	0.928	0.927	0.921
Skip-2	4	8192	0.959	0.937	0.931	0.940	0.574
Skip-6	1	128	0.821	0.795	0.787	0.794	0.783
Skip-6	1	2048	0.827	0.780	0.782	0.784	0.761
Skip-6	4	2048	0.843	0.786	0.776	0.784	0.773
Skip-6	4	8192	0.878	0.784	0.784	0.805	0.484
Repeat-only	1	128	1.000	1.069	1.061	1.071	1.068
Repeat-only	1	2048	1.001	1.080	1.075	1.110	1.049
Repeat-only	4	2048	1.004	1.078	1.074	1.112	1.057
Repeat-only	4	8192	1.011	1.075	1.072	1.127	0.716

C.3 Quality at the Measured Resource Points

The measured savings are useful only if their quality cost is acceptable. Table 2 reports 1.5B quality for both data-order seeds against a prespecified screen allowing at most a five-percent relative increase in mean NLL and a two-percentage-point accuracy drop. A+B does not meet this screen. Direct Skip-6 on 1.5B also has lower NLL and shorter execution than A+B at a similar memory level; the small accuracy difference between these two graphs changes direction across seeds. The simple control therefore remains a necessary comparison even when resource savings coexist.

These results leave adaptation quality as an open design question. Different objectives, trainable modules, or budgets could change the quality of a given resource point, but the present experiment does not identify insufficient adaptation as the cause of the measured loss. The resource example establishes what is saved by this graph composition; the quality and control measurements specify what remains to be improved.

Table 2: Quality after common adaptation for the selected resource graphs on Qwen2.5-1.5B. Brackets give descriptive paired-bootstrap 95% intervals over sequences or questions. Accuracy differences are percentage points; graph selection was fixed using the primary seed's calibration data.

Graph	Seed	NLL	ΔNLL [interval]	ΔAcc. [interval]
Linear	7	2.603	0.000 [0.000,0.000]	0.00 [0.00,0.00]
Linear	17	2.603	0.000 [0.000,0.000]	0.00 [0.00,0.00]
A	7	2.752	0.149 [0.143,0.154]	-7.62 [-10.94,-4.49]
A	17	2.752	0.149 [0.144,0.155]	-9.18 [-12.30,-6.25]
B	7	3.356	0.753 [0.735,0.770]	-11.13 [-15.04,-7.23]
B	17	3.357	0.754 [0.736,0.772]	-13.28 [-17.38,-9.18]
A+B	7	3.581	0.979 [0.959,0.998]	-15.63 [-19.73,-11.52]
A+B	17	3.585	0.982 [0.962,1.001]	-15.82 [-20.12,-11.52]
Skip-2	7	3.047	0.444 [0.430,0.459]	-4.30 [-7.81,-0.78]
Skip-2	17	3.048	0.445 [0.430,0.460]	-3.71 [-7.23,-0.20]
Skip-6	7	3.499	0.896 [0.877,0.915]	-15.04 [-18.95,-11.13]
Skip-6	17	3.501	0.898 [0.879,0.918]	-16.02 [-20.12,-11.91]
Repeat-only	7	2.622	0.019 [0.017,0.021]	-2.15 [-4.49,0.20]
Repeat-only	17	2.624	0.021 [0.019,0.023]	-3.32 [-5.66,-0.98]

The transfer tests whether fixing the invocation programs preserves this trade-off on the larger model. A+B meets the stricter resource criterion on 7B at both registered 2048-token workloads (Table 3), but fails the quality screen for both data-order seeds (Table 4). Skip-6 again has lower NLL and shorter execution with no greater peak memory; its accuracy point estimate is also higher in both runs. The fixed topology therefore reproduces the resource-saving direction without resolving the quality cost.

Not every transferred graph has the same direction of task response. Repeat-only has similar NLL damage on both models, yet its accuracy-change point estimates are negative on 1.5B and positive on 7B for both data-order seeds. Subtracting the 1.5B edit-versus-linear accuracy change from the corresponding 7B change gives 5.08 percentage points [2.15,8.20] for seed 7 and 5.27 [2.34,8.20] for seed 17. These descriptive paired-bootstrap 95% intervals resample the same questions across both models. They characterize the fixed model pair rather than establishing a general cross-model response.

Table 3: Resource measurements for the seven fixed programs transferred to Qwen2.5-7B without reselection. Ratios use the adapted 7B linear model measured in the same round and workload. r_M, r_P, and r_T denote peak allocated memory, prefill time, and per-token decode time; the second subscript identifies measurement round 1 or 2. All workloads generate the same 32 continuation tokens on the L40.

Graph	Batch	Context	$r_{M,1}$	$r_{M,2}$	$r_{P,1}$	$r_{P,2}$	$r_{T,1}$	$r_{T,2}$
Linear	1	2048	1.000	1.000	1.000	1.000	1.000	1.000
Linear	4	2048	1.000	1.000	1.000	1.000	1.000	1.000
A	1	2048	0.941	0.941	1.078	1.075	1.055	1.052
A	4	2048	0.947	0.947	1.076	1.071	1.059	1.067
B	1	2048	0.880	0.880	0.854	0.853	0.900	0.838
B	4	2048	0.886	0.886	0.851	0.848	0.894	0.852
A+B	1	2048	0.821	0.821	0.927	0.927	0.908	0.928
A+B	4	2048	0.832	0.832	0.932	0.924	0.926	0.928
Skip-2	1	2048	0.940	0.940	0.929	0.921	0.898	0.904
Skip-2	4	2048	0.943	0.943	0.930	0.923	0.912	0.924
Skip-6	1	2048	0.820	0.820	0.778	0.778	0.796	0.757
Skip-6	4	2048	0.828	0.828	0.779	0.775	0.803	0.771
Repeat-only	1	2048	1.001	1.001	1.076	1.078	1.064	1.047
Repeat-only	4	2048	1.002	1.002	1.071	1.075	1.061	1.050

Table 4: Quality of the fixed programs on Qwen2.5-7B after the same 131,072-token adaptation allowance. Seed indexes training-data order; LoRA initialization is fixed. NLL and its change are in nats per scored token; accuracy changes are percentage points on the same 512 ARC-Challenge questions. Brackets give descriptive paired-bootstrap 95% intervals over 256 text sequences or 512 questions. Each change uses the adapted linear model with the same data-order seed.

Graph	Seed	NLL	ΔNLL [interval]	ΔAcc. [interval]
Linear	7	2.303	0.000 [0.000,0.000]	0.00 [0.00,0.00]
Linear	17	2.306	0.000 [0.000,0.000]	0.00 [0.00,0.00]
A	7	2.432	0.129 [0.123,0.134]	-1.76 [-4.69,1.17]
A	17	2.423	0.118 [0.113,0.122]	-1.95 [-4.88,0.98]
B	7	2.869	0.566 [0.551,0.583]	-12.11 [-16.02,-8.20]
B	17	2.868	0.562 [0.546,0.579]	-13.09 [-16.99,-9.18]
A+B	7	3.026	0.724 [0.705,0.742]	-15.23 [-19.53,-10.94]
A+B	17	3.025	0.719 [0.701,0.738]	-16.02 [-20.31,-11.72]
Skip-2	7	2.651	0.348 [0.335,0.362]	-7.42 [-11.13,-3.71]
Skip-2	17	2.653	0.347 [0.334,0.361]	-7.03 [-10.55,-3.52]
Skip-6	7	2.976	0.673 [0.656,0.691]	-13.87 [-18.16,-9.57]
Skip-6	17	2.975	0.669 [0.652,0.687]	-15.23 [-19.34,-10.94]
Repeat-only	7	2.322	0.020 [0.017,0.022]	2.93 [0.98,4.88]
Repeat-only	17	2.324	0.019 [0.016,0.021]	1.95 [-0.39,4.30]

The resource-composition example reduces the combined graph's layer-call count. A separate question is whether mixtures improve quality when they retain the original call budget. This comparison uses Qwen2.5-1.5B and Mistral-7B, with 49 candidates adapted per model under the same rank-eight query/value LoRA protocol and 131,072-token primary budget. Each model selects its graphs by adapted calibration NLL. The selected skip–repeat and skip–repeat–branch graphs retain the original model's layer-call count. The single-operation comparator is the lowest-calibration-loss eligible graph with one operation type and no more calls; it is branch–merge in both models. Evaluation uses 256 held-out text sequences. Both mixtures have higher held-out loss than their adapted linear and single-operation references (Table 5). Holding these selections fixed, a second seed (17 rather than 7) and a shorter budget of 32,768 tokens retain increased loss relative to their matched linear references. These model-specific selections therefore provide no quality advantage in the tested setting; they are not a transfer of identical layer indices.

Table 5: Mixed-operation quality controls after common adaptation. All entries subtract comparator NLL from mixture NLL, in nats per scored token; positive values mean the mixture has higher loss. Brackets give paired-bootstrap 97.5% intervals against the linear reference, using the wider interval level for the two primary mixture comparisons. Differences from the calibration-selected single-operation graph are point estimates, without transferring the linear-reference intervals.

Model	Mixture	Versus linear [interval]	Versus single
Qwen2.5-1.5B	Skip–repeat	$+0.07444$ $[0.06990, 0.07910]$	$+0.05175$
	Skip–repeat–branch	$+0.10079$ $[0.09566, 0.10597]$	$+0.07811$
Mistral-7B	Skip–repeat	$+0.06298$ $[0.05801, 0.06816]$	$+0.05317$
	Skip–repeat–branch	$+0.08323$ $[0.07725, 0.08931]$	$+0.07342$

D Evaluation-Requirement Ablation and Example Cases

This appendix tests whether adding evaluation requirements changes the interpretation of the same candidate edits. The analysis fixes 21 skip candidates with $k = 2$ evaluated under data-partition seed 7 across Qwen2.5-1.5B, Mistral-7B, and Gemma2-2B. Each entry pairs a checkpoint with a selection rule; different rules may select the same layer set. The analysis combines held-out mean damage, per-sequence tail damage, downstream performance, and cache-enabled serving measurements. The nominal thresholds are $\Delta\text{NLL} \leq 1.0$, $Q_{0.95}(\delta) \leq 1.0$, accuracy drop at most 0.05, and median prefill and decode ratios at most 1.0. Here the quantile in the tail condition denotes the per-sequence statistic $Q_{.95}(G)$ defined in Section 4.

To distinguish a measured failure from insufficient evidence, a candidate meets the goal only when every required measurement is present and passes. Any measured failure favors the original model; otherwise, a missing requirement leaves the decision unresolved. Table 6 and Figure 9 apply this rule as requirements are added: fewer candidates remain supported, through both additional failures and missing measurements. We next test each requirement's contribution and the dependence on the chosen thresholds.

Table 6: Nested decision-requirement ablation. Counts are candidate edits, not model-quality scores.

Required evidence	Meets goal	Original favored	Missing measure
Mean	18	3	0
Mean + tail	17	4	0
Mean + tail + task	12	6	3
Mean + tail + task + cost	6	6	9

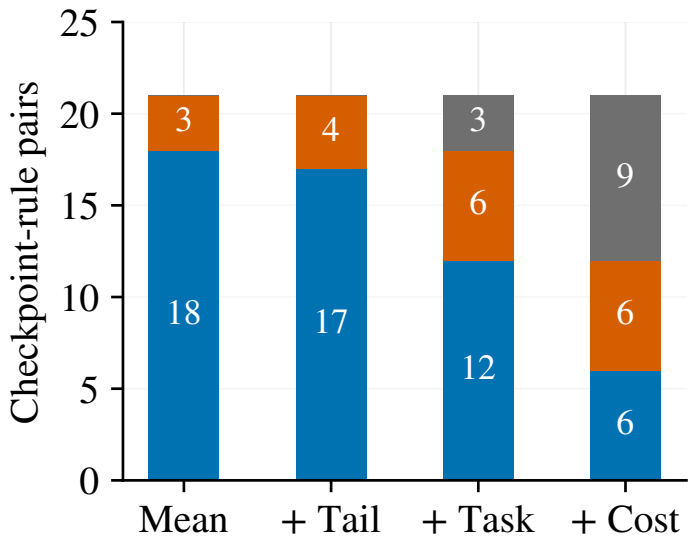
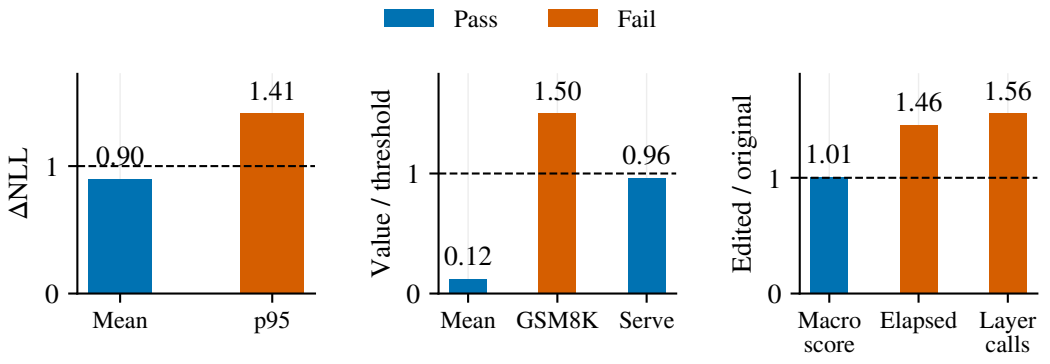


Figure 9: Adding tail loss, task performance, and serving cost changes which edits meet the stated goal. The gray group lacks a required measurement.

Removing one requirement while retaining the others tests whether it changes the decision beyond the remaining measurements. Omitting the tail requirement admits one edit that fails the tail test. Omitting task performance changes two decisions: one task failure is admitted, while another becomes unresolved because its serving measurement is missing. Omitting cost admits six candidates whose serving effect was not measured. Mean loss has no independent leave-one-out effect in this matched set because every mean failure also fails the tail requirement. Thus, different requirements can change the decision for different reasons, while overlapping failures can make one requirement redundant in this particular set.

To check whether this conclusion depends on the nominal thresholds, the threshold grid crosses mean-loss limits of 0.25, 0.5, and 1.0; tail-loss limits of 0.5, 1.0, and 2.0; accuracy-drop limits of 0.03, 0.05, and 0.10; and serving-ratio limits of 0.98 and 1.0. The same candidate entries and missing-data rule are used throughout. Across 54 prespecified threshold combinations, between two and eight candidates meet all requirements. The exact decisions depend on the threshold, but the contribution of broader evaluation is not confined to one setting.



(a) Tail requirement (b) Behavior requirement (c) Cost requirement

Figure 10: Three cases in which broader evaluation changes the interpretation. **(a)** Gemma2-2B random skipping passes the mean-loss threshold but fails the tail threshold. **(b)** Qwen2.5-7B low-update skipping passes the mean-NLL and serving checks but exceeds the specified GSM8K-drop threshold. Bars are normalized by their thresholds. **(c)** The official Qwen3-4B frozen loop slightly raises the macro task score while also increasing elapsed time and layer calls; ratios are relative to the original model. Blue denotes a passed requirement and vermilion a failed requirement.

Figure 10 shows what the additional measurements reveal in concrete cases. Random skipping on Gemma2-2B passes the mean-loss test but fails the tail test; low-update skipping on Qwen2.5-7B passes several other checks but exceeds the stated generation-accuracy-drop threshold. The frozen loop instead offers a measured quality-for-compute trade rather than a no-extra-cost improvement. These cases and the ablation show how evaluation requirements affect conclusions without proposing one threshold for every deployment.

E Literature Reporting Audit

This appendix reports the 34-work analysis summarized in Section 4. It asks which evaluation details readers can recover from the selected papers. We define the reporting fields and scoring rules, examine sensitivity to model-based reannotation, and then provide the per-work records.

E.1 Audit Codebook

The audit covers representative works from six depth-architecture families. They include depth analysis and pruning, recurrent depth, skipping and early exit, conditional computation, branch or merge methods, and adapter or editing methods. The goal is to cover different research questions rather than construct an exhaustive survey.

The codebook first asks whether the paper identifies a change and its intended benefit. *Graph edit* records whether a topology operation can be recovered, including preserving, reusing, skipping, splitting, routing, inserting, or updating a node. *Target trade-off*, labeled Target resource in Figure 11, records whether the work names the resource or capability being traded, such as latency, active compute, stored parameters, editability, or adaptation cost. These binary fields use 0/1 for absence/presence.

The remaining fields ask how the proposed change is justified and evaluated. *Selection evidence*, labeled Diagnostic in the aggregate displays, asks whether the paper reports an analysis or intervention that selects an edit. *Matched budget* asks whether the trade-off is compared under the relevant controlled resource. *Failure condition* asks whether the paper states which result would favor the original model. *Training evidence*, labeled Train-time evidence in Figure 11, asks whether trainability, executability, or the architectural training bias is discussed. These graded fields use 0=absent, 1=partial or implicit, and 2=explicit. The scores describe reporting structure rather than method quality or empirical correctness.

The 34-work audit uses one primary annotation. To assess sensitivity to the annotator, GPT-5.4 and GPT-5.5 independently reannotated a blinded, stratified 12-work subset using the same codebook. Table 7 reports their agreement on these judgments. This model-based comparison tests annotation sensitivity, not human inter-annotator agreement.

Table 7: Agreement between two independent model reannotations on the 12-work blind subset. Agreement is the fraction of papers receiving identical scores from both annotators; MAD is their mean absolute paired score difference across papers.

Field	Agreement	MAD
Graph edit	1.00	0.00
Target trade-off	1.00	0.00
Diagnostic	0.92	0.08
Matched budget	0.92	0.08
Failure condition	0.75	0.25
Training evidence	0.83	0.17

The model annotators agreed on which papers expressed a graph edit and target trade-off, but agreed less often on failure conditions and training evidence. Together with the selected scope of the corpus, this annotation sensitivity limits interpretation of the coverage rates in Figure 11: they describe the selected papers rather than providing a precise field-wide estimate.

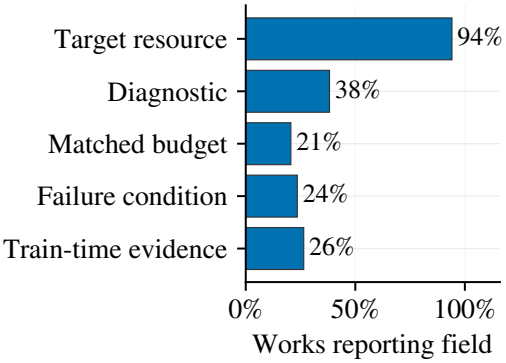


Figure 11: Reporting coverage for five evaluation fields in the 34-work literature audit. The binary target-resource field counts presence; the other graded fields count only explicit reporting, not partial or implicit reporting. The figure summarizes reporting structure rather than ranking paper quality.

E.2 Per-Work Annotations

Table 8 records the selected works and the judgments underlying the reporting audit. Family gives a more specific descriptive category than the broad groups listed above. Edit and Trade-off give qualitative descriptions rather than the binary scores used for aggregate coverage. Select., Budget, Failure, and Train. give the four graded scores defined above. A descriptive trade-off phrase alone does not establish that the paper explicitly reported that objective.

Table 8: Per-work literature reporting audit.

Work	Family	Edit	Trade-off	Select.	Budget	Failure	Train.
Transformer Layers as Painters	depth analysis	measure edges	diagnostic protocol	2	1	1	1
Do Language Models Use Their Depth Efficiently?	depth efficiency	delete skip node	active compute	2	1	1	1
What Affects the Effective Depth of Large Language Models?	depth efficiency	measure edges	effective depth	1	1	1	1
ShortGPT	depth pruning	delete skip node	active compute	2	1	1	1
The Unreasonable Ineffectiveness of the Deeper Layers	depth pruning	delete skip node	active compute	1	1	1	1
What Matters in Transformers? Not All Attention is Needed	sublayer pruning	delete skip sublayer	active compute	2	1	1	1
Tuned Lens	interface probe	measure interface	diagnostic protocol	2	0	0	0
Universal Transformers	recurrent depth	reuse node	stored params	1	1	1	2

Work	Family	Edit	Trade-off	Select.	Budget	Failure	Train.
ALBERT	parameter sharing	reuse node	stored params	1	2	1	1
Looped Transformers are Better at Learning Learning Algorithms	recurrent depth	reuse node	algorithmic generalization	1	1	1	2
Huginn	latent recurrence	reuse node	test time compute	1	1	1	2
Ouro / Looped Language Models	latent recurrence	reuse node	test time compute	1	1	1	2
Loop the Middle	middle recurrence	reuse middle node	flops quality	2	1	1	2
Hyperloop Transformers	middle recurrence	reuse middle node	flops quality	1	1	1	2
LayerDrop	dynamic depth	delete skip node	active compute	2	1	1	1
LayerSkip	early exit	delete skip suffix	latency	2	2	1	1
Learning to Skip the Middle Layers of Transformers	dynamic depth negative	delete skip middle	active compute	2	2	2	2
Reuse Transformers	reuse attention	reuse internal state	active compute	1	1	1	1
SNLP	layer parallel	split edge	latency	2	2	2	2
Sparsely-Gated Mixture-of-Experts	conditional compute	select subgraph	active params	1	1	2	1
GShard	conditional compute	select subgraph	scaling capacity	1	1	2	1
Switch Transformer	conditional compute	select subgraph	active params	1	2	2	1
GLaM	conditional compute	select subgraph	active params	1	2	2	1
Mixtral of Experts	conditional compute	select subgraph	active params	1	1	1	1
Branch-Train-Merge	model branch merge	split training path	specialization cost	1	1	1	1
Branch-Train-MiX	model branch merge	split training path	specialization cost	1	1	1	1
Branch-Solve-Merge	external branch merge	split problem path	solution quality	1	0	1	1
Parameter-Efficient Transfer Learning for NLP	adapter	insert update node	adaptation cost	1	2	1	1
AdapterFusion	adapter composition	select merge adapter	adaptation cost	1	1	1	1
AdapterDistillation	adapter compression	insert update node	stored params	1	1	1	1
Knowledge Neurons	localization	measure or update node	editability	2	0	1	0
ROME	model editing	update node	editability	2	1	2	1
MEMIT	model editing	update node	editability	2	1	2	1
Modular Memory for Continual Learning Agents	modular memory	insert update node	continual adaptation	1	1	1	2

F Frozen Graph-Edit Diagnostic Details

This appendix examines where individual frozen edits change next-token negative log-likelihood (NLL) and whether local hidden-state differences track that damage. The first comparison measures sensitivity across checkpoints and edit types; the second tests local diagnostic correlations on two checkpoints. These studies support the model-specific design argument in Section 3. Their scope is local edit tolerance, not a comparison of architectures trained under different graphs.

The diagnostic comparisons require the same edit meaning at every candidate location. For a single block M_i, skip removes the block and repeat applies it twice. For adjacent Transformer blocks A_i and B_i with prefix state h_i, swap replaces the serial computation $z_i = B_i(A_i(h_i))$ with $\tilde{z}_i = A_i(B_i(h_i))$. Branch–merge instead uses $\hat{z}_i = h_i + \Delta_{A_i} + \Delta_{B_i}$, where $\Delta_{A_i} = A_i(h_i) - h_i$ and $\Delta_{B_i} = B_i(h_i) - h_i$. These definitions fix what changes locally; the loss measurement below records the effect on the complete model.

Specifically, for edit $e \in \{\text{skip}, \text{repeat}, \text{swap}, \text{branch-merge}\}$, let $\text{NLL}_{i,e}(D)$ be next-token NLL after applying edit e to candidate i, averaging over sampled sequences and non-final token positions in D. The reported degradation is

$$\Delta_{i,e}(D) = \text{NLL}_{i,e}(D) - \text{NLL}_{\text{base}}(D),$$

where larger values are worse. Raw ΔNLL is model- and base-loss-dependent, so cross-model values should be interpreted as heterogeneity evidence rather than calibrated quality comparisons.

F.1 Extended Frozen Audit Setup

The cross-checkpoint comparison tests whether the same type of frozen edit produces similar damage at different model scales. All runs use WikiText-2 validation text, sequence length 256, 32 sampled sequences, block size 1, stride 1, and the same frozen checkpoint weights. Table 9 specifies the checkpoints and baseline losses. Interior summaries exclude the first and final adjacent pairs for pair edits where applicable. Figure 12 and Table 10 report the test-split results: within each checkpoint, branch–merge has lower mean damage than the other listed operations, while the size of the difference varies across checkpoints. This comparison supports testing each operation on its target model rather than assuming a common tolerance.

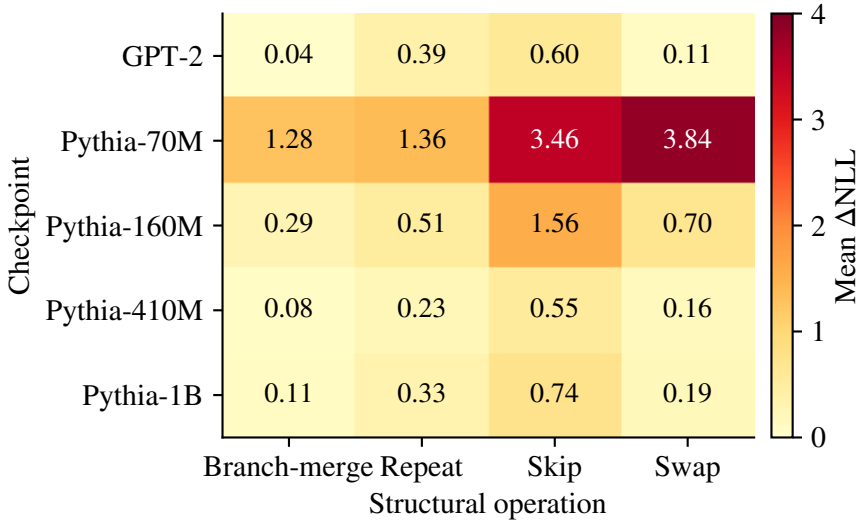


Figure 12: Frozen graph-edit degradation across models and edit types. Values are mean ΔNLL on interior candidates; larger values indicate more damage. This figure provides appendix context for the expanded static evaluation and is not a calibrated cross-model quality ranking.

Table 9: Extended frozen graph-edit setup for Figure 12. Base NLL is reconstructed as $\mathrm{NLL}_{\mathrm{edit}} - \Delta\mathrm{NLL}$ and averaged over test-split edit rows; it is included only to contextualize raw ΔNLL.

Checkpoint	Arch.	Blocks	Pairs	Seq.	Length	Base NLL
GPT-2	gpt2	12	11	32	256	3.67
Pythia-70M	gpt-neox	6	5	32	256	4.19
Pythia-160M	gpt-neox	12	11	32	256	3.68
Pythia-410M	gpt-neox	24	23	32	256	3.20
Pythia-1B	gpt-neox	16	15	32	256	3.05

Table 10: Interior mean ΔNLL values plotted in Figure 12. Larger values indicate more damage under the frozen edit.

Checkpoint	Branch–merge	Repeat	Skip	Swap
GPT-2	0.04	0.39	0.60	0.11
Pythia-70M	1.28	1.36	3.46	3.84
Pythia-160M	0.29	0.51	1.56	0.70
Pythia-410M	0.08	0.23	0.55	0.16
Pythia-1B	0.11	0.33	0.74	0.19

F.2 Two-Checkpoint Correlation Pilot

The variation in edit damage raises a further question: do local hidden-state differences track the final-model loss change? A two-checkpoint comparison on GPT-2 and Pythia-160M examines this

association for swap and branch–merge. Its local scores are relative errors

$$C_i = \frac{\|z_i - \tilde{z}_i\|}{\|z_i - h_i\| + \epsilon}, \qquad M_i = \frac{\|z_i - \hat{z}_i\|}{\|z_i - h_i\| + \epsilon},$$

averaged over sampled tokens, with $\epsilon = 10^{-8}$. We report $\mathrm{corr}(\{C_i\}, \{\Delta_{i,\mathrm{swap}}\})$ and $\mathrm{corr}(\{M_i\}, \{\Delta_{i,\text{branch-merge}}\})$, using Pearson and Spearman correlations. These are diagnostic associations, not general predictor validation.

Table 11 gives the sampling conditions for this diagnostic comparison. Table 12 separates correlations across all candidate pairs from those excluding boundary pairs, and Figure 13 locates the corresponding loss changes across depth. Reading them together tests whether a correlation describes interior behavior or is dominated by boundary layers.

Table 11: Correlation pilot setup. These diagnostic-correlation runs use WikiText-2 validation text, sequence length 256, 64 sequences, block size 1, stride 1, and seed 7.

Checkpoint	Architecture	Seq.	Length	Pairs	Edits
gpt2	GPT-2	64	256	11	swap, branch–merge
EleutherAI/pythia-160m	GPT-NeoX	64	256	11	swap, branch–merge

Table 12: Frozen graph-edit diagnostic correlations. Higher ΔNLL is worse. Interior rows exclude the first and final adjacent pairs. These numbers are limited diagnostic associations, not validation of a general predictor.

Checkpoint	Pairs included	n	Pearson swap	Spearman swap	Pearson branch–merge	Spearman branch–merge
GPT-2	all	11	0.986	0.327	0.936	0.545
GPT-2	interior	9	0.212	0.433	0.824	0.767
Pythia-160m	all	11	0.754	0.091	0.645	0.318
Pythia-160m	interior	9	0.060	0.083	0.425	0.500

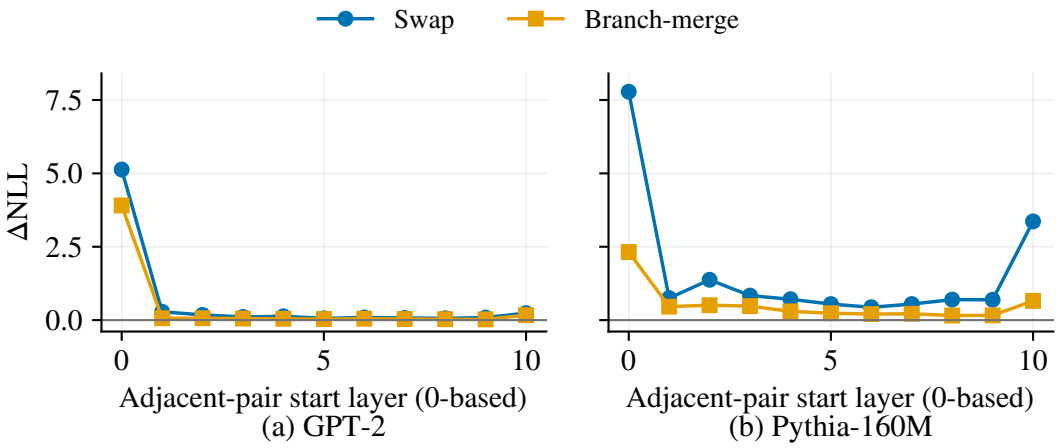


Figure 13: Depth profile of frozen graph-edit tolerance. For each adjacent layer pair, the figure shows ΔNLL from swap and branch–merge edits; higher values are worse. Both panels share the same vertical scale and zero-based layer indexing.

The depth profiles and correlations reveal why a single local association is insufficient. Branch–merge yields lower ΔNLL than swapping at many displayed interior layers, but the correlations vary across models and boundary layers dominate several trends. Because depth remains a possible confound, these observations motivate broader model-level evaluation rather than establishing a standalone predictor of edit quality.

G Static Graph-Edit Follow-Up Details

This appendix supplies supporting comparisons for Sections 3 and 4: joint-skip quality, forward execution time, and quality recovered by interface repair for skip and branch–merge edits. The frozen checkpoints are Pythia-410M, Pythia-1B, Pythia-2.8B, and Pythia-6.9B; repair uses the two smaller checkpoints. The tables summarize the evaluated model, data, and workload configurations, rather than comparisons between separately trained architectures. Their aggregation populations differ and are specified below.

G.1 Local-Global Selection

The first comparison tests whether locally selected layers produce different damage when skipped together. Table 13 reports the complete edited model's held-out ΔNLL by edit count k and selection rule. Each cell is an equal-weight mean of fourteen model/data/partition configurations, including multiple corpora and additional partitions of the smallest checkpoint. The low-update group has less damage than the high-update controls, while middle-depth and random rules sit between these extremes or vary by k. This comparison identifies lower-damage candidates within the evaluated set, without itself specifying an acceptable deployment tolerance.

Table 13: Mean held-out ΔNLL after jointly skipping locally selected layers. Lower is better.

k	Low update	Middle	Random	High update
1	0.129	0.150	0.276	0.875
2	0.283	0.315	0.582	1.799
4	0.661	1.091	1.039	3.958

G.2 Cost Ratios

Execution time is a complementary outcome because loss differences alone do not establish runtime savings. Table 14 reports mean edited/original forward-time ratios by k and selection rule, giving equal weight to seventeen model/workload configurations. The ratios decrease as more layers are skipped for every listed rule, even though the quality comparison distinguishes their damage. Timing and loss therefore answer different parts of the model-choice question. These are stack-level timings; Section 4 provides separate serving-oriented measurements.

Table 14: Mean edit/full forward-time ratio by selection rule. Lower is faster under the measured regime.

k	Low update	Middle	Random	High update
1	0.957	0.906	0.904	0.921
2	0.887	0.851	0.853	0.851
4	0.788	0.774	0.764	0.789

G.3 Repair Summaries

A separate comparison of skip and branch–merge candidates tests whether training a reconnection map recovers damage while the checkpoint stays frozen. Table 15 reports mean recovery ratios as defined in Appendix B. Each skip cell averages eighteen low-rank repair configurations; each branch–merge cell averages eighteen configurations spanning scalar, diagonal, and low-rank maps. Positive values indicate reduced degradation relative to the no-repair edit, while negative values indicate increased damage. Recovery is partial in the positive groups, and the high-update branch–merge group worsens on average. Repair consequently changes the trade-off, rather than guaranteeing that an edited model becomes acceptable.

Table 15: Mean interface-repair recovery ratio. Skip rows use low-rank maps; branch–merge rows pool scalar, diagonal, and low-rank maps. Untrained controls are excluded from these means.

Edit	Low update	Random	High update
Branch-merge	0.136	0.203	-0.219
Skip	0.304	0.195	0.013

G.4 Scope Boundary

Together, the quality, timing, and repair comparisons describe different consequences of static edits to existing checkpoints. A separate training study can compare models trained under alternative graphs with post-training edits under matched quality, cost, repair, and data budgets.

H NVIDIA A800 CROSS-FAMILY STATIC GRAPH-EDIT DETAILS

This appendix tests whether candidate-selection and quality–cost patterns extend across model families on NVIDIA A800 hardware. The suite covers Pythia-1B, Pythia-2.8B, Pythia-6.9B, Qwen2.5-0.5B, Qwen2.5-1.5B, Qwen2.5-7B, TinyLlama-1.1B, and Mistral-7B. We report joint-skip damage, quality-constrained timing, downstream behavior, and interface repair as supporting evidence for Sections 3 and 4. Table 16 inventories the available records; the candidate-level artifact retains individual checkpoint, dataset, and layer decisions.

Table 16: Record counts in the A800 evaluation summary tables and candidate-level artifact. These are data rows, not independent training runs.

Audit	Local-global	Cost	Downstream	Serving	Repair	Frontier	Bootstrap	Candidates
3784	324	864	25	468	324	72	3	5465

Selection and joint damage. Table 17 compares the held-out loss increase after the selected layers are skipped together. High-update selection is substantially more damaging than the other rules, while the lowest-damage competitive rule changes with the edit count. Table 18 quantifies the high-minus-low-update gap using matched evaluation configurations; its intervals remain positive at every listed edit count.

Table 17: Cross-family held-out local-global ΔNLL by selector. Each cell averages 18 model/data configurations with equal weight; lower is better.

Selector	$k = 1$	$k = 2$	$k = 4$
Low calibration ΔNLL	0.050	0.130	0.418
Suffix hidden cosine	0.065	0.136	0.343
Middle depth	0.087	0.167	0.483
Low update	0.096	0.214	0.587
Random	0.176	0.410	0.944
High update	2.600	3.030	4.302

Table 18: NVIDIA A800 paired-bootstrap gap in local-global ΔNLL, computed as high update minus low update.

k	Pairs	Mean gap	95% low	95% high
1	18	2.504	1.800	3.151
2	18	2.816	2.057	3.605
4	18	3.716	3.373	4.120

Quality constraints and task behavior. The separation in loss does not yet identify an acceptable quality–cost trade-off. Table 19 asks whether each evaluation configuration has an edit within a stated loss tolerance and reports the mean accepted timing ratio. Most configurations admit an edit even at the strictest listed tolerance. Task accuracy remains a separate requirement, tested in Table 20: low-update skipping has less task damage than the high-update control, while middle-depth selection has the highest edited-model accuracy in this aggregate.

Table 19: A800 quality-constrained forward-time comparison. Quality and timing records are matched by model, selector, and edit count. Here the accepted timing ratio measures a full forward pass, not cached per-token decoding.

ϵ	Within tolerance	Outside tolerance	Mean accepted r_T
0.10	17	1	0.947
0.25	18	0	0.940
0.50	18	0	0.932
1.00	18	0	0.933

Table 20: A800 downstream-task subset after skipping two layers, averaged across Mistral-7B, Pythia-6.9B, Qwen2.5-1.5B, Qwen2.5-7B, and TinyLlama-1.1B. Tasks are HellaSwag, ARC-Easy, ARC-Challenge, and three MMLU subjects.

Condition	Accuracy	Mean loss Δ	p95 loss Δ
Original	0.448	0.000	0.000
Low update	0.422	0.125	0.954
Middle-depth	0.440	0.097	0.801
Random	0.417	0.298	1.318
High update	0.309	2.572	5.289

The downstream evaluation in Table 20 also requested PIQA, but its dataset loader rejected the legacy dataset script. PIQA was therefore excluded rather than counted as a successful evaluation. The remaining downstream suite contains 318–320 valid multiple-choice examples per checkpoint.

Timing proxies and repair. Two further comparisons examine execution cost and recoverable quality separately. The no-cache timings in Table 21 decrease as more layers are skipped. The decode proxy used for these measurements recomputes the growing sequence without a production key–value cache, so these ratios are distinct from the cache-enabled serving results in Section 4.

For recoverable quality, Table 22 measures the effect of training a reconnection map. Low-update edits recover some loss on average, whereas random and high-update groups worsen on average under the reported aggregation. The repair comparison therefore adds a possible quality adjustment to the cost profile, not a guarantee that the original damage can be removed.

Table 21: A800 serving-facing proxy ratios averaged over selection rules.

k	Prefill ratio	Decode-proxy ratio
1	0.954	0.962
2	0.915	0.922
4	0.837	0.844

Table 22: Mean A800 interface-repair recovery ratio, defined in Appendix B. Positive values indicate recovered loss; negative values indicate increased damage. The All repair rows aggregation includes untrained zero-recovery controls, while Low-rank only retains fitted low-rank maps.

Aggregation	Low update	Random	High update
All repair rows	0.099	-0.084	-0.148
Low-rank only	0.141	-0.269	-0.411

These aggregates summarize different comparison populations. To trace a specific model choice, the supplementary artifact bundle retains the candidate-level selections, matched quality and cost records, metadata, and failure records. Its candidate-level table identifies the selected layers and support/reject labels; the summary averages do not constitute a separate selection procedure.

I Exact ShortGPT Baseline Details

This appendix tests whether exact ShortGPT Block Influence selects low-damage skips and how its ordering relates to complete-model quality. The comparison uses frozen Qwen2.5-1.5B, Qwen2.5-7B, and Mistral-7B-v0.1 checkpoints, with the same calibration/test split and skipped-layer budget for every selector within a checkpoint. Pythia-410M was used only for an implementation check and is excluded from the evidence tables. We first report joint-skip loss and individual-layer ranking correlations, then examine downstream accuracy and tail damage. These results concern deletion, not recurrence, routing, branch–merge, or pretraining under alternative graphs.

I.1 Held-Out Local-Global Quality

The first comparison asks which selector preserves language-model quality when its chosen layers are skipped together (Table 23). The selector ordering changes with both checkpoint and edit budget.

Exact BI is competitive in several Qwen regimes but does not dominate the direct calibration-loss, suffix-cosine, or middle-depth controls. On Mistral, it has higher joint loss than those three controls at every listed budget.

Table 23: Held-out local-global ΔNLL for the exact ShortGPT follow-up. Lower is better. Suffix cosine is the earlier final-hidden-state control and is not ShortGPT BI.

Model	k	Low update	Exact BI	Suffix cos.	Cal. loss	Middle	Random	High update
Qwen2.5-1.5B	1	0.070	0.070	0.046	0.046	0.089	0.128	3.173
	2	0.111	0.111	0.116	0.116	0.117	0.180	3.000
	4	0.246	0.246	0.246	0.246	0.201	0.590	3.441
Qwen2.5-7B	1	0.044	0.044	0.044	0.054	0.035	0.110	0.333
	2	0.120	0.087	0.087	0.085	0.085	0.286	0.653
	4	0.246	0.185	0.185	0.204	0.258	0.448	3.752
Mistral-7B	1	0.169	0.077	0.052	0.042	0.036	0.098	3.001
	2	0.517	0.209	0.116	0.136	0.108	0.177	3.319
	4	1.943	0.546	0.327	0.291	0.416	0.336	4.069

The joint-edit results leave open how well a score tracks the damage of individual candidates. Table 24 addresses that narrower question using held-out interior-layer Spearman correlations with per-layer ΔNLL. Graph diagnostic denotes the relative-update score; both the scores and loss changes are evaluated on the held-out split, excluding the first and last layer. Relative-update correlations are higher than Exact BI correlations on both Qwen checkpoints, while both scores correlate weakly with damage on Mistral. Comparing these correlations with the joint-edit table shows why within-split ranking agreement does not guarantee the best calibration-selected combination at every k.

Table 24: Held-out interior-layer Spearman correlation with per-layer ΔNLL.

Model	Graph diagnostic	Exact BI
Qwen2.5-1.5B	0.867	0.420
Qwen2.5-7B	0.880	0.662
Mistral-7B	0.097	0.059

I.2 Downstream Performance and Tail Risk

The downstream comparison asks whether mean language-model loss adequately describes the behavior of selected skips. We evaluate $k = 2$ selections on a 200-example multiple-choice subset per checkpoint, drawn from HellaSwag, ARC-Easy, ARC-Challenge, and MMLU. PIQA was requested but excluded after its dataset loader failed. Table 25 reports accuracy and the 95th percentile of per-example answer-loss change. With 200 examples, accuracy differences remain uncertain, while the tail statistic can expose individual regressions hidden by the mean.

Table 25: Downstream performance for $k = 2$. Higher accuracy is better; lower p95 Δloss is better.

Model	Condition	Accuracy	p95 Δloss
Qwen2.5-1.5B	Original	0.480	0.000
	Low update	0.440	0.976
	Exact BI	0.440	0.976
	Cal. loss	0.445	1.125
	Middle	0.460	0.705
	Random	0.400	1.006
	High update	0.270	6.784
Qwen2.5-7B	Original	0.490	0.000
	Low update	0.510	0.359
	Exact BI	0.495	0.360
	Cal. loss	0.515	0.470
	Middle	0.515	0.470
	Random	0.490	0.939
	High update	0.450	1.970
Mistral-7B	Original	0.515	0.000
	Low update	0.460	1.284
	Exact BI	0.485	0.813
	Cal. loss	0.500	0.438
	Middle	0.500	0.596
	Random	0.490	0.439
	High update	0.235	6.816

The task results make this distinction concrete: Exact BI has lower tail damage than the high-update control on every reported checkpoint, but calibration-loss and middle-depth selection have higher accuracy point estimates in these comparisons. Avoiding the harmful control therefore does not identify the best task outcome. Taken together, the comparisons distinguish tracking individual damage, choosing jointly skipped layers, and preserving task behavior. A model choice must use the outcome relevant to the application, together with the resource measurements discussed in Section 4.

J Submission Validation Details

The comparisons in this appendix test different reasons why an edit's apparent benefit may not settle the model choice in Sections 3 and 4. Partition changes test selection stability, a published loop illustrates a quality-for-computation trade, and serving measurements determine the actual resource consequence. Complementary repair, task, and corpus-transfer comparisons examine what happens beyond the initial loss measurement. Graph selection uses calibration-side evidence and quality values use held-out data. Checkpoint weights remain frozen; interface repair trains only an added reconnection map.

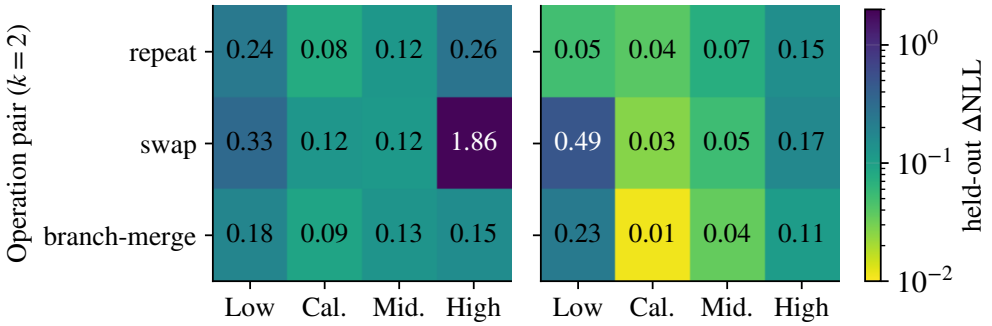
Selection across data partitions. Table 26 tests how skip selection behaves across partitions of the evaluation corpus. It pools three WikiText calibration/test partitions each for Qwen2.5-1.5B, Mistral-7B, and Gemma2-2B. High-update selection has the largest mean damage at each listed edit count, while the ranking of the remaining rules by selected-model loss depends on the edit count. These are repeated evaluations of fixed checkpoints, not independent training runs.

Table 26: Partition-seed mean held-out ΔNLL across 9 evaluated checkpoint/partition units. Here, k is the number of jointly edited blocks. Lower is better.

Selector	$k = 1$	$k = 2$	$k = 4$
Cal. loss	-0.1012	-0.1136	0.0447
Middle	0.1000	0.0084	0.1276
Exact BI	0.0685	0.3046	0.6777
Low update	0.0938	0.3892	1.0207
High update	2.6515	3.0418	3.2013

Non-skip selector differences. Figure 14 compares held-out damage after jointly applying two non-overlapping repetition, swap, or branch–merge edits of the same type. Calibration loss, middle depth, and low/high update select different candidate locations before held-out evaluation; their

exact calculations appear in Appendix B. The ranking of these selection rules changes across checkpoints and operation types, so a rule that identifies a less damaging edit in one setting is not a universal choice for another.



(a) Qwen2.5-1.5B (b) Mistral-7B

Figure 14: Held-out negative-log-likelihood increase after jointly applying two non-overlapping edits of the same operation type: repetition, swapping (reordering), or branch–merge (parallel execution). Lower is better and color uses a logarithmic scale. Calibration loss, middle depth, and low/high update use the selection rules described in Appendix B. Selector order changes across checkpoints and operation types.

A published loop's task trade-off. Table 27 reports the quality–computation comparison summarized in the main text, and Table 28 decomposes its task aggregate. The loop benchmark draws on the AI2 Reasoning Challenge (ARC) (Clark et al., 2018), HellaSwag (Zellers et al., 2019), Physical Interaction Question Answering (PIQA) (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2020), LAMBADA (Paperno et al., 2016), and SciQ (Welbl et al., 2017). It also includes OpenBookQA (Mihaylov et al., 2018), CommonsenseQA (Talmor et al., 2019), TruthfulQA (Lin et al., 2022), and Massive Multitask Language Understanding (MMLU) (Hendrycks et al., 2021). Task changes have both signs: the small aggregate improvement does not mean every task improves.

The loop reproduction fixes the authors' released repository at revision `2b20a873`. It applies the block-mode anchored damped Euler recipe to zero-indexed Qwen3-4B-Base layers 15–18, with six block passes and anchor coefficient 0.5. The cache policy retains the first pass's key–value states; single-token cached decoding bypasses the extra loop passes. The 36-to-56 call comparison in Table 27 therefore describes the full looped forward path, not every decoding step. This executable repository recipe differs from the fixed two- or three-pass descriptions in the paper's first arXiv version (Chen et al., 2026); the reported results reproduce the specified code revision.

Table 27: Reproduction of the released Training-Free Looped Transformer recipe on Qwen3-4B-Base. Accuracy rows are macro aggregates; n is the number of tasks. Table 28 gives task-level results.

Measure	n	Original	Loop	Difference
Commonsense accuracy	9	0.6713	0.6769	+0.0056
Math/physics/logic accuracy	6	0.5806	0.5883	+0.0077
All-task macro accuracy	16	0.6385	0.6445	+0.0061
Layer calls	–	36	56	+20
Wall-clock (s)	–	829.3	1206.8	+377.4

Table 28: Task-level official frozen-loop results on Qwen3-4B-Base. Each row uses the harness-standard accuracy variant; higher is better. The main-text commonsense aggregate contains the nine non-MMLU tasks other than LAMBADA; the six-subject MMLU aggregate includes formal logic. LAMBADA is included separately in the 16-task mean.

Task	Original	Loop	Difference
ARC-Easy	0.7622	0.7904	+0.0282
ARC-Challenge	0.5145	0.5469	+0.0324
HellaSwag	0.7359	0.7384	+0.0025
PIQA	0.7769	0.7769	+0.0000
WinoGrande	0.7056	0.7064	+0.0008
Lambada	0.6903	0.6907	+0.0004
SciQ	0.9430	0.9450	+0.0020
OpenBookQA	0.4100	0.4020	-0.0080
CommonsenseQA	0.8247	0.8198	-0.0049
TruthfulQA MC1	0.3684	0.3660	-0.0024
MMLU Elem. Math	0.7143	0.7169	+0.0026
MMLU HS Math	0.5111	0.5481	+0.0370
MMLU HS Physics	0.6689	0.6556	-0.0132
MMLU College Math	0.5300	0.5300	+0.0000
MMLU Formal Logic	0.5794	0.5794	+0.0000
MMLU Abstract Algebra	0.4800	0.5000	+0.0200

Cache-enabled serving costs. The serving comparisons separate skipped computation from physical memory savings. In Table 29, skipping reduces prefill time, per-token decoding time, and cache allocation, while peak allocation changes little because inactive weights remain resident. Table 30 reports a different implementation: removing unused weights under a fixed memory budget increases the available cache pool, alongside lower latency and higher throughput. These are measured runtime-specific outcomes, not consequences inferred from graph depth alone.

Table 29: Cache-enabled Hugging Face serving summary. Ratios are relative to matched unedited runs. Here, k is the number of removed blocks and n is the number of measured serving regimes.

Model	k	n	Prefill	Decode/token	KV	Peak
Qwen2.5-1.5B	2	18	0.9415	0.9421	0.9286	0.9988
Qwen2.5-1.5B	4	18	0.8813	0.8744	0.8571	0.9976
Qwen2.5-7B	2	6	0.9378	0.9430	0.9286	0.9997
Mistral-7B	2	20	0.9407	0.9425	0.9375	0.9991
Gemma2-2B	2	24	0.9362	0.9387	0.9231	0.9977
Gemma2-2B	4	24	0.8753	0.8637	0.8462	0.9950
Gemma2-9B	2	24	0.9573	0.9567	0.9524	0.9981
Gemma2-9B	4	24	0.9130	0.9129	0.9048	0.9963

Table 30: vLLM serving transfer on Qwen2.5-1.5B. Throughput, time, and peak columns are median edited/original ratios across nine context–concurrency regimes, each measured three times. TTFT is time to first token, TPOT is mean time per output token, and KV denotes key–value cache. KV blocks and weight memory are engine-profile values under a fixed 65% graphics processing unit (GPU) memory budget. GiB denotes gibibytes.

Condition	Layers	Throughput	TTFT	TPOT	Peak	Weight GiB	KV blocks
Original	28	1.000	1.000	1.000	1.000	2.91	63,982
Skip $k = 2$	26	1.061	0.935	0.948	0.999	2.74	69,343
Skip $k = 4$	24	1.124	0.880	0.896	0.999	2.56	75,598

Repair and task behavior. A separate quality question is whether adapting a reconnection can reduce the damage caused by repetition or swapping. Table 31 evaluates these repairs, rather than repairing the skips in the serving comparison. The best measured repairs reduce mean damage in the listed cases but leave residual damage. Safe and Unsafe in this table denote low and high operation-specific local-discrepancy selection, not guarantees about the resulting model.

Table 31: Repeat/swap reconnection repair, measured by held-out NLL increase. "Best" is the minimum mean damage over the prespecified repair grid. The 95th percentile (p95) comes from the same configuration, and recovery is the median across candidate units; n counts those units. This held-out minimum summarizes repairability and is not used to select the original edit.

Model	Edit	Rule	n	No repair	Best	p95	Recovery
Qwen2.5-1.5B	Repeat	Safe	2	0.1182	0.0824	0.1425	0.310
Qwen2.5-1.5B	Repeat	Unsafe	2	0.1866	0.1371	0.2546	0.257
Qwen2.5-1.5B	Swap	Safe	2	0.1599	0.1365	0.2434	0.146
Qwen2.5-1.5B	Swap	Unsafe	2	0.8552	0.7135	1.6871	0.204
Mistral-7B	Repeat	Safe	1	0.0305	0.0303	0.0753	0.008
Mistral-7B	Repeat	Unsafe	1	0.1104	0.0872	0.1284	0.210
Mistral-7B	Swap	Safe	1	0.4721	0.4591	0.6657	0.027
Mistral-7B	Swap	Unsafe	1	0.0761	0.0610	0.1783	0.198

Loss recovery and task correctness are different outcomes. In a separate skipping experiment, Table 32 measures answer correctness and formatting on GSM8K (Cobbe et al., 2021) after removing two layers. The edited conditions have higher formatting rates but lower strict-answer accuracy than the original model. This contrast illustrates why an improvement in one behavioral measure cannot stand in for the task outcome the application needs.

Table 32: Qwen2.5-7B GSM8K performance after skipping two layers on a fixed 200-example sample. EM denotes exact match and CI denotes confidence interval. The final column is the paired bootstrap 95% CI for strict-EM change versus the original model.

Condition	Strict EM	Format rate	ΔEM 95% CI
Original	0.310	0.370	–
Low update	0.235	0.430	[-0.135, -0.015]
Exact BI	0.260	0.600	[-0.110, +0.015]
High update	0.270	0.490	[-0.100, +0.020]

The Gemma comparisons further separate text loss from task correctness. Table 33 reports held-out language-model damage, while Table 34 tests multiple-choice behavior. A favorable mean text-loss change does not ensure a favorable accuracy change, as illustrated by low-update selection on the larger checkpoint.

Table 33: Gemma cross-family skip selection at $k = 2$, measured by held-out NLL increase. Gemma2-2B entries average three calibration/test partition seeds; Gemma2-9B uses one partition. Lower is better.

Model	Low upd.	Exact BI	Cal. loss	Middle	Random	High upd.
Gemma2-2B	0.580	0.580	-0.579	-0.195	1.367	2.601
Gemma2-9B	-0.239	-0.239	-1.091	-0.393	-0.442	1.007

Table 34: Gemma downstream-task check at $k = 2$ on two hundred multiple-choice examples per checkpoint from HellaSwag, ARC-Easy, ARC-Challenge, and MMLU. Accuracy is the fraction of correct examples. Loss columns summarize per-example answer-loss changes relative to the original checkpoint; lower is better.

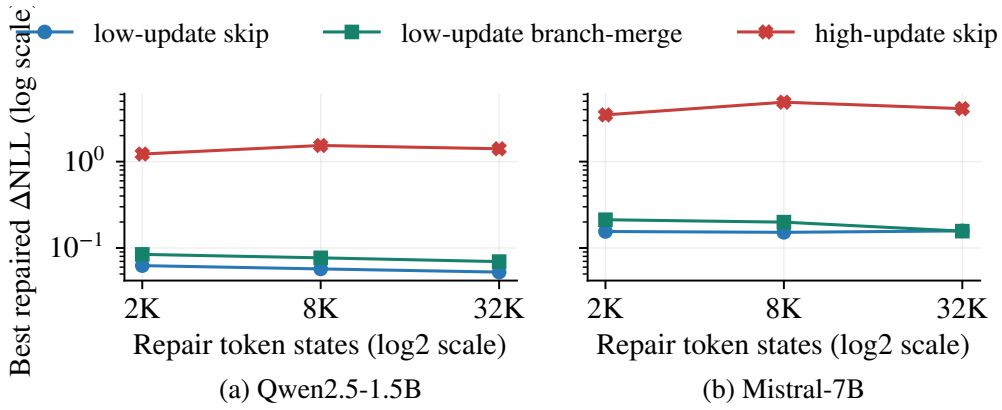
Model	Condition	Accuracy	Mean Δloss	p95 Δloss
Gemma2-2B	Original	0.310	0.000	0.000
Gemma2-2B	Low update	0.330	0.509	2.385
Gemma2-2B	Middle	0.320	-0.230	1.526
Gemma2-2B	Random	0.315	0.591	2.949
Gemma2-2B	High update	0.310	2.360	5.813
Gemma2-9B	Original	0.330	0.000	0.000
Gemma2-9B	Low update	0.280	-0.259	1.201
Gemma2-9B	Middle	0.335	-0.193	1.539
Gemma2-9B	Random	0.315	-0.314	1.300
Gemma2-9B	High update	0.320	0.733	2.672

Transfer and repair-budget boundaries. The final comparisons vary either the evaluation corpus or the interface-adaptation budget. Table 35 tests whether edits selected on WikiText retain their behavior on other corpora. The high-update rule is strongly harmful for skipping on both checkpoints, but its relative performance differs for other operations. Thus, transferring a selection rule requires checking the operation as well as the model and corpus.

Table 35: Cross-domain held-out ΔNLL at $k = 2$, averaged over TinyStories and Pile-10k. Candidate identities are selected on WikiText and frozen before transfer. Lower is better.

Model	Edit	Cal. loss	Low update	High update
Qwen2.5-1.5B	Skip	0.129	0.113	3.274
	Repeat	0.072	0.168	0.153
	Swap	0.101	0.212	1.282
	Branch-merge	0.085	0.087	0.121
Mistral-7B	Skip	0.063	0.296	2.571
	Repeat	0.029	0.059	0.123
	Swap	0.011	0.284	0.096
	Branch-merge	0.005	0.122	0.075

The budget comparison in Figure 15 varies interface-training exposure for skip and branch–merge candidates. The best observed recovery depends on both the edit and token budget. These curves complement the repeat/swap repair summary in Table 31 by examining another operation set and making budget dependence visible: extra adaptation does not uniformly recover lost model quality.



(a) Qwen2.5-1.5B (b) Mistral-7B

Figure 15: Best held-out ΔNLL after interface-only repair at each hidden-token-state budget. Lower is better. The shared vertical axis uses a logarithmic scale, and the horizontal axis uses a base-2 logarithmic scale for the shared token budgets. Some candidate interfaces improve, while high-update skips remain poor or worsen; additional repair tokens therefore do not guarantee monotonic improvement.

K Submission Notes

This appendix summarizes language-model assistance in the research and manuscript preparation, then locates the details needed to interpret and reproduce the reported comparisons.

Large language model usage. Large language model (LLM) tools supported implementation and analysis as well as ideation, outlining, prose revision, terminology checks, and adversarial critique, as summarized in the main-text tool-use statement. They were not used as authoritative sources of technical facts. The authors checked claims, citations, and diagnostic results against cited work, manuscript sources, and experimental outputs and remain responsible for the manuscript, code, and analyses.

Reproducibility statement. The supporting materials connect the graph-design principles in Section 3 and the evaluation framework in Section 4 to their detailed evidence. Appendix C records the

resource-composition and mixed-operation experiments after common LoRA post-training. Frozen studies cover Pythia, Qwen2.5, TinyLlama, Mistral, and Gemma2, with local sensitivity in Appendix F, aggregate comparisons in Appendices G and H, and the exact ShortGPT Block Influence comparison in Appendix I. The external loop reproduction uses Qwen3; Appendix J specifies its recipe and the supplementary task and serving tests. Appendices D and E document the separate evaluation-requirement comparison and literature reporting audit.

These experimental records distinguish behavioral outcomes from resource measurements. Quality evaluation reports next-token loss and task-specific accuracy separately. Cost evaluation records the execution stack, prefill/decode regime, and key–value caching configuration. These resource measurements characterize the tested single-device settings; broader production deployments require evaluation with their own serving stack.