PROGRESSIVE BRANCH-MERGE: TRAINABLE SEQUENCE PROCESSING ACROSS FROZEN TRANSFORMER DEPTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Equally sized additions to a frozen language model can learn different computations: some widen updates to existing states, while others transform token context throughout decoder depth. We introduce Progressive Branch-Merge (PBM), which pairs each frozen decoder block with a trainable branch that attends to earlier tokens and writes the fused backbone–branch output into a shared residual state that both paths read at the next stage. After one pass through the training text in mathematics, physics, chemistry, and biology, PBM has the lowest mean next-token loss among the tested adaptations of Pythia-1B. On separately fixed fresh evaluation arrays after two passes, it also lowers next-token negative log likelihood relative to a nearly parameter-matched branch-free updater in mathematics, physics, and biology, with four-domain means of 2.0781 and 2.0847; the direct updater has lower loss in chemistry. PBM's relative gain rises with sequence difficulty estimated from thirteen other adaptations, and bypassing its trained branch blocks raises loss more on harder sequences. These results establish stagewise trainable causal processing as a useful allocation of adaptation parameters and show where that allocation yields the larger loss reduction.

1 INTRODUCTION

Domain continued pretraining adapts a language model to unlabeled target text while preserving its next-token objective (Gururangan et al., 2020). Freezing the pretrained backbone and training added parameters makes it possible to keep a separate, compact set of weights for each domain (Houlsby et al., 2019; Hu et al., 2022). An approximately equal number of trainable parameters can support different computations: a wide update to a frozen block's output, or causal token transformations distributed across decoder depth. The latter gives the adaptation its own sequence processing at multiple stages rather than relying entirely on direct state updates. We ask when that allocation improves the model's fit to new-domain text.

Parallel adapters compute local updates beside frozen sublayers, while side networks can take intermediate backbone activations into a separately advancing stream (He et al., 2022; Zhang et al., 2020; Sung et al., 2022). We introduce Progressive Branch-Merge (PBM) to make the next stage a shared point of departure for pretrained and newly learned computation. At every decoder stage, a frozen block and a trainable causal branch read the same incoming residual state; a shared gated module fuses their outputs into a new residual state that both paths read next. This repeated return to one state distinguishes PBM from carrying an independent side-stream state forward and gives the causal branch fresh backbone information throughout depth.

The key test is whether this distributed causal computation earns its parameter budget. We compare PBM with other adaptation families and with a branch-free updater that spends nearly the same number of parameters on wider direct state transformations. Matched structural controls further distinguish the value of causal token mixing from the choice of whether the branch reads a frozen block's input or output. We then examine which evaluation sequences account for the difference, using difficulty estimated from other adaptations, and measure how loss changes when PBM's trained branch blocks are bypassed.

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.

PBM has the lowest four-domain mean among the tested one-pass methods and, on new evaluation arrays, lower loss than the branch-free updater in three domains and on their equal-weight mean. The relative gain rises across harder sequences, while the branch-skipping comparison shows that the trained branch contributes to PBM's predictions. A matched branch that processes each token independently is weaker, locating part of the value in causal token mixing; letting the causal branch read the current frozen block's output can improve further, showing that read position remains a separate design choice. Our contribution is a concrete frozen-backbone adaptation graph and a controlled account of when distributing trainable causal computation across depth is more useful than widening direct state updates in the studied domains.

2 Related Work

Adaptation methods allocate learned parameters to different computations. Domain-adaptive continued pretraining learns from unlabeled target-domain text while retaining the language-modeling objective (Gururangan et al., 2020). With a largely frozen backbone, adapters transform intermediate states (Houlsby et al., 2019; He et al., 2022), low-rank methods change selected weight updates (Hu et al., 2022; Zhang et al., 2023; Liu et al., 2024), and scaling or learned-context methods change activations or inputs (Liu et al., 2022; Lester et al., 2021; Li & Liang, 2021). These methods show that a count of trainable parameters does not identify the computation they add. Our question compares two uses of a similar budget: wider direct updates to backbone states and causal sequence processing distributed across decoder stages.

Parallel adapters and side networks differ in when they read backbone states. Parallel adapters compute an update from the input to a frozen sublayer and combine the two outputs (He et al., 2022; Chen et al., 2022). Side-Tuning trains a separate network whose output joins that of a fixed base model (Zhang et al., 2020). Ladder Side-Tuning feeds intermediate backbone activations into a side network that continues from its own preceding state (Sung et al., 2022). PBM instead writes each backbone–branch fusion back into the residual state read by both the next frozen block and the next causal branch. The question is whether this repeated shared-state update makes good use of a fixed adaptation budget, compared with allocating the same parameters to a wider direct updater.

Added depth and parallel tracks expose another design axis. LLaMA Pro inserts complete trainable Transformer blocks while retaining pretrained blocks (Wu et al., 2024). For multi-device inference, Kraken studies parallel computation within a layer (Prabhakar et al., 2024), and Parallel Track Transformers let separate tracks advance between merge points (Wang et al., 2026a). These works establish ways to add trainable depth or organize parallel tracks for different purposes, including multi-device inference. PBM studies a frozen-backbone domain-adaptation setting: one new causal block runs beside each pretrained block, and both outputs return to a shared state before the next stage.

Adaptive depth motivates a difficulty-stratified test. Adaptive-depth Transformers can assign different amounts of repeated computation to different tokens (Simoulin & Crabbé, 2021). Mixture-of-Depths distributes a fixed compute budget across token positions (Raposo et al., 2024), and AdaPonderLM reports more computation on high-loss tokens in recurrent Pythia models (Song et al., 2026). PBM keeps branch depth fixed and uses difficulty as an evaluation condition rather than a routing rule. We test whether the fixed causal branch contributes more on sequences that other adaptation methods find difficult, and whether its value exceeds a wider direct state update under the studied budget.

3 Preliminaries and Problem Setting

The allocation question concerns two related objects: the states produced by a frozen decoder and the computation graph through which added trainable operations read and update those states. We define the decoder's inter-block states first, then state what remains free to change when the trainable-parameter budget is held approximately fixed.

3.1 FROZEN DECODER STATES

Let $x = (x_1, \ldots, x_n)$ contain n input tokens. A causal decoder embeds x, applies L Transformer blocks of width d, and uses final normalization and an output head to predict the next token at each position. Write $\vec{H}_0 \in \mathbb{R}^{n \times d}$ for the embedded input and $\vec{H}_\ell$ for the outgoing residual representation after block ℓ, with $\ell \in \{1, \ldots, L\}$. For $\ell < L$, $\vec{H}_\ell$ enters the next block; $\vec{H}_L$ enters final normalization and the output head. The states $\vec{H}_0, \ldots, \vec{H}_L$ form the residual stream at which an adaptation can read or write. Given $\vec{H}_{\ell-1}$, frozen block ℓ produces

$$\vec{U}_\ell = \texttt{BackboneBlock}_\ell \left(\vec{H}_{\ell-1}; \vec{\theta}_\ell \right) \in \mathbb{R}^{n \times d}, \tag{1}$$

where $\vec{\theta}_\ell$ denotes that block's pretrained parameters. The direct block output $\vec{U}_\ell$ equals the outgoing state $\vec{H}_\ell$ without adaptation. An added operation can instead change $\vec{H}_\ell$ before the next block reads it. We call the computation from $\vec{H}_{\ell-1}$ to $\vec{H}_\ell$ stage ℓ; it contains frozen block ℓ and may contain trainable operations. The pretrained embedding, blocks, final normalization, and output head together have parameters $\vec{\theta}$, all fixed in the setting studied here.

3.2 THE ADAPTATION-GRAPH QUESTION

An adaptation computation graph specifies where a trainable operation reads the decoder state, how many trainable transformations lie along its path, and where its output rejoins the residual stream. A path can repeatedly use states updated by intervening frozen blocks, or continue from its own preceding trainable state. Likewise, a graph can place most parameters in a wide update at one state boundary or distribute narrower transformations across depth. Similar trainable-parameter counts therefore allow different state dependencies and different executed computation.

We hold the frozen backbone, target-domain text, next-token objective, data exposure, and approximate number of trainable parameters fixed. The open question is when distributing causal sequence transformations across stages fits held-out domain text better than using the same budget for wider direct transformations of backbone states. Parameter matching alone does not match floating-point operations, throughput, or memory; the experiments measure those resource differences separately. Section 4 defines the two adaptation graphs, and Section 5 specifies their evaluation.

4 PROGRESSIVE BRANCH-MERGE

Progressive Branch-Merge (PBM) adds causal sequence processing at every stage of a frozen decoder without changing its pretrained parameters. Each frozen block and its trainable branch block read the same incoming state; a shared gated module writes their fused output back before the next stage. The next pair therefore receives a state shaped by both paths rather than a separately carried branch state. Figure 1 shows this repeated computation, followed by the branch, fusion, and initialization details below. We then define a branch-free updater that uses a similar trainable-parameter budget for direct state transformation.

4.1 STAGE-SPECIFIC BRANCH BLOCKS PROVIDE TRAINABLE DEPTH

At stage ℓ, PBM applies a width-b causal Transformer block to the same $\vec{H}_{\ell-1}$ read by the frozen width-d block in (1). The branch projects $\vec{H}_{\ell-1}$ from d to b, performs a trainable sequence transformation, and projects the transformed sequence back to d:

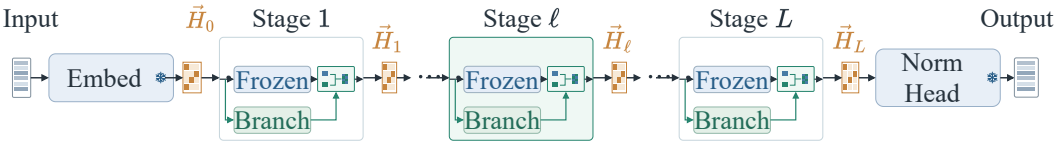
$$\vec{Z}_\ell = \texttt{BranchInput} \left(\vec{H}_{\ell-1}; \vec{\phi}_{\text{in}} \right), \tag{2}$$

$$\vec{Q}_\ell = \texttt{BranchBlock}_\ell \left(\vec{Z}_\ell; \vec{\phi}_\ell \right), \tag{3}$$

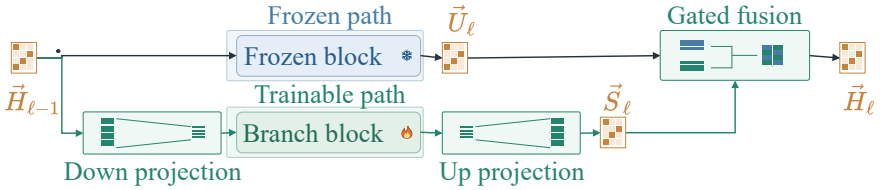
$$\vec{S}_\ell = \texttt{BranchOutput} \left(\vec{Q}_\ell; \vec{\phi}_{\text{out}} \right). \tag{4}$$

The input and output operations each contain trainable layer normalization and a linear projection, with parameters $\vec{\phi}_{\text{in}}$ and $\vec{\phi}_{\text{out}}$ shared across stages. The intermediate states $\vec{Z}_\ell$ and $\vec{Q}_\ell$ have shape

$n \times b$, and the branch output $\vec{S}_\ell$ has shape $n \times d$. Each $\texttt{BranchBlock}_\ell$ has its own parameters $\vec{\phi}_\ell$, placing trainable causal computation at every decoder stage.



(a) Overall computation



(b) Computation within one stage

Figure 1: Progressive Branch-Merge computation schematic. Blue blocks and snowflakes mark frozen computation; green blocks and flames mark trainable computation. Frozen embedding maps input tokens to $\vec{H}_0$. At each of L stages, the frozen backbone block and trainable causal branch read $\vec{H}_{\ell-1}$ in parallel and produce $\vec{U}_\ell$ and $\vec{S}_\ell$. The branch projections change its width before and after its causal block. Shared gated fusion returns $\vec{H}_\ell$, which both paths read at the next stage. The final state passes through frozen final normalization and the output head.

4.2 Stagewise refresh keeps the two paths synchronized

After stage ℓ, the next branch block needs a merged residual representation containing both paths' updates, not only the preceding branch output. PBM therefore merges each backbone–branch pair into $\vec{H}_\ell$. At stage $\ell + 1$, both the frozen block and $\texttt{BranchInput}$ read $\vec{H}_\ell$.

One fusion network is reused at all L stages. It receives the current backbone output $\vec{U}_\ell$, branch output $\vec{S}_\ell$, and stage identity. Separate layer normalizations process the two outputs, while learned width-d depth embeddings $\vec{e}_\ell^{\,\text{backbone}}$ and $\vec{e}_\ell^{\,\text{branch}}$ identify the stage:

$$\vec{A}_\ell = \texttt{BackboneNorm}(\vec{U}_\ell; \vec{\psi}_{\text{backbone}}) + \vec{e}_\ell^{\,\text{backbone}}, \tag{5}$$

$$\vec{B}_\ell = \texttt{BranchNorm}(\vec{S}_\ell; \vec{\psi}_{\text{branch}}) + \vec{e}_\ell^{\,\text{branch}}. \tag{6}$$

Broadcasting each depth embedding across token positions gives $\vec{A}_\ell, \vec{B}_\ell \in \mathbb{R}^{n \times d}$. To compute a correction from both paths, the fusion module concatenates these states and projects them into a value matrix and a gate matrix:

$$[\vec{V}_\ell, \vec{G}_\ell] = \texttt{Linear}_{\text{in}}\left(\vec{A}_\ell \parallel \vec{B}_\ell; \vec{\psi}_{\text{in}}\right), \tag{7}$$

Here $\parallel$ concatenates features, producing an $n \times 2d$ input. The projection returns $n \times 2m$ values split into $\vec{V}_\ell, \vec{G}_\ell \in \mathbb{R}^{n \times m}$, where m is the fusion width. The sigmoid linear unit (SiLU) activates $\vec{V}_\ell$; $\vec{G}_\ell$ gates it elementwise; and the output projection adds a correction to the frozen block output:

$$\vec{H}_\ell = \vec{U}_\ell + \texttt{Linear}_{\text{out}}\left(\texttt{SiLU}(\vec{V}_\ell) \odot \vec{G}_\ell; \vec{\psi}_{\text{out}}\right), \tag{8}$$

The symbol $\odot$ denotes elementwise multiplication. The output projection computes the correction from both path outputs, with the depth embeddings allowing it to vary by stage. The normalization and linear parameters $\vec{\psi}_{\text{backbone}}, \vec{\psi}_{\text{branch}}, \vec{\psi}_{\text{in}}, \vec{\psi}_{\text{out}}$ are shared; only the embeddings and branch blocks are stage specific.

4.3 Zero output initialization preserves the pretrained function

Before learning, fusion should leave the pretrained decoder unchanged. We initialize the weight and bias of $\texttt{Linear}_{\text{out}}$ to zero, so $\vec{H}_\ell = \vec{U}_\ell$ at every stage regardless of the initial branch activations. PBM

therefore starts from exactly the pretrained decoder function. Training updates all added parameters while keeping the pretrained parameters $\vec{\theta}$ fixed.

4.4 A BRANCH-FREE CONTROL REALLOCATES THE SAME PARAMETER BUDGET

To compare causal sequence processing with direct state transformation, the branch-free control removes `BranchInput`, all $\texttt{BranchBlock}_\ell$, and `BranchOutput`. At each stage, its shared gated updater separately normalizes and depth-conditions $\vec{H}_{\ell-1}$ and $\vec{U}_\ell$, concatenates them, and uses the same value, gate, and zero-initialized output operations as PBM. It adds the resulting correction to $\vec{U}_\ell$ to obtain $\vec{H}_\ell$. Widening this updater's fusion hidden width to 7,696 gives 78,898,208 trainable parameters, compared with PBM's 78,901,062. The control retains the frozen backbone and each stage's write point; the comparison matches trainable-parameter count, not executed floating-point operations.

5 EXPERIMENTS

The experiments ask how PBM compares with other adaptations, when its causal branch improves on a wider direct update, and how that benefit varies with sequence difficulty and training cost. We begin with one-pass method breadth, then use fresh arrays and matched controls to locate the value of causal processing, and finally examine which sequences benefit and what training resources it requires.

5.1 EXPERIMENTAL SETUP

We first establish the shared data, endpoint rule, and loss metric, then describe the one-pass comparisons. The paired branch analysis is introduced where its results appear.

Data and task. We select the PBM design on mathematics and transfer it without retuning to physics, chemistry, and biology. For each domain, we adapt an independent copy of the 16-block `pythia-1b-deduped` checkpoint (Biderman et al., 2023); every backbone parameter stays frozen. Mathematics uses UltraData-Math-L2-preview (Wang et al., 2026b); physics uses the filtered arXiv categories listed in Section B.4; chemistry uses the LibreTexts Chemistry portion of ChemPile (Mirza et al., 2025); and biology uses PubMed text from the Common Pile (Kandpal et al., 2025). After source filtering, deduplication, and tokenization, the training streams contain 31,232 mathematics, 18,240 physics, 31,232 chemistry, and 26,304 biology sequences, each 1,024 tokens long. Each domain also has 1,024 validation and 1,024 held-out sequences. One pass reads each training sequence once without wraparound, giving 488, 285, 488, and 411 updates, respectively, at global batch size 64.

Evaluation and endpoint selection. Each run uses fixed randomization and a planned final checkpoint, never one chosen from held-out scores. Validation measurements diagnose training and select the architecture on mathematics; the other domains use that architecture unchanged. For any adapted model, let $\vec{\omega}$ collect its trainable adaptation parameters while $\vec{\theta}$ remains frozen. For sequence $x = (x_1, \ldots, x_n)$, the next-token negative log likelihood (NLL) is

$$r_x = -\frac{1}{n-1} \sum_{t=1}^{n-1} \log p_{\vec{\theta},\vec{\omega}} \left(x_{t+1} \mid x_{\leq t} \right), \tag{9}$$

where lower values mean higher probability for the observed next tokens. For domain q with held-out set $\mathcal{D}_q^{\text{test}}$, the domain mean is

$$\bar{r}_q = \frac{1}{|\mathcal{D}_q^{\text{test}}|} \sum_{x \in \mathcal{D}_q^{\text{test}}} r_x. \tag{10}$$

The cross-domain mean weights the four domain means equally, regardless of training-stream size. Paired intervals use 10,000 resamples of the same sequences from two fixed trained models within each domain. Four-domain intervals equally average the four resampled domain differences; they quantify evaluation-sequence variation, not variation across training runs.

One-pass quality comparison. The primary group freezes Pythia-1B and gives each method approximately 79 million trainable parameters. Its seven alternatives are rank-152 Low-Rank Adaptation (LoRA) (Hu et al., 2022), four-position serial, Houlsby (Houlsby et al., 2019), feed-forward parallel (He et al., 2022), post-block serial, in-feed-forward serial, and one serial full-Transformer-block control. They share the backbone, domain arrays, global batch, one-pass exposure, final-checkpoint rule, and evaluator. Eight additional method families broaden the comparison with one recorded implementation and training configuration each; their trainable counts appear in Table 3.

Optimization and execution. The evaluated PBM has $L = 16$, backbone width $d = 2048$, branch width $b = 360$, and fusion width $m = 5115$, totaling 78,901,062 trainable parameters. Added parameters use AdamW with cosine decay, mixed-precision computation, gradient clipping, and a fixed global batch. The primary one-pass comparisons and PBM-versus-branch-free trajectories run on NVIDIA RTX A6000 processors. The primary comparisons align trainable parameters and data exposure; Section 5.6 reports the corresponding throughput and memory costs.

5.2 One-pass comparison across adaptation methods

We first ask how the complete PBM design compares with other ways to adapt the same frozen backbone after one pass over each domain's training text. Table 1 keeps all fifteen alternatives in one place: seven approximately parameter-matched controls and eight representative configurations with unmatched trainable counts.

Table 1: One-pass held-out NLL (lower is better) with frozen Pythia-1B. Seven unmarked controls are approximately parameter-matched; † marks one pre-specified unmatched configuration per additional family. Trainable counts, including the parallel adapter's domain-specific counts, appear in Tables 2 and 3. Mean equally weights domains; bold and underline mark first and second. Held-out arrays were accessed during earlier exploration.

Method	Math	Physics	Chemistry	Biology	Mean
Low-rank and matrix updates					
LoRA (rank 152)	1.8280	<u>2.4685</u>	2.0139	1.9758	2.0715
Weight-decomposed LoRA (DoRA)†	1.8350	2.4726	2.0294	1.9762	2.0783
Adaptive LoRA (AdaLoRA)†	1.8969	2.5085	2.1339	1.9896	2.1322
Butterfly orthogonal tuning (BOFT)†	1.9443	2.5325	2.1801	2.0049	2.1655
Random-matrix scaling (VeRA)†	1.9477	2.5403	2.1829	2.0087	2.1699
Activation, bias, prompt, and prefix updates					
Activation scaling (IA3)†	1.9380	2.5275	2.1745	2.0030	2.1607
Bias-only tuning (BitFit)†	1.9270	2.5128	2.1635	1.9978	2.1503
Input soft-prompt tuning†	1.9017	2.5061	2.1381	1.9867	2.1332
Layerwise key-value prefix tuning†	5.6541	5.9787	5.4601	6.0284	5.7803
Adapters and additional causal computation					
Houlsby adapter	1.8385	2.4760	2.0334	1.9767	2.0811
Feed-forward parallel adapter	<u>1.8169</u>	2.4693	<u>1.9770</u>	<u>1.9757</u>	<u>2.0597</u>
Post-block serial adapter	1.8289	2.4731	2.0056	1.9766	2.0711
In-feed-forward serial adapter	1.8198	2.4689	1.9849	**1.9756**	2.0623
Four-position serial adapter	1.8363	2.4816	2.0187	1.9778	2.0786
Serial full Transformer block	1.8809	2.5170	2.0967	1.9853	2.1200
PBM (ours)	**1.8048**	**2.4640**	**1.9316**	1.9760	**2.0441**

PBM has the lowest four-domain mean, 2.0441, compared with 2.0597 for the strongest matched alternative, the feed-forward parallel adapter. Their paired difference is 0.0156, with a 95% sequence-resampling interval of $[0.0146, 0.0167]$ for these fixed endpoints. PBM also has the lowest loss in mathematics, physics, and chemistry; in biology, the in-feed-forward serial adapter is lower by 0.0004. Among the eight representative configurations, Weight-Decomposed Low-Rank Adaptation (DoRA) is closest at 2.0783 mean NLL. The one-pass comparison uses fixed configurations rather than equally searched optima; Section A records its scope. We next test the parameter allocation directly against a wider state updater on newly sampled evaluation arrays.

5.3 A CAUSAL BRANCH IMPROVES QUALITY IN THREE DOMAINS

The one-pass ranking shows PBM's overall quality but does not isolate how its branch uses trainable parameters. PBM and the nearly parameter-matched branch-free control therefore complete one additional pass over the same training arrays, restoring their first-pass optimizer states and using fixed endpoints. We score them on newly sampled arrays of 1,024 sequences per domain, fixed before either endpoint was evaluated. Figure 2 plots branch-free minus PBM NLL; a positive value means that PBM assigns higher probability to the observed text.

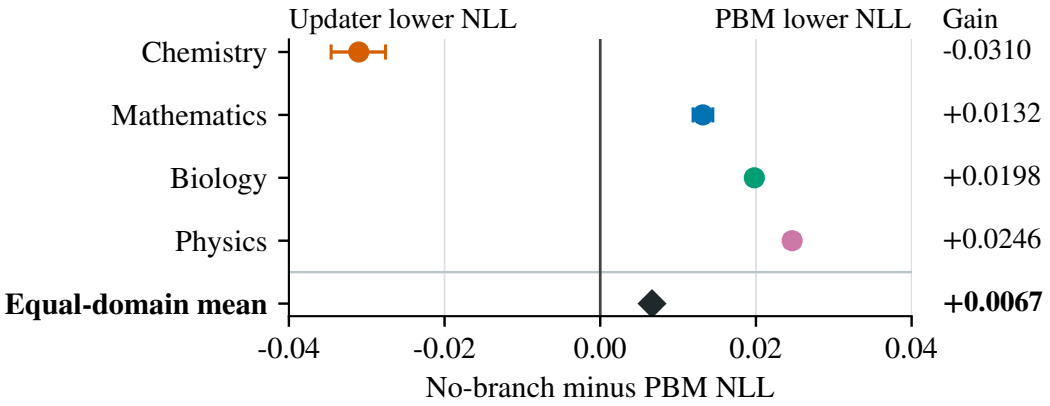


Figure 2: PBM gain over a separately trained branch-free updater on 1,024 fresh paired sequences per domain. Positive values mean lower PBM NLL. Bars on the four domain rows are 95% paired sequence-resampling intervals from 10,000 draws for fixed endpoints; the diamond is the equal-weight mean of the unrounded domain gains, shown without an interval. Exact losses appear in Table 5.

PBM has lower NLL than the branch-free updater on mathematics, biology, and physics, while the updater has lower NLL on chemistry. The equal-weight four-domain mean is 2.0781 for PBM and 2.0847 for the updater, a paired advantage of 0.0067 with interval $[0.0057, 0.0076]$. The chemistry reversal identifies a useful distinction: adding causal processing throughout depth can outperform a wider direct update, but a matched parameter budget does not make it the best allocation in every domain.

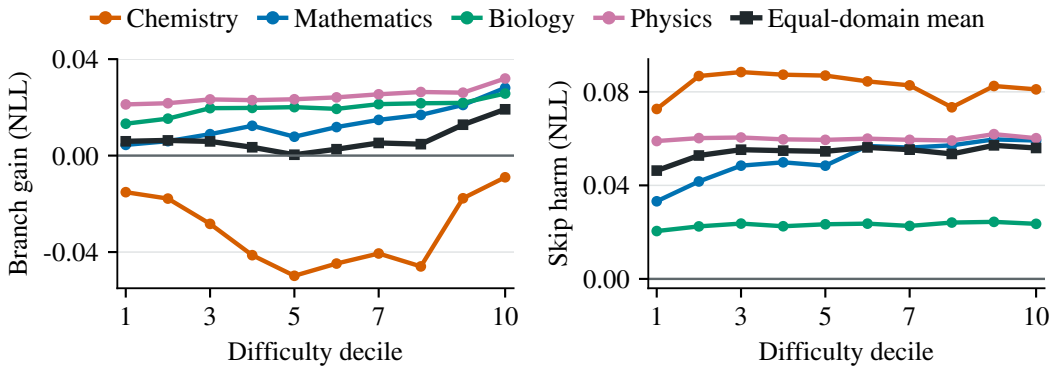
5.4 CAUSAL PROCESSING AND STAGEWISE REFRESH CONTRIBUTE TO THE DESIGN

After observing lower PBM loss than the wider direct updater in three domains, we ask which parts of PBM's stagewise computation contribute to this difference. PBM feeds the latest merged state to its next causal branch rather than carrying only a separate branch state forward. In local mathematics comparisons, refreshing from the merged state lowers NLL from 1.8272 to 1.8135; depth-conditioned fusion lowers NLL by 0.0010, and a separate gated-fusion comparison lowers it by 0.0065. Table 6 preserves the corresponding reference and updated endpoints. The refresh comparison has the largest local difference, supporting repeated access to the merged state as a consequential part of this design. The three local improvements are not additive because their references differ.

A separate matched structural comparison distinguishes causal token mixing from the branch's read position. On a fixed one-pass evaluation array, a stage-specific branch that processes tokens independently has 0.0007 higher four-domain mean NLL than PBM, while a causal branch that reads each frozen block's output has 0.0023 lower NLL. Together, these results locate a measurable contribution in trainable causal mixing across depth and show that the best read position is a separate design choice; the full comparison appears in Table 7.

5.5 Relative branch gain rises with measured sequence difficulty

The allocation and structural comparisons leave a more specific question: which evaluation sequences receive the larger loss reduction from distributed causal processing? Within each domain, we rank the fresh sequences by the median loss of thirteen other trained adaptations and divide them into ten difficulty groups. We also bypass PBM's trained branch blocks while retaining its projections and fusion; the resulting loss increase measures how much this trained PBM uses those blocks. Figure 3 shows both the PBM-versus-updater difference and this branch-skipping effect across the ten groups.



(a) Across architectures (b) Within PBM

Figure 3: Branch value by within-domain difficulty decile (1 is easiest; 102–103 sequences per group), using median loss across thirteen non-PBM baselines to order sequences. Left: branch-free minus PBM NLL. Right: loss from skipping PBM's trained branch blocks. Black is the equal-weight domain mean. Vertical scales differ; curves are unsmoothed.

The hardest decile receives 0.01327 more branch gain than the easiest on average, with a paired sequence-resampling interval of $[0.00956, 0.01697]$; the hardest-minus-easiest difference is positive in all four domains. Mathematics shows the largest rise, 0.0236; chemistry's gain also rises with difficulty, although the direct updater retains lower loss across its deciles.

Skipping the trained branch blocks raises PBM's loss in every domain. The mean skip harm is 0.00967 larger in the hardest than in the easiest decile, with interval $[0.00719, 0.01218]$, and mathematics again shows the largest rise. Together, the two panels separate a branch that PBM actually uses from a branch that outperforms another trained architecture: the former holds even in chemistry, whereas the latter does not. This distinction shows why the value of a trainable branch is best read from its effect relative to another trained allocation, not only from whether the model uses it.

5.6 Quality and training resources

The added causal blocks yield a quality–resource trade-off that matters when choosing an adaptation. Figure 4 shows mathematics endpoint quality, full-run input-token throughput, and peak reserved memory for PBM and four controls on the same accelerator; exact values appear in Table 8.

Compared with the feed-forward parallel adapter, PBM lowers mathematics NLL by 0.0121 while processing 42% as many training tokens per second and reserving 5.09 GiB more memory. The added causal depth thus provides a quality-oriented allocation under the reported budget, with a measurable training cost.

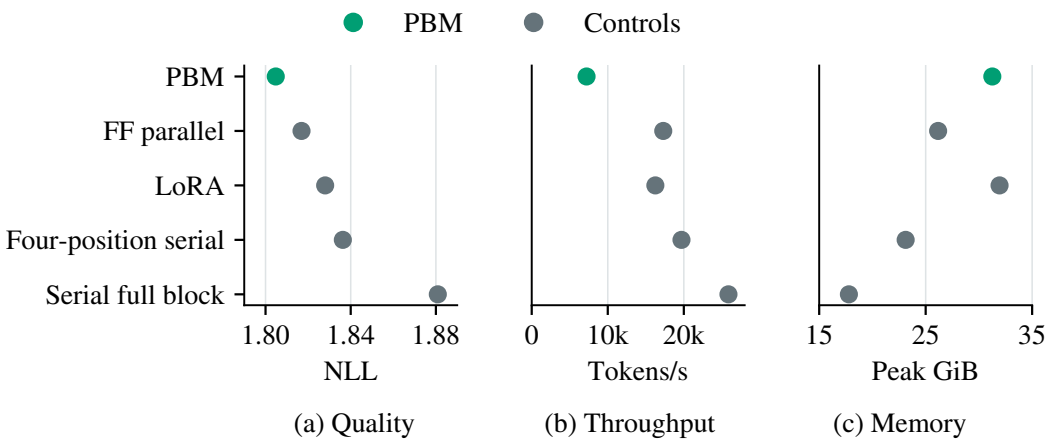


Figure 4: Mathematics one-pass quality and training resources on the same NVIDIA RTX A6000. Methods share row order, but panel scales differ. Lower NLL and memory are better; higher full-run input-token throughput is better. Green marks PBM; gray marks controls. Exact values appear in Table 8.

6 Discussion

PBM shows that the computation learned with an adaptation budget matters alongside the size of that budget. With similar trainable-parameter counts and data exposure, its stagewise causal branch lowers loss relative to a wider direct state update in three domains and yields a larger relative gain on harder sequences. The tokenwise control supports a contribution from cross-token processing, while the same-stage serial control separates that contribution from the exact state the branch reads. Together, these comparisons make trainable causal depth a concrete design choice for frozen-backbone adaptation, rather than treating added parameter count as the whole design.

The two difficulty analyses add different information to the overall comparison. PBM's advantage over a separately trained updater rises with measured difficulty, and the loss from bypassing PBM's trained branch blocks rises as well. The chemistry results distinguish these effects: PBM uses its branch even though the direct updater remains better there. Difficulty-stratified evaluation therefore reveals where an allocation helps more without treating measured difficulty as the cause of that gain.

PBM provides this quality gain as a training-time trade-off: in the measured mathematics runs, it has lower throughput and higher peak memory than a feed-forward parallel adapter. The measured quality, throughput, and memory differences make resource use and sequence-level variation complementary to the overall loss comparison.

7 Conclusion

We introduced Progressive Branch-Merge, a frozen-backbone adaptation that places trainable causal processing beside every decoder stage and returns both paths to a shared state. In the studied Pythia-1B domains, PBM has the lowest mean among the tested one-pass methods and lowers fresh-array next-token loss relative to a nearly parameter-matched direct updater in three of four domains. Its relative gain grows on harder sequences, and matched structural controls identify causal token mixing as a useful part of the design while leaving read position open to improvement. The broader contribution is a concrete way to distribute learnable sequence transformations through frozen depth and an empirical account of when that allocation improves domain modeling.

REFERENCES

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023.

Shoufa Chen, Chongjian Ge, Zhan Tong, Jiangliu Wang, Yibing Song, Jue Wang, and Ping Luo. AdaptFormer: Adapting vision transformers for scalable visual recognition. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.13535.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8342–8360. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.740. URL https://aclanthology.org/2020.acl-main.740/.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2110.04366.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

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. In *International Conference on Learning Representations*, 2022.

Nikhil Kandpal, Brian Lester, Colin A. Raffel, Sebastian Majstorovic, Stella Biderman, Baber Abbasi, Luca Soldaini, Enrico Shippole, A. Feder Cooper, Aviya Skowron, Shayne Longpre, Lintang Sutawika, Alon Albalak, Zhenlin Xu, Guilherme Penedo, Loubna Ben Allal, Elie Bakouch, John Pressman, Honglu Fan, Dashiell Stander, Guangyu Song, Aaron Gokaslan, John Kirchenbauer, Tom Goldstein, Brian Bartoldson, Bhavya Kailkhura, and Tyler Murray. The common pile v0.1: An 8TB dataset of public domain and openly licensed text. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-1918. URL https://doi.org/10.52202/085713-1918.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/0cde695b83bd186c1fd456302888454c-Abstract-Conference.html.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Adrian Mirza, Nawaf Alampara, Martiño Rĩos-García, Mohamed Abdelalim, Jack Butler, Bethany Connolly, Tunca Dogan, Marianna Nezhurina, Bünyamin Şen, Santosh Tirunagari, Mark Worrall, Adamo Young, Philippe Schwaller, Michael Martin Pieler, and Kevin Maik Jablonka. ChemPile: A 250 GB diverse and curated dataset for chemical foundation models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=l8Mz3dcK1S.

Rohan Baskar Prabhakar, Hengrui Zhang, and David Wentzlaff. Kraken: Inherently parallel transformers for efficient multi-device inference. *arXiv preprint arXiv:2408.07802*, 2024.

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

Antoine Simoulin and Benoit Crabbé. How many layers and why? an analysis of the model depth in transformers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pp. 221–228. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-srw.23. URL https://aclanthology.org/2021.acl-srw.23/.

Shixiang Song, He Li, Zitong Wang, Boyi Zeng, Feichen Song, Yixuan Wang, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. AdaPonderLM: Gated pondering language models with token-wise adaptive depth. *arXiv preprint arXiv:2603.01914*, 2026. URL https://arxiv.org/abs/2603.01914.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. LST: Ladder side-tuning for parameter and memory efficient transfer learning. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2206.06522.

Chong Wang, Nan Du, Tom Gunter, Tao Lei, Kulin Seth, Senyu Tong, Jianyu Wang, Guoli Yin, Xiyou Zhou, Kelvin Zou, and Ruoming Pang. Parallel track transformers: Enabling fast GPU inference with reduced synchronization. *arXiv preprint arXiv:2602.07306*, 2026a.

Yudong Wang, Zixuan Fu, Hengyu Zhao, Chen Zhao, Chuyue Zhou, Xinle Lin, Hongya Lyu, Shuaikang Xue, Yi Yi, Yingjiao Wang, Zhi Zheng, Yuzhou Zhang, Jie Zhou, Chaojun Xiao, Xu Han, Zhiyuan Liu, and Maosong Sun. Data science and technology towards AGI part I: Tiered data management. *arXiv preprint arXiv:2602.09003*, 2026b. URL https://arxiv.org/abs/2602.09003.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ying Shan, and Ping Luo. LLaMA Pro: Progressive LLaMA with block expansion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Jeffrey O. Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-tuning: A baseline for network adaptation via additive side networks. In *European Conference on Computer Vision*, pp. 698–714. Springer, 2020. URL https://www.ecva.net/papers/eccv_2020/papers_ECCV/html/1104_ECCV_2020_paper.php.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

REPRODUCIBILITY STATEMENT

Sections 3, 4 and 5.1 define the model, PBM, branch-free control, data exposure, endpoints, metrics, and comparison roles; Section B records architecture geometry, domain sizes, fresh-array and structural-control comparisons, intervention protocols, software, and artifact hashes. All 32 eight-family endpoints, four serial full-Transformer-block endpoints, 60 fresh-array model scores, and 12 intervention scores across three conditions and four domains completed. The supplied Secure

Hash Algorithm 256-bit (SHA-256) file verifies the backbone, tokenizer, comparison archives, fresh-array manifest, and branch-analysis summaries. A training-only PBM implementation and usage instructions are available through the anonymous code mirror. It provides the model, optimizer, and one- and two-pass training entry point; datasets, checkpoints, evaluation scripts, and complete experiment orchestration are not part of this release.

A LIMITATIONS

The study covers one Pythia-1B backbone, four scientific-text domains, and one reported training trajectory per configuration. Paired intervals vary fixed evaluation sequences, not training runs, so they do not establish stability across initialization or optimization trajectories.

The one-pass main comparison uses held-out arrays that had been accessed during earlier exploratory work. The branch analysis uses newly sampled arrays fixed before model scoring, with no complete-sequence overlap with earlier evaluation or training arrays; document-level isolation is not established.

The branch comparison repeats the same training arrays for a second pass rather than adding independent text. Its empirical difficulty measure is the median loss of thirteen selected stable baselines, not an intrinsic measure of domain complexity. Only four domain means support the observed cross-domain rank ordering, and difficulty is not experimentally manipulated; token entropy, long-range dependence, or corpus regularity could also vary with the observed branch gain.

The primary comparisons match trainable parameters and data exposure rather than floating-point operations or wall-clock time. The approximately parameter-matched baselines use fixed configurations, but the search effort that produced them was not matched to PBM's. The additional eight families use one representative configuration each rather than exhaustive per-method tuning. The one-block serial baseline in the main table tests one placement and width. A separate same-stage serial causal-branch control, with one branch block after each frozen block, performs better than PBM under its matched protocol, so the results do not establish that reading before the frozen block is the best branch position. PBM also lacks an incremental key-value cache, so the reported training measurements do not establish cached-generation cost.

B REPRODUCIBILITY

This appendix fixes the evaluated PBM and branch-free configurations, comparison geometry, domain exposure, fresh-array analysis, and artifact scope.

B.1 EVALUATED PBM CONFIGURATION

The reported endpoints use the following joint configuration.

- **Backbone:** Pythia-1B deduplicated checkpoint with 16 decoder blocks and hidden width 2,048.
- **Frozen identities:** `evidence_checksums.txt` covers the backbone, tokenizer, comparison archives, fresh-array manifest, branch-analysis summaries, and structural-control summary; individual run identities remain in experiment records.
- **Branch:** 16 unique width-360, five-head causal Transformer blocks with shared 2,048-to-360 and 360-to-2,048 projections.
- **Fusion:** one shared depth-conditioned gated module invoked after every backbone block, with hidden width 5,115 and a zero-initialized output projection.
- **Trainable parameters:** 78,901,062, including 26,437,456 in the branch blocks and shared interface and 52,463,606 in shared gated fusion.
- **Branch-free control:** no branch blocks or branch projections; one shared depth-conditioned gated updater of width 7,696 reads each frozen block's input and output, for 78,898,208 trainable parameters.
- **Training exposure:** the one-pass quality comparison traverses each domain array once; the branch analysis restores the first-pass endpoints and traverses the same array once more in the same order.
- **Training code:** the anonymous code mirror provides the PBM model, optimizer, and example one- and two-pass commands. It is a training-only reference implementation, not the complete experiment orchestration.

- **Objective:** next-token language modeling over 1,023 real targets per sequence.
- **Checkpoint rule:** complete planned endpoint for each one-pass or two-pass comparison; held-out values do not select models or checkpoints.

B.2 MATCHED-BASELINE GEOMETRY

The one-pass main table includes the seven approximately parameter-matched alternatives below alongside eight representative configurations with unmatched trainable counts; the separate branch-free control isolates parameter allocation after two passes.

Table 2: Trainable count and placement for the seven one-pass parameter-matched alternatives and the separate two-pass branch-free control.

Method (trainable M)	Placement or update surface
LoRA, rank 152 (79.69)	Four linear maps in each of 16 decoder blocks (64 low-rank modules)
Four-position serial adapter (79.78)	Width 4,867 after backbone blocks 5, 9, 13, and 16
Houlsby adapter (78.86)	Width 601 after attention and feed-forward outputs in every block
Feed-forward parallel adapter	Width 1,217 on mathematics: 79,809,552 trainable parameters. Width 1,203 on physics, chemistry, and biology: 78,891,824 trainable parameters. Parallel to every feed-forward sublayer.
Post-block serial adapter (78.89)	Width 1,203 after every complete backbone block
In-feed-forward serial adapter (78.89)	Width 1,202 inside every feed-forward sublayer
Serial full Transformer block (79.73)	Full-width causal attention and width-15,360 feed-forward layer after decoder block 12
Branch-free gated updater (78.90)	Shared width-7,696 gated update from each frozen block's input and output

B.3 REPRESENTATIVE METHOD COUNTS AND PUBLIC CHECKPOINT SCALE CONTEXT

The eight representative adaptation configurations appear in Table 1; their trainable counts are listed below without implying matched counts or equal tuning effort across families.

Table 3: Added trainable parameters for the eight representative one-pass adaptation configurations. Counts are rounded as displayed.

Method	Added trainable
Weight-decomposed LoRA (DoRA)	80.0M
Adaptive LoRA (AdaLoRA)	100.7M
Butterfly orthogonal tuning (BOFT)	1.21M
Random-matrix scaling (VeRA)	311K
Activation scaling (IA3)	229K
Bias-only tuning (BitFit)	362K
Input soft-prompt tuning	524K
Layerwise key-value prefix tuning	34.8M

The unadapted Pythia checkpoints below provide model-scale context only.

Table 4: Unadapted public Pythia checkpoints on the same one-pass held-out arrays. These models receive no domain training and have zero added trainable parameters; they provide scale context, not an adaptation-graph control. NLL is lower when the observed text is assigned higher probability; Mean equally weights the four domains. Bold marks the lowest NLL among these public checkpoints.

Checkpoint	Math	Physics	Chemistry	Biology	Mean
Pythia 410M	2.1982	2.8943	2.4667	2.2584	2.4544
Pythia 1B	2.0110	2.6423	2.2392	2.0741	2.2416
Pythia 1.4B	1.9578	2.5613	2.1776	2.0312	2.1820
Pythia 2.8B	1.8463	2.4816	2.0442	1.9256	2.0744
Pythia 6.9B	**1.8058**	**2.3832**	**1.9978**	**1.9021**	**2.0222**

B.4 One-pass domain exposure and evaluation sets

Training and evaluation sizes. Every row uses sequences of 1,024 tokens.

Domain	Train sequences	Updates	Validation	Held out
Mathematics	31,232	488	1,024	1,024
Physics	18,240	285	1,024	1,024
Chemistry	31,232	488	1,024	1,024
Biology	26,304	411	1,024	1,024

The mathematics stream is UltraData-Math-L2-preview. The physics stream includes arXiv categories under `physics.*`, `astro-ph.*`, `cond-mat.*`, `hep-*`, and `nucl-*`, plus `gr-qc` and `quant-ph`; `physics.chem-ph` and `cond-mat.mtrl-sci` are excluded. Chemistry uses LibreTexts Chemistry from ChemPile, and biology uses PubMed text from the Common Pile.

B.5 Two-pass branch analysis and fresh arrays

PBM, the branch-free control, and the thirteen stable non-PBM difficulty baselines are evaluated at fixed endpoints after one additional pass over their original training arrays. The difficulty set is Adaptive LoRA, bias-only tuning, butterfly orthogonal tuning, Weight-Decomposed LoRA, feed-forward parallel, Houlsby, activation scaling, in-feed-forward serial, Low-Rank Adaptation, input soft-prompt tuning, four-position serial, post-block serial, and random-matrix scaling. The unstable fixed prefix-tuning configuration and the earlier PBM variant are excluded before fresh-array scores are read. The serial full-Transformer-block control appears in the one-pass main comparison but is not among these thirteen models used to assign sequence difficulty.

Each fresh evaluation array contains 1,024 sequences of 1,024 tokens. The array is fixed before model evaluation and has zero complete-sequence overlap with the training, validation, original held-out, and earlier branch-analysis arrays; document-level overlap is not established. Empirical difficulty is the per-sequence median NLL across the thirteen baselines. Within each domain, the 1,024 fresh sequences are sorted by empirical difficulty and split into ten groups of 102–103 sequences. The analysis uses these difficulty deciles, 10,000 domain-stratified bootstrap repetitions, and a fixed bootstrap randomization state. The primary intervention skips the trained branch Transformer blocks while retaining the branch interface and gated fusion; branch-output zeroing and within-batch mismatch are secondary checks and do not determine the reported confirmation decision.

The fresh-array manifest, architecture-comparison summary, intervention summary, and final audit have SHA-256 identities in `evidence_checksums.txt`. The final audit verifies four arrays, 60 fixed model scores, four primary branch-skipping confirmation scores, and the frozen summary hashes; all 12 intervention runs across the three conditions are preserved in the run records. The paired four-domain mean intervals reported in Sections 5.2 and 5.3 were computed from the aligned per-sequence loss files for the fixed endpoints: 10,000 within-domain paired resamples of 1,024 sequences, equal weighting across four domains, and randomization seed 20260925. The four per-domain intervals in Figure 2 use 10,000 paired sequence resamples per domain, with recorded seeds 20260925–20260928; the derived CSV and source-file hashes accompany the figure script.

B.6 DETAILED DOMAIN, DESIGN, AND RESOURCE VALUES

The domain endpoints and branch-skipping measurements underlying Figures 2 and 3 are listed below. The difficulty curves additionally use the ten within-domain sequence groups described above.

Table 5: Fresh-array two-pass domain values. Difficulty is the domain mean of per-sequence median NLL across thirteen non-PBM baselines. "No branch" is a separately trained updater. Gain is no-branch minus PBM loss; skip harm is PBM's loss increase when branch blocks are bypassed. H–E is the hardest-minus-easiest decile difference. Values are rounded independently after computing differences.

Domain	Difficulty	PBM	No branch	Gain	Gain H–E	Skip harm	Skip H–E
Chemistry	1.9087	1.6661	**1.6351**	-0.0310	0.0062	0.0826	0.0084
Mathematics	1.9166	**1.8820**	1.8952	0.0132	0.0236	0.0510	0.0260
Biology	1.9967	**1.9968**	2.0166	0.0198	0.0125	0.0231	0.0031
Physics	2.7663	**2.7674**	2.7921	0.0246	0.0107	0.0600	0.0012

The following three rows record separate local mathematics design comparisons, not a factorial decomposition; their reference endpoints are not identical.

Table 6: Local design-path comparisons on held-out mathematics text. Improvement is reference NLL minus updated NLL. An intervening optimizer-setting selection separates the gated-fusion reference from the preceding row.

Named choice	Reference NLL	Updated NLL	Improvement
Refresh from the latest merged state	1.8272	1.8135	0.0137
Condition shared fusion on depth	1.8135	1.8125	0.0010
Use gated fusion	1.8113	**1.8048**	0.0065

The next comparison keeps the one-pass domain exposure and nearly the same trainable-parameter budget while changing the branch computation or its read point. The stage-specific tokenwise control replaces the branch's causal token mixing with per-token feed-forward blocks. The same-stage serial control retains the causal branch and fusion but reads each frozen block's output rather than its input. These controls and PBM are evaluated at fixed endpoints on an additional array of 1,024 sequences per domain; this array differs from those used for Table 1 and Figure 2.

Table 7: One-pass structural controls on a separate fixed four-domain evaluation array. The difference is control minus PBM NLL, so positive values mean lower PBM loss. All values are rounded from the frozen analysis summary.

Configuration	Mean NLL	Difference from PBM
PBM	2.1015	–
Stage-specific tokenwise branch	2.1022	$+0.0007$
Same-stage serial causal branch	**2.0992**	-0.0023
Branch-free direct updater	2.1040	$+0.0025$

For the two branch-structure controls, 10,000 paired resamples of the fixed evaluation sequences give 95% intervals of $[0.000414, 0.001048]$ for tokenwise minus PBM and $[-0.002588, -0.002035]$ for same-stage serial minus PBM. These intervals measure variation across evaluation sequences, not across independent training runs. The frozen comparison summary is included with the figure data and identified in `evidence_checksums.txt`.

The measured mathematics resource values behind Figure 4 use the same accelerator, sequence length, and global batch as the one-pass quality comparison.

Table 8: Mathematics one-pass endpoint NLL, full-run input-token throughput, and peak reserved graphics memory.

Method	NLL	Tokens/s	Peak GiB
PBM	**1.8048**	7,210	31.26
Feed-forward parallel adapter	1.8169	17,295	26.17
Low-Rank Adaptation	1.8280	16,257	31.94
Four-position serial adapter	1.8363	19,692	23.12
Serial full Transformer block	1.8809	25,869	17.79

B.7 Software and artifact scope

The eight-family baselines use Python 3.9, PyTorch 2.5.1 with Compute Unified Device Architecture (CUDA) 12.4 support, Transformers 4.57.6, and Parameter-Efficient Fine-Tuning 0.17.1; PBM and the matched controls use the same PyTorch and Transformers runtime family. The one-pass primary comparisons, PBM and branch-free trajectories, and fresh-array scoring use NVIDIA RTX A6000 graphics processors. Some second-pass difficulty-baseline continuations use NVIDIA L40 or RTX 4090 graphics processors; their model, data, endpoint, and software identities remain fixed and are recorded with the run artifacts.

PBM analyses remain with their delivery artifacts; `evidence_checksums.txt` covers the backbone, tokenizer, five broader-comparison deliverables, the serial full-Transformer-block control, and the fresh-array branch-analysis summaries. All 780 members of the broader-comparison archive and all 90 members of the serial-control archive were verified by path, byte count, and hash; the branch-analysis audit reports all required fresh-array checks as passed.

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.