SOME TOKENS THINK, OTHERS REMEMBER: SEPARATING COMPUTE AND CAPACITY IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped transformers apply a shared block multiple times and have emerged as a parameter-efficient route to scaling compute in language models. However, at fixed FLOPs a looped model has strictly less capacity than a baseline transformer. We argue that *compute*, the number of sequential operations applied to a hidden state, and *capacity*, the parameters available at a single step, are distinct scaling axes that standard transformer layers conflate. We propose a *dual-path block* that exposes both axes as parallel pathways within a single layer: a *deep* sublayer re-applied K times with shared parameters, and a *wide* sublayer with an enlarged feed-forward network applied once, combined by independent per-token gates. At two FLOP budgets, the best dual-path configuration surpasses both iso-FLOP single-axis controls on aggregate language modelling, commonsense, and math metrics, while using 17–33% *fewer* parameters than the width-scaled baseline. The learned gates are directly interpretable and show systematic per-token allocation: function words and lexical content trend wide, while punctuation, symbols, and arithmetic tokens trend deep. Our results suggest that compute and capacity can be scaled separately, with the model balancing the two token by token.

1 INTRODUCTION

Looped (or recursive) transformers re-apply a shared block K times, trading parameter count for sequential compute (Dehghani et al., 2019; Geiping et al., 2025; Saunshi et al., 2025; Zhu et al., 2025). The appeal is parameter efficiency: L layers looped K times reach the effective depth of KL unshared layers at $1/K$ the parameters, and recent work shows this is enough to recover much of the reasoning performance of a deeper unshared stack (Saunshi et al., 2025; Zhu et al., 2025). However, at fixed FLOPs, a looped model has strictly less capacity than an unshared stack of comparable compute, and the gap shows up empirically on tasks that depend on stored knowledge (Frey et al., 2026; Zhu et al., 2025).

Looping and width scaling therefore sit on two qualitatively different axes of a transformer layer. Looped models use *compute* to increase the number of sequential operations applied to a hidden state, while width-scaled models increase *capacity*, the parameters available at a single step. Standard architectures conflate the two, with every token paying the same cost on both. A looped model puts its whole per-layer feed-forward network (FFN) budget on compute and a width-scaled FFN puts it all on capacity. Neither lets a token that needs more sequential refinement get it without also paying for capacity it does not need, or the other way around.

Recent work relaxes this one axis at a time. Mixture-of-Experts (Shazeer et al., 2017; Fedus et al., 2022) routes tokens to a subset of similar experts, scaling capacity sub-linearly in compute. Mixture-of-Depths (Raposo et al., 2024) routes tokens to skip or apply a layer, varying effective depth per position. Mixture-of-Recursions (Bae et al., 2025) varies the number of loop iterations per token, and adaptive halting mechanisms in looped models do the same via learned stopping (Graves, 2016; Banino et al., 2021; Frey et al., 2026). Each of these routes along a single axis, i.e. which experts, how many loop iterations, or whether to apply a layer at all.

We propose a transformer layer that exposes the two axes separately within itself. Each *dual-path block* contains two parallel sublayers: a *deep* sublayer applied K times with shared parameters (a loop, as above), and a *wide* sublayer with an enlarged feed-forward dimension applied once. A

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

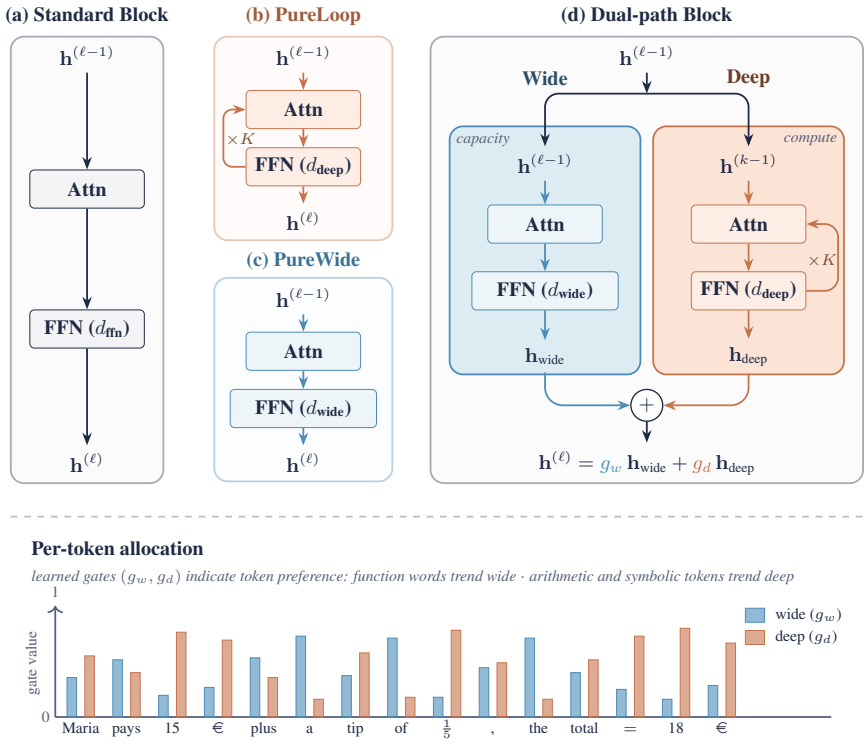


Figure 1: Block architectures. (a) Standard transformer block. (b) PURELOOP: a shared block re-applied K times. (c) PUREWIDE: a single block with enlarged FFN. (d) Dual-path block, which runs (c) and (b) in parallel on the same input and combines them via two per-token sigmoid gates g_w, g_d. Bottom panel: schematic of the learned gates across sequence tokens.

learned per-token gate combines them. We train the dual-path block across different wide and deep ratios (α) and the iso-FLOP controls for each axis separately. We show that:

- At both FLOP budgets, the best dual-path configuration beats both single-axis controls on language-modelling, commonsense, and math evaluations *while using fewer parameters than the width-scaled control*.
- The gate does not collapse to one path. Its allocation depends on the layer and the task. The middle of the stack leans wide and the last layers lean deep. Function words and lexical content (verbs, adjectives) lean wide, while punctuation, symbols, and arithmetic tokens lean deep.
- The per-token routing affects the output. Forcing arithmetic tokens through the wide path raises GSM8K loss while leaving TriviaQA unchanged, while forcing verbs, adjectives, and function words through the deep path hurts TriviaQA but not GSM8K.

2 RELATED WORK

Looped and recursive transformers. Re-applying a shared block across depth has been studied as a parameter-efficient route to scaling transformers since the release of Universal Transformers (Dehghani et al., 2019). The idea has been revived recently in language modelling: looped decoders scale test-time compute by re-applying a shared block (Geiping et al., 2025; Saunshi et al., 2025; Zhu et al., 2025; Jeddi et al., 2026; Shomali et al., 2026) which saves parameters while scaling compute. The cost of the parameter saving is reduced capacity. Zhu et al. (2025) show that looped models match standard transformers on knowledge manipulation but not on per-parameter memorisation, and Frey et al. (2026) report a corresponding empirical pattern downstream: adaptive looping improves

mathematical reasoning but leaves commonsense benchmarks largely flat. Both observations show that looping buys compute at the cost of capacity.

Per-token compute allocation. A separate line of work allocates compute adaptively at the token level. Mixture-of-Experts (Shazeer et al., 2017; Fedus et al., 2022) routes each token to a small subset of many parallel experts of the same shape, decoupling parameter count from per-token compute. Mixture-of-Depths (Raposo et al., 2024) routes tokens to skip or apply a layer, varying effective depth per position. Mixture-of-Recursions (Bae et al., 2025) extends this to looped stacks by varying the number of shared-block applications per token. VersatileFFN (Nie et al., 2026) and MoUE (Chen et al., 2026) also expose both axes per token. However, both VersatileFFN and MoUE enforce strict parameter reuse (mathematically slicing a shared FFN or reusing a universal expert pool), whereas our dual-path block introduces dedicated capacity parameters via a separate, enlarged wide FFN. Second, both rely on conditional routing to maintain a fixed activation budget. In contrast, our updates are observed at every token and every layer, making our gate's value a direct, un-collapsed read-out of how the trained model chose to allocate its budget between two options that were both processed.

While MoE and our dual-path block both attach a learned router to a transformer layer, they route over different sets of options. In MoE, the router selects among N feed-forward experts that share the same architecture but hold different learned weights; the choice is over which parameters a token sees. In the dual-path block, the router weighs two sublayers that differ in kind: one is a shared-parameter sublayer applied K times, the other is a wider sublayer applied once. The choice is instead over what type of update a token receives. Furthermore, MoE routers are top-k sparse, whereas our gate is dense. Because both paths are always evaluated in our block, we do not skip compute but re-allocate it. This also means the gate's value is a direct read-out of how the trained model chose to allocate between two options that were both processed. Finally, because the two mechanisms target different axes, they are compositional rather than competing. In theory, a MoE layer could be placed inside the wide path of a dual-path block, with the gate selecting whether to route a token through looped compute or through routed capacity.

Memory-augmented transformers. A complementary route to recovering capacity in parameter-shared models is to attach learned memory banks the model can query, as in product-key memory layers (Lample et al., 2019) and persistent memory in attention (Sukhbaatar et al., 2019). Frey et al. (2026) combine adaptive looping with per-layer and global memory banks and find that memory closes part of the commonsense gap that looping alone cannot bridge. The dual-path block addresses the same capacity bottleneck from the architectural side: rather than adding a separately queried memory module that scales weakly, it adds a parallel wide FFN sublayer inside each layer and lets a per-token gate decide how much of each token's update comes from looped compute vs. wider parameters.

3 METHOD

3.1 PROBLEM STATEMENT AND NOTATION

Scaling a transformer layer generally involves either expanding *capacity* (adding parameters, typically via a wider feed-forward network) or extending *compute* (increasing sequential operations on the hidden state, usually via more layers or by recursively re-applying a shared one). We study an architecture that exposes the two axes *separately within each layer* and let the model decide, per token, how to allocate between them.

3.2 BASELINE

Given an input tensor $x \in \mathbb{R}^{B\times T\times d}$, where B denotes the batch size, T represents the sequence length, and d is the model dimension, a standard transformer sublayer Φ is defined as

$$\begin{aligned} u &= x + s \cdot \mathrm{Attn}(\mathrm{RMSNorm}(x)) \,, \\ \Phi(x; s) &= u + s \cdot \mathrm{FFN}(\mathrm{RMSNorm}(u)) \,, \end{aligned} \tag{1}$$

where $s \in \mathbb{R}_{>0}$ is a scalar gain on the sublayer contribution. In a standard transformer $s = 1$; in our dual-path block, s is learned per recursion step and per path.

3.3 DUAL-PATH BLOCK

Each block exposes the two scaling axes as two parallel sublayers that share the same input x, as illustrated in Figure 1. One sublayer adds sequential compute by re-applying itself K times with shared parameters; the other adds capacity by using a wider FFN and is applied once. A per-token gate combines them.

Deep path (compute). The deep sublayer Φ_{deep} uses the FFN hidden dimension d_{deep} and is applied K times iteratively with shared parameters:

$$h^{(k)} = \Phi_{\text{deep}}\Big(h^{(k-1)};\, s_d^{(k)}\Big)\,, \tag{2}$$

where $h^{(0)} = x$ and $k = 1, \ldots, K$. The per-step gains $s_d^{(k)} = \text{softplus}(\alpha^{(k)})$ are learned, with one $\alpha^{(k)}$ per step. Initialising $\alpha^{(k)} = -7$ gives $s_d^{(k)} \approx 9 \times 10^{-4}$, so the recursion begins as a near-identity and the model learns step-wise deviations from x.

Rather than returning $h^{(K)}$ directly, the deep path is a learned weighted combination of all intermediate states. A small router (a linear projection from the current state $h^{(k)}$ and a normalised step index $k/(K-1)$) produces a per-step weight $q_k \in [0, 1]$. Letting $\pi_k = \prod_{j<k}(1 - q_j)$, the deep representation is

$$h_{\text{deep}} \;=\; \sum_{k=1}^{K-1} \pi_k\, q_k\, h^{(k)} \;+\; \pi_K\, h^{(K)}. \tag{3}$$

The router thus lets the model down-weight later loop iterations on a per-token basis.

Wide path (capacity). The wide sublayer Φ_{wide} has the same attention configuration as Φ_{deep} but an enlarged FFN hidden dimension $d_{\text{wide}} > d_{\text{deep}}$, and is applied once:

$$h_{\text{wide}} = \Phi_{\text{wide}}(x;\, s_w), \quad s_w = \text{softplus}(\beta). \tag{4}$$

The scalar β is initialised to -7, matching the deep path. This path adds parameters through the wider FFN but no sequential compute beyond a normal layer.

Per-token gating. A linear projection $W_g \in \mathbb{R}^{d\times 2}$ with bias $b_g \in \mathbb{R}^2$ maps the layer input to logits $(\ell_d, \ell_w) = xW_g + b_g$, giving two independent sigmoid gates $g_d = \sigma(\ell_d)$ and $g_w = \sigma(\ell_w)$, both in $[0, 1]^{B\times T}$. The combined update is

$$y = g_d \odot h_{\text{deep}} \;+\; g_w \odot h_{\text{wide}}. \tag{5}$$

We initialise W_g and b_g to zero, so each token receives $g_d = g_w = 0.5$ at the start of training. The gate is the mechanism by which the model can route tokens that benefit from compute toward the deep path and tokens that benefit from capacity toward the wide path. Because g_d and g_w are independent, their sum can exceed 1. While this is stable at $L = 16$ it can cause the residual stream to compound and diverge at greater depths ($L \geq 32$). In the appendix we show ablations using a gate soft cap that normalizes this sum to 1 (see Table 2).

Single-axis baselines. Disabling one path reduces a block to a *looped* (PURELOOP) layer ($y = h_{\text{deep}}$, K recursions of the standard FFN; Figure 1b) or a *width-scaled* (PUREWIDE) layer ($y = h_{\text{wide}}$, one pass of the enlarged FFN; Figure 1c). These two configurations share the backbone, data, and recipe with the dual-path block and put the entire per-layer FFN FLOP budget on one axis. PUREWIDE has the largest parameter count within a budget (no parameter sharing across recursion); PURELOOP has the smallest (one FFN re-applied K times).

3.4 ROUTING READ-OUTS

Because both paths are evaluated for every token (Section 3.3), the forward pass exposes the raw gates ((g_d, g_w)) *and* the per-path update vectors $\Delta_d = h_{\text{deep}} - x$ and $\Delta_w = h_{\text{wide}} - x$ at every layer and every token.

The raw gate value alone ignores update size. We therefore report the fraction of the residual update that came from the deep path,

$$\rho_d \;=\; \frac{g_d \,\|\Delta_d\|}{g_d \,\|\Delta_d\| + g_w \,\|\Delta_w\|} \;\in\; [0,1], \tag{6}$$

computed per token per layer. $\rho_d = 1$ means the entire update at that position came from the deep path; $\rho_d = 0$ means it came entirely from the wide path; $\rho_d = 0.5$ is balanced. We refer to ρ_d as the *deep share* throughout.

Path alignment. We also record the cosine similarity between the two path deltas,

$$\cos(\Delta_d, \Delta_w) \;=\; \frac{\Delta_d \cdot \Delta_w}{\|\Delta_d\| \,\|\Delta_w\|}. \tag{7}$$

A value near $+1$ means the two paths push the residual in the same direction, i.e. deep and wide path do the same, while a value near 0 means they push in orthogonal directions.

4 Experiments

The backbone is a GPT2-style (Radford et al., 2019) decoder-only transformer with $L = 16$ layers, hidden dimension $d = 768$, and 12 attention heads. Attention uses rotary positional embeddings (Su et al., 2024b) applied to queries and keys, with RMSNorm on queries and keys prior to attention (Dehghani et al., 2023). Feed-forward sublayers are SwiGLU (Shazeer, 2020). These parameters are held fixed across every configuration in the paper, including the single-axis controls; only the FFN widths and the per-layer recursion depth vary.

4.1 Setup

Models and configurations. We train models at two iso-FLOP budgets, specified by the per-layer FFN FLOPs per token ($F_M = 80$M and $F_M = 160$M, corresponding to $\sim$1.28G and $\sim$2.56G total FLOPs per token for a 16-layer model). The detailed procedure for solving for matched FFN widths is given in Appendix B. Within each budget we sweep the dual-path FFN allocation $\alpha \in \{25, 50, 75\}$ (the share of FFN FLOPs spent on the deep path) and the recursion depth $K \in \{2, 3, 4\}$. At fixed budget, larger K and larger α reduce parameter count, since both shift compute toward the shared-parameter deep path. Unique parameter counts across all configurations span from roughly 240M up to 1.4B. All models share pre-RMSNorm (Zhang & Sennrich, 2019), RoPE (Su et al., 2024b), SwiGLU (Shazeer, 2020) and QK-norm (Henry et al., 2020). Exact L, d, h_q, h_{kv}, d_{ffn}, $d_{\text{ffn}}^{\text{wide}}$ per configuration are listed in Appendix C (Table 8).

Data and training. All models are trained on a deduplicated subset of Nemotron-CC (Su et al., 2024a) for 38B tokens with the GPT-2 tokenizer. Sequence length is 4096. We use the `modalities` training framework (Lübbering et al., 2026) using the AdamW optimizer (with peak learning rate 5×10^{-4}, a linear warmup of 184 steps, and a cosine decay schedule down to 5×10^{-5}). Wall-clock training times range from 12.8 to 21.4 hours per model across 64 GPUs. Full optimizer hyperparameters are listed in Appendix A.

Baselines. We compare against two iso-FLOP single-axis controls, trained with the same backbone, data, and recipe:

- PureWide (L layers, single wider FFN of width $d_{\text{ffn}}^{\text{wide}}$, no recursion): our block with the deep path disabled. Spends the entire per-layer FFN FLOP budget on capacity.
- PureLoop (L layers, standard FFN width, recursion depth K, no wide path): our block with the wide path disabled. Spends the entire per-layer FFN FLOP budget on sequential compute.

Evaluation. We report bits-per-byte (BPB) on Paloma C4 and WikiText-103 for language modelling, mean accuracy and BPB on six commonsense tasks (ARC-c, ARC-e, HellaSwag, PIQA, SIQA, WinoGrande), GSM8k accuracy and the OLMo3 base-easy math BPB average over seven sub-tasks,

Table 1: **Main results.** Iso-FLOP comparison at two budgets (Mean ± Std across three seeds). DUAL configurations use `loop=4`; rows vary the FFN FLOP allocation between the deep and wide paths ($a25/a50/a75$ = 25/50/75% on the deep path). The full sweep over loop depths is in Appendix C. Params is shown relative to PUREWIDE within each budget. All rows within a budget are matched in FLOPs.

			Language modelling		Commonsense		Math	
Configuration	Params	FLOPs	C4 ↓	Wiki ↓	Acc. ↑	BPB ↓	Acc. ↑	BPB ↓
PUREWIDE	719M (1.00×)	80M	1.037±0.002	0.946±0.003	0.5030±0.0204	0.9965±0.0083	0.0732±0.0164	0.5305±0.0039
PURELOOP								
Loop=2	398M (0.55×)	80M	1.051±0.002	0.961±0.002	0.4962±0.0093	1.0034±0.0023	0.0762±0.0160	0.5382±0.0059
Loop=3	294M (0.41×)	80M	1.065±0.001	0.976±0.002	0.5038±0.0056	1.0256±0.0045	0.0580±0.0130	0.5500±0.0081
Loop=4	238M (0.33×)	80M	1.078±0.001	0.992±0.002	0.4991±0.0071	1.0334±0.0024	0.0566±0.0122	0.5632±0.0060
DUALPATH *(ours)*								
$\alpha = 25$	606M (0.84×)	80M	1.037±0.002	0.942±0.003	0.5061±0.0134	0.9914±0.0025	0.0659±0.0047	0.5229±0.0024
$\alpha = 50$	483M (0.67×)	80M	**1.036**±0.001	**0.939**±0.002	**0.5192**±0.0081	**0.9816**±0.0004	**0.0982**±0.0073	**0.5200**±0.0047
$\alpha = 75$	360M (0.50×)	80M	1.045±0.001	0.948±0.002	0.5043±0.0232	0.9955±0.0101	0.0905±0.0118	0.5289±0.0057
PUREWIDE	1361M (1.00×)	160M	1.018±0.001	0.921±0.002	0.5376±0.0032	0.9682±0.0089	0.1195±0.0050	0.5066±0.0010
PURELOOP								
Loop=2	719M (0.53×)	160M	1.027±0.000	0.929±0.002	0.5294±0.0033	0.9821±0.0045	0.1201±0.0147	0.5095±0.0010
Loop=3	502M (0.37×)	160M	1.037±0.000	0.940±0.002	0.5215±0.0027	0.9846±0.0050	0.1126±0.0260	0.5141±0.0010
Loop=4	398M (0.29×)	160M	1.045±0.002	0.946±0.002	0.5134±0.0014	0.9973±0.0113	0.0957±0.0138	0.5190±0.0052
DUALPATH *(ours)*								
$\alpha = 25$	1125M (0.83×)	160M	**1.014**±0.001	**0.913**±0.002	**0.5393**±0.0040	**0.9607**±0.0033	0.1182±0.0064	0.5069±0.0024
$\alpha = 50$	880M (0.65×)	160M	1.017±0.001	0.917±0.003	0.5380±0.0048	0.9705±0.0128	0.1250±0.0083	0.5019±0.0076
$\alpha = 75$	644M (0.47×)	160M	1.021±0.001	0.921±0.002	0.5339±0.0010	0.9655±0.0041	**0.1442**±0.0089	**0.5015**±0.0039

and LAMBADA / QASPER for reading and QA. BPB rather than accuracy gives finer-grained signal for these pre-training scales. The aggregate BPB used in Section 4.2 is computed hierarchically: we first form a commonsense BPB (the unweighted mean over the six commonsense task BPBs) and a math BPB (the unweighted mean over the seven OLMo3 base-easy math sub-task BPBs), then take the unweighted mean of four category values: C4, WikiText-103, commonsense, and math. Full per-task numbers are in Appendix E (Tables 5 and 6).

4.2 MAIN RESULTS

A dual-path configuration performs best at both budgets. We focus on $K{=}4$ throughout this section as it is the deepest recursion in our sweep and therefore yields the largest parameter savings at fixed FLOPs. The full sweep over $K \in \{2, 3, 4\}$ is reported in Appendix E. At both FLOP budgets, a dual-path configuration beats both single-axis controls on aggregate BPB. At $F_M = 80$M, $\alpha = 50$ (483M params, $0.67\times$ PUREWIDE) achieves the best aggregate BPB (0.8692 vs. 0.8775 for PUREWIDE and 0.8884 for the best PURELOOP), and is also best on C4, Wiki, commonsense BPB, math BPB, and commonsense accuracy. At $F_M = 160$M, the optimum shifts toward capacity: $\alpha = 25$ (1125M params, $0.83\times$ PUREWIDE) is best on aggregate BPB (0.8487 vs. 0.8535 for PUREWIDE and 0.8619 for the best PURELOOP), and wins C4, Wiki, commonsense BPB, and commonsense accuracy. This shift toward capacity at the higher budget is consistent with the broader picture that looped (parameter-shared) compute is parameter-bottlenecked.

Allocation between deep and wide path. We now investigate how the model performance changes when varying α within the $K{=}4$ models. Capacity-heavy configurations ($\alpha = 25$) are strongest on language modelling and commonsense: at $F_M = 160$M, $\alpha{=}25$ gives the best C4 and WikiText-103 BPB and the best commonsense accuracy and BPB. Compute-leaning configurations are strongest on math: at $F_M = 160$M, GSM8k accuracy and math BPB peak at $\alpha{=}75$ (0.1442 and 0.5015), and at $F_M = 80$M both $\alpha{=}50$ and $\alpha{=}75$ clearly exceed the capacity-heavy $\alpha{=}25$ on GSM8k (0.0982 and 0.0905 vs. 0.0659). This shows that shifting FFN FLOPs toward the wide path adds parameters and helps knowledge-heavy tasks, while shifting them toward the deep path adds sequential compute and helps reasoning-heavy tasks.

Position in the parameter–quality plane. We next plot aggregated bits-per-byte against parameter count for all configurations ($K \in \{2, 3, 4\}$ and $\alpha \in \{25, 50, 75\}$) in Figure 2. For optimal parameter efficiency at a fixed FLOP budget, a model should land in the lower left corner. Within each α,

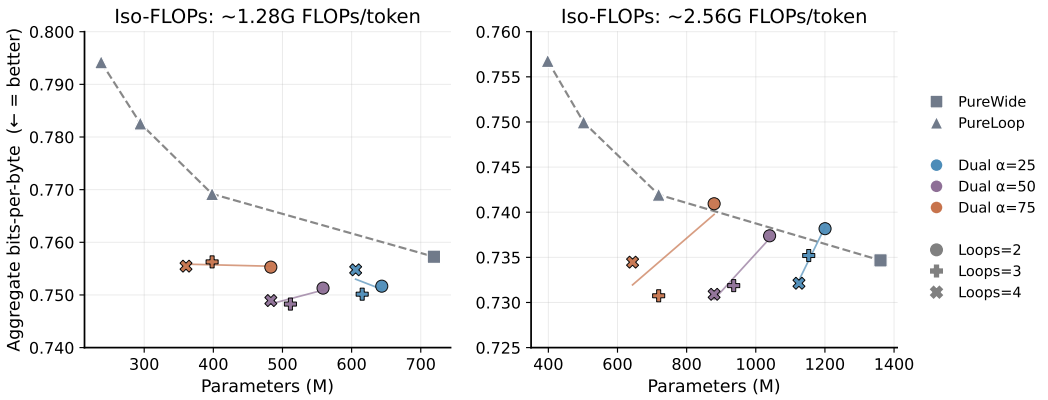


Figure 2: **Parameter scaling sweep of dual-path configurations** compared against PUREWIDE and PURELOOP controls at matched FLOP budgets. Number of Loops for PURELOOP (triangle) is K=4, 3, 2 (from left to right). Note that PUREWIDE is equivalent to PURELOOP with K=1 but without additional routing overhead.

connecting $K \in \{2, 3, 4\}$ traces a short per-α segment moving from right to left (as higher K reduces parameter count at fixed FLOPs). At the lower budget ($F_M = 80$M, ∼1.28G FLOPs), the segments have different slopes, with some increasing performance along the line while others remain flat. However, at the higher budget ($F_M = 160$M, ∼2.56G FLOPs), increasing the loop count generally improves performance across all α values, driving the models further into the optimal lower-left region.

Scaling depth. Table 3 keeps the per-layer budget at $F_M = 80$M and varies the number of layers $L \in \{8, 16, 32\}$, using the gate cap $c = 1$ for all dual-path runs (without it the $L = 32$ run diverges). At $L = 8$ and $L = 16$, the dual-path block beats PUREWIDE on every metric with 0.67–$0.70\times$ the parameters. At $L = 32$ it still leads on math (GSM8K 0.1748 vs. 0.1260, math BPB 0.4848 vs. 0.4925), but PUREWIDE is now slightly ahead on C4, WikiText-103, and commonsense. This suggests that a deep enough stack already supplies most of the sequential compute that language modelling and commonsense need.

4.3 WHERE DOES THE MODEL SPEND ITS BUDGET?

Having established that the dual-path block improves overall performance, we now examine the learned gates to understand how the model allocates its budget. Because the architecture evaluates both paths densely, we can directly read out the routing decisions and residual updates at inference time to uncover the model's underlying preferences for sequential compute (the deep path) or parameter capacity (the wide path). We find that the routing preference varies a) across the layer's position in the stack, b) the identity of the token (e.g. noun vs. number), and c) the task the token is part of (math vs. question answering). Unless otherwise stated, all analyses below use the balanced $\alpha = 50$, $K = 4$ model evaluated on three Paloma sources (WikiText-103, TriviaQA, GSM8K).

Depth in the stack. Figure 3(a) plots the mean deep share per layer for $\alpha \in \{25, 50, 75\}$, along with their average. Two regimes are visible. The middle of the network (L2–L9) is wide-dominated, with deep share between 0.28 and 0.40, while the last two layers (L14–L15) flip to deep-dominated, with the average rising to 0.5–0.6. The pattern is consistent across alpha values, so the shape reflects the layer's role in the stack rather than the specific ratio between compute and capacity.

Panel (b) shows the mean cosine similarity between the update vectors from the deep path (Δ_d) and the wide path (Δ_w) at each layer. A value near $+1$ means the paths produce nearly identical updates (and one is redundant), a value near 0 means they push in orthogonal directions (the paths contribute non-overlapping information). We observe mostly low values across the middle of the network, indicating that the deep and wide paths are processing the same input in genuinely different ways rather than producing scaled copies of one another.

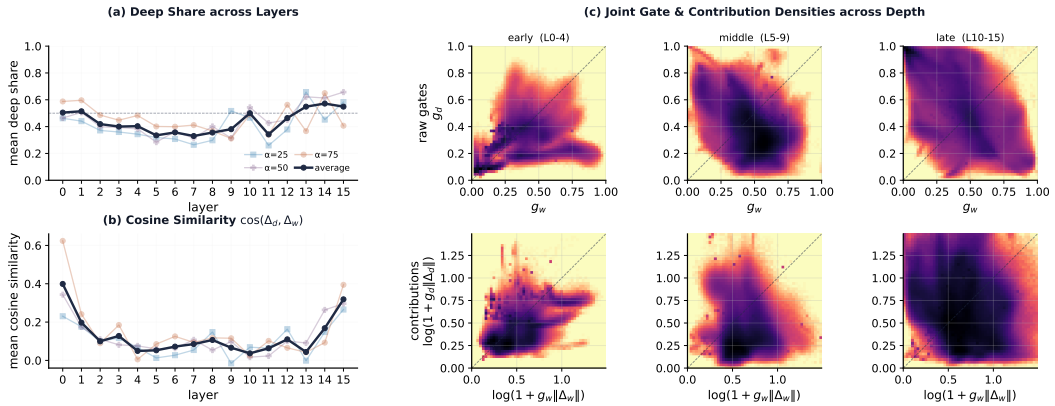


Figure 3: **Routing share, representation alignment, and joint gate densities across depth.** (a) Mean deep share per layer for $\alpha \in \{25, 50, 75\}$ and average. (b) Mean cosine similarity between deep and wide updates (Δ_d, Δ_w). (c) Joint 2D density across depth bands (early L0–4, middle L5–9, late L10–15): raw gates (g_w, g_d) (top) and update contributions weighted by norm (bottom).

Task. The gate responds to the task a token is part of. Figure 4 shows two example sequences side by side: in the GSM8K answer, numbers and arithmetic operators (`15`, `*`, `=`, `3`, `18`) are deep-leaning, and the deep preference strengthens through the arithmetic blocks. In the TriviaQA example, the answer token `Oxy` is the most wide-leaning position in the sequence, consistent with factual recall drawing on the capacity path.

Figure 5 resolves the same examples layer by layer and shows the depth pattern from earlier: later layers prefer the deep path, mid layers prefer the wide path. For a population estimate we align one thousand samples from each task to the `Answer` token and plot the per-layer difference $\rho_d^{\text{GSM8K}} - \rho_d^{\text{TriviaQA}}$ (Figure 5c). Alignment to a shared anchor is needed because the `Answer` token sits at different absolute positions in sequences of varying length. The post-`Answer` positions in GSM8K are markedly more deep-leaning than the corresponding TriviaQA positions in the late layers. At the same depth and the same relative position, a reasoning task routes more deeply than a knowledge task.

Token identity. To investigate in more detail beyond task identity we tag every token with its Universal POS tag (Petrov et al., 2012; Nivre et al., 2016) using spaCy (Honnibal et al., 2020) (`en_core_web_sm`) on the decoded text, with a regex override for arithmetic tokens (digits, operators, =, «, », ####). Each model token inherits the tag of the character span it overlaps most. Figure 6 plots, for every (POS, layer) pair, the mean deep share over all tokens with that tag, restricted to tags with at least 10 occurrences. The ordering is stable across all three datasets and visible in the boxplot of panel (e): SPACE, PUNCT (e.g., `,`, `.`), and SYM (e.g., =, −, «) receive the highest deep share; ADV (e.g., `also`, `as`), PART (e.g., `to`, `'s`), PRON (e.g., `he`, `it`), ADJ (e.g., `many`, `first`), and VERB (e.g., `made`, `used`) receive the lowest; NUM (e.g., `2`, `5`) and NOUN (e.g., `Question`, `Answer`) sit in the middle. Overall, the POS pattern follows an interpretable split: symbolic and structural tokens (SPACE, PUNCT, SYM) route to compute, while lexical content tokens (VERB, ADJ, ADV, PRON, PART), route to capacity. When overriding the gates of arithmetic tokens to bypass the deep path, we observe a degradation in GSM8K loss compared to a size-matched random override, while forcing function words through the deep path selectively degrades WikiText-103 and TriviaQA but not GSM8K (Table 4).

Per-token decisions become more polarised with depth. The analyses so far show *what* the gate prefers but not *how strongly* it commits to those preferences. We therefore plot the joint distribution of the two gates. Figure 3 plots the 2D density of (g_w, g_d) pooled across all tokens, grouped into three layer bands (early L0–4, middle L5–9, late L10–15). The dashed diagonal marks equal routing ($g_d = g_w$): mass on the diagonal corresponds to balanced mixtures, while mass off the diagonal corresponds to tokens that commit primarily to one path. The per-layer breakdown is shown in Appendix Figure 7. We observe that the density moves off the diagonal as depth increases. Early

layers (L0–4) concentrate in a narrow band along the positive diagonal near the origin, indicating small and roughly balanced gates for most tokens. Late layers (L10–15) push density onto the boundaries with a cluster near $g_d \to 1$ (tokens routed almost entirely through the deep path) and a cluster near $g_w \to 1$ (tokens routed almost entirely through the wide path). Row B plots the same density after re-weighting each axis by the actual update magnitude ($\log(1 + g_w\|\Delta_w\|)$ vs. $\log(1 + g_d\|\Delta_d\|)$), which accounts for the fact that a high gate value contributes little if its path's update is small. We observe that the gate is not just choosing different mixtures for different tokens but it also commits more strongly to one path or the other as the residual stream moves through the network.

5 Conclusion

Standard transformer layers conflate two distinct scaling axes: compute (sequential operations on a hidden state) and capacity (parameters available at a single step). The dual-path block separates them, using a recursive deep sublayer and a parallel wider sublayer combined by a dense learned gate. At two iso-FLOP budgets, the best dual-path configuration improves on both single-axis controls on aggregate language modelling, commonsense, and math metrics while using fewer parameters than the width-scaled baseline. The ratio between the wide and deep path α provides a predictable trade-off, with capacity-heavy settings favoring knowledge tasks and compute-heavy settings favoring reasoning.

Our results show that this parallel formulation can outperform iso-FLOP single-axis baselines on aggregate language modelling, commonsense, and math metrics at two FLOP budgets. Because both paths are evaluated at every token, the routing decisions are a direct read-out of how the trained model chose to spend its budget, not a sampling artifact. This learned allocation is interpretable. It varies systematically with layer depth, with wide being preferred in the middle of the stack, and deep dominated in the last two layers. Moreover, function words and lexical content trend wide while punctuation, symbols, and arithmetic tokens trend deep.

The dual-path block opens several directions we find promising. First, the per-α trajectories in Figure 2 steepen from the lower to the higher FLOP budget. Whether this trend continues, plateaus, or reverses at billion-parameter budgets and across different depth/width ratios is the most direct extension.

Second, as noted above, the dual-path gate and MoE routing target orthogonal axes and are architecturally compositional: loops increase FLOPs while keeping parameters the same, while MoEs increase the parameters while keeping FLOPs fixed (with fixed top-k). In our model a MoE layer could occupy the wide path, with the outer gate deciding between looped compute and routed capacity.

These directions suggest that separating compute and capacity within a layer can be an extension that combines with other scaling ideas, while providing an interpretable signal as a free byproduct.

6 Limitations

Our experiments cover two per-layer FLOP budgets and models up to 1.36B parameters trained on 38B tokens. The dual-path advantage holds at both budgets and across three backbone depths (Table 3), and testing whether it persists at larger scales is an open question. The dual-path block does not provide conditional computation: both paths and all K recursion steps run for every token, so the gates re-allocate contribution but do not reduce per-token FLOPs (Section 3.3). Sparsifying the gate to obtain per-token savings is left for future work.

The model does not benefit from running more loops at inference than it was trained with. This is expected for fixed training depth and prior work obtains extrapolation by randomising the recursion depth during training (Geiping et al., 2025), which is compatible with our block. Lastly, with independent gates, uncapped training diverges at $L \geq 32$. Capping $g_d + g_w$ at 1 removes the divergence and improves results at $L = 16$ (Table 2).

AI USE STATEMENT

In this manuscript, we used Claude for writing support, i.e. copy-editing and phrasing and for ideation during the intial stages of the project. We also used it for visualization of results and for evaluating weaknesses in the manuscript. We have not used it to generate datasets or for designing the experiments. All AI-assisted text and code were reviewed and verified for correctness. We take full responsibility for the final content of this work.

REFERENCES

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Yilong Chen, Naibin Gu, Junyuan Shang, Zhenyu Zhang, Yuchen Feng, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, et al. Mixture of universal experts: Scaling virtual width via depth-width transformation. *arXiv preprint arXiv:2603.04971*, 2026.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International Conference on Machine Learning*, pp. 7480–7512. PMLR, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Markus Frey, Behzad Shomali, Ali Hamza Bashir, David Berghaus, Joachim Koehler, and Mehdi Ali. Adaptive loops and memory in transformers: Think harder or know more? In *Latent & Implicit Thinking Workshop @ ICLR*, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4246–4253, 2020.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy: Industrial-strength natural language processing in python, 2020.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. *arXiv preprint arXiv:2602.11451*, 2026.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

Max Lübbering, Timm Ruland, Richard Rutmann, Felix Stollenwerk, David Fitzek, Michael Fromm, Alexander Weber, Rafet Sifa, Nicolas Flores-Herr, Joachim Köhler, et al. Modalities, a pytorch-native framework for large-scale LLM training and research. *arXiv preprint*, 2026.

Ying Nie, Kai Han, Hongguang Li, Hang Zhou, Tianyu Guo, Enhua Wu, Xinghao Chen, and Yunhe Wang. Versatileffn: Achieving parameter efficiency in llms via adaptive wide-and-deep reuse, 2026. URL https://arxiv.org/abs/2512.14531.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pp. 1659–1666, 2016.

Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pp. 2089–2096, 2012.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.

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.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. 2025.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Behzad Shomali, Markus Frey, David Berghaus, Joachim Koehler, and Mehdi Ali. Loopmtp: A looped transformer guided by latent multi-token prediction. *arXiv preprint arXiv:2608.03624*, 2026.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2024a.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024b.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Herve Jegou, and Armand Joulin. Augmenting self-attention with persistent memory. *arXiv preprint arXiv:1907.01470*, 2019.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

GSM8K MATHEMATICAL REASONING EXAMPLE

```
Question: Ashley's pizza delivery costs $15. What is the total amount
that Ashley should give the delivery man if she wants to give a tip that
is equal to 1/5 of the amount she ordered?
Answer: The tip that Ashley wants to give amounts to $15 x 1/5 =
$<<15*1/5=3>>3.
Hence, she will give a total of $15 + $3 = $<<15+3=18>>18 to the
delivery man.
#### 18
```

TRIVIAQA FACTUAL KNOWLEDGE EXAMPLE

```
Question: What is the second most common gas in the atmosphere?
Answer: Oxygen
```

wide-leaning  deep-leaning

Figure 4: Token-level deep share for GSM8K and TriviaQA. Blue denotes wide-leaning (prefers the capacity path) while red denotes deep-leaning, meaning it prefers the compute (looped) path.

APPENDIX

A TRAINING DETAILS

All models are trained using the `modalities` framework. We use the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and weight decay of 0.3 (excluding embedding and RMSNorm layers). The learning rate is warmed up linearly from 5×10^{-6} to a peak of 5×10^{-4} over 184 steps, and then decayed to 5×10^{-5} using a cosine annealing schedule. Wall-clock training times range from 12.8 to 21.4 hours per model (average of 16.0 hours) on the cluster hardware. Total pretraining tokens are 38B.

B FLOP-MATCHING PROTOCOL

The FLOP budget F_M is the per-token, per-layer FFN compute. For each configuration we solve for the largest legal d_{ffn} (and, for dual-path, $d_{\text{ffn}}^{\text{wide}}$) whose induced h_{eff} keeps the per-layer FFN cost at or below F_M, also accounting for the router on the dual-path block. Within each budget the controls and the dual configurations agree on total per-token FFN FLOPs to within 0.5%; the same backbone ($L = 16$, $d = 768$, $h_q = h_{kv} = 12$) is held fixed across all configurations, so attention compute is identical.

This appendix gives the exact accounting used to match FLOPs across configurations. All FLOPs are reported per token and per layer.

Per-sublayer FLOPs. Let d be the model dimension, $n_{\text{rep}} = h_q/h_{kv}$ the repeat factor (1 in all our runs), and d_{ffn} the *configured* FFN hidden width. The SwiGLU *effective* hidden width used by the model is

$$h_{\text{eff}}(d_{\text{ffn}}) \;=\; 64 \cdot \left\lceil \tfrac{1}{64} \lfloor 2d_{\text{ffn}}/3 \rfloor \right\rceil, \tag{8}$$

i.e. the LLaMA-style $2/3$ scaling rounded up to a multiple of 64. The per-token FLOPs of one sublayer are

$$\text{FLOP}_{\text{attn}} = 4d^2 + 4d^2/n_{\text{rep}}, \tag{9}$$

$$\text{FLOP}_{\text{ffn}}(d_{\text{ffn}}) = 6d \cdot h_{\text{eff}}(d_{\text{ffn}}). \tag{10}$$

The dual-path two-gate router contributes $\text{FLOP}_{\text{gate}} = 4d$ FLOPs per token (linear $d \to 2$).

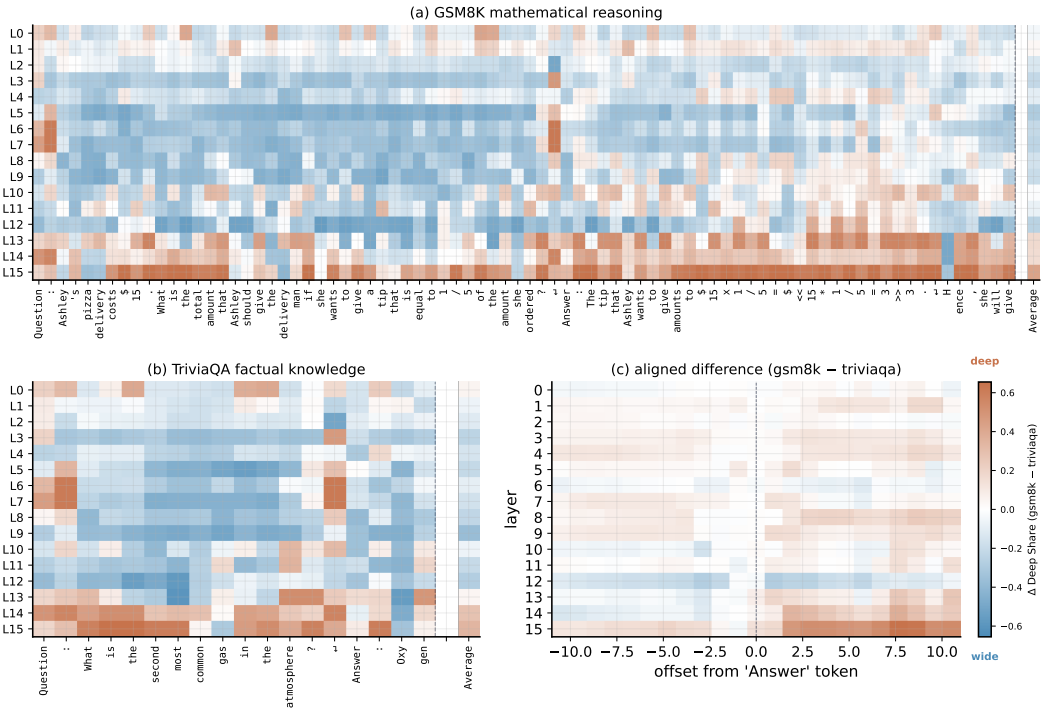


Figure 5: **Step-by-step token-level routing grid and task alignment.** Panel (a) shows the layer-by-token heatmap of the deep share for a sequence from GSM8K (mathematical reasoning). Panel (b) shows the heatmap for a sequence from TriviaQA (factual knowledge). Panel (c) shows the aligned difference in preference (gsm8k − triviaqa) around the anchor token "Answer", averaged over one thousand sequences per dataset.

Per-layer FLOP budgets. For the three layer types, the per-layer FFN-side FLOP count F_M (which we hold fixed at 80M or 160M) is

$$F_M^{\text{PUREWIDE}} = \text{FLOP}_{\text{attn}} + \text{FLOP}_{\text{ffn}}(d_{\text{ffn}}^{\text{wide}}), \tag{11}$$

$$F_M^{\text{PURELOOP}} = K\big(\text{FLOP}_{\text{attn}} + \text{FLOP}_{\text{ffn}}(d_{\text{ffn}})\big), \tag{12}$$

$$\begin{aligned} F_M^{\text{DUAL}} = {} & K\big(\text{FLOP}_{\text{attn}} + \text{FLOP}_{\text{ffn}}(d_{\text{ffn}})\big) \\ & + \text{FLOP}_{\text{attn}} + \text{FLOP}_{\text{ffn}}(d_{\text{ffn}}^{\text{wide}}) \\ & + \text{FLOP}_{\text{gate}}. \end{aligned} \tag{13}$$

The dual-path layer therefore pays an extra attention pass and a small router term relative to a single-axis layer at the same FFN FLOPs. Note that we hold all FLOPs fixed at 80M or 160M per layer and per token by adjusting the size of the FFN.

Solving for FFN widths. Given F_M, K, and (for dual) a deep-FLOP fraction $\alpha \in (0, 1)$, we set $\text{FLOP}_{\text{ffn}}(d_{\text{ffn}}) = (\alpha(F_M - \text{FLOP}_{\text{gate}})/K) - \text{FLOP}_{\text{attn}}$ and analogously for the wide width with $(1 - \alpha)$. We then invert Eq. 8 for the *largest* d_{ffn} (rounded down to a multiple of 64) whose h_{eff} keeps the equation at or below the target – the "floor" rounding mode. The exception is PUREWIDE, where we round up ("ceil"); this keeps the largest single-axis capacity baseline honest by spending the entire budget and has strictly more FLOPs than the dual-path baselines. The residual mismatch is $< 2\%$ of F_M in every configuration.

C MODEL CONFIGURATIONS

All configurations share the same backbone: $L = 16$ layers, $d = 768$, $h_q = h_{kv} = 12$ (GQA repeat factor 1), sequence length 4096, vocabulary size 50,304, weight-tied input/output embeddings, and

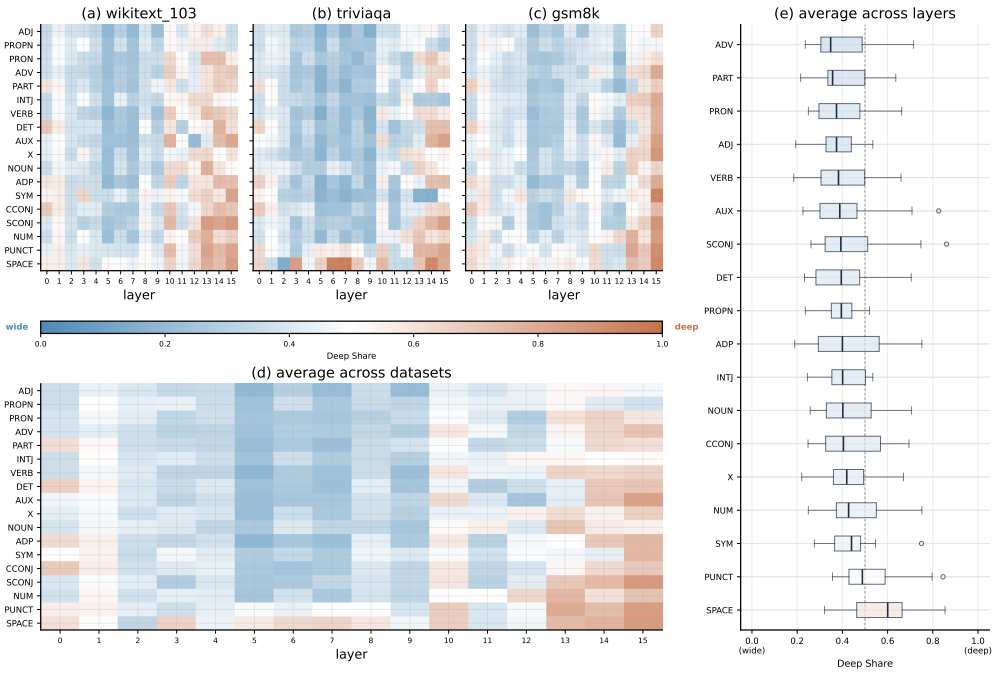


Figure 6: **Parts-of-speech (POS) routing characteristics and commitment.** Panels (a, b, c) show heatmaps of the mean deep share per universal POS tag across layers for `wikitext_103`, `triviaqa`, and `gsm8k`, respectively, with tags sorted by the overall mean deep share. Panel (d) plots the average heatmap across the three datasets. Panel (e) shows boxplots of the deep share distribution per POS tag across all layers and datasets, sorted by their median preference (most deep-leaning at the bottom).

Table 2: **Gate soft cap.** The two-gate residual gain $g_d + g_w$ lies in $(0, 2)$ and is free to sit above 1, so it compounds with depth. Capping the *sum* at $c = 1$ (dividing both gates by $\max(1, (g_d + g_w)/c)$, which preserves the routing ratio $g_d : g_w$ exactly and leaves tokens below the cap untouched) removes the divergence seen at $L \geq 32$ and does not cost accuracy at $L = 16$. All rows are $\alpha = 50$, `loop=4`, FLOP-matched to PUREWIDE. Single runs.

			Language modelling		Commonsense		Math	
Configuration	Params	FLOPs	C4 ↓	Wiki ↓	Acc. ↑	BPB ↓	Acc. ↑	BPB ↓
PUREWIDE	719M	80M	1.037	0.946	0.5203	0.9920	0.0791	0.5261
DUALPATH $\alpha = 50$	483M	80M	**1.036**	0.939	**0.5256**	0.9820	0.0898	0.5198
+ gate cap $c = 1$	483M	80M	**1.036**	**0.933**	0.5233	**0.9740**	**0.1348**	**0.5116**
PUREWIDE	1361M	160M	1.018	0.922	0.5384	0.9636	0.1250	0.5077
DUALPATH $\alpha = 50$	880M	160M	1.017	0.920	0.5326	0.9704	0.1328	0.4959
+ gate cap $c = 1$	880M	160M	**1.010**	**0.905**	**0.5430**	**0.9529**	**0.1377**	**0.4894**

the SwiGLU effective-hidden rule of Appendix B (multiple of 64). Table 8 lists, for every model in this paper, the configured deep FFN width d_{ffn}, wide FFN width $d_{\text{ffn}}^{\text{wide}}$, recursion depth K, total parameter count, and wall-clock training time.

D INFERENCE-TIME ABLATIONS

We probe the trained dual-path model ($F_M = 80$M, $\alpha = 50$, $K = 4$) at inference time, overriding either the loop count or the gate. Table 7 reports per-token cross-entropy on three Paloma sources.

Table 3: **Depth sweep at the F80M budget.** F_M is a *per-layer* budget, so the block is identical at every depth and only L changes; total FLOPs/token and parameters scale linearly with L. DUALPATH uses the gate cap $c = 1$, `loop=4`. Without the cap the $L = 32$ run diverges to `NaN`, so to make this sweep comparable we re-ran L=8 & 16 using the cap $c = 1$.

				Language modelling		Commonsense		Math	
L	Configuration	Params	GFLOPs	C4 ↓	Wiki ↓	Acc. ↑	BPB ↓	Acc. ↑	BPB ↓
8	PUREWIDE	398M	0.64	1.078	0.997	0.4893	1.0469	0.0322	0.5724
	DUALPATH $\alpha = 50$	280M	0.63	**1.072**	**0.983**	**0.4992**	**1.0252**	**0.0508**	**0.5611**
16	PUREWIDE	719M	1.28	1.037	0.946	0.5203	0.9920	0.0791	0.5261
	DUALPATH $\alpha = 50$	483M	1.26	**1.036**	**0.933**	**0.5233**	**0.9740**	**0.1348**	**0.5116**
32	PUREWIDE	1361M	2.57	**1.008**	**0.903**	**0.5438**	**0.9488**	0.1260	0.4925
	DUALPATH $\alpha = 50$	889M	2.52	1.014	0.905	0.5412	0.9517	**0.1748**	**0.4848**

The loop budget matters, and the schedule does not extrapolate. Forcing fewer loop iterations at inference monotonically degrades loss (e.g. GSM8K loss rises from 2.148 at the trained $K{=}4$ to 3.592 at $K{=}1$). Returns diminish past the trained value, i.e. the loop dynamics learned at training do not extrapolate past the training budget.

Both paths are needed. Disabling either gate ($g_d{=}1, g_w{=}0$ or $g_d{=}0, g_w{=}1$) costs 3.6–5.6 nats across the three sources – larger than the gap between an early-training and a fully-trained model. A uniform $0.5{:}0.5$ split still loses ~ 2.6 nats. Opening both gates fully ($g_d{=}1, g_w{=}1$) is the worst override: the model has trained with bounded gate magnitudes and the residual update is far out of its training distribution.

The per-token decision carries information. The shuffled-gates row keeps the marginal distribution of (g_d, g_w) but randomises which token gets which assignment within each sequence. Loss rises by 0.51–0.94 nats, indicating that the gate's per-token decisions, not just its average behaviour are load-bearing.

Routing choices are functionally necessary for specific token types. The global overrides above show that the gate matters overall. To test if individual tokens rely on their preferred paths, we run gate overrides across all checkpoints. We push arithmetic tokens (NUM and SYM) through the wide path instead of the deep one, and function words (ADJ, ADV, PART, PRON, VERB) through the deep path instead of the wide one. Because any forced routing hurts a trained model, we compare each intervention against a control group where an identical number of random tokens are overridden. Table 4 reveals that forcing arithmetic tokens wide causes a consistent loss spike on GSM8K ($+0.37$ nats over control, $p < 0.001$) and WikiText-103, but leaves TriviaQA unaffected. Conversely, forcing function words deep hurts language modeling on WikiText-103 and TriviaQA ($p < 0.05$) while sparing math.

E EVALUATION DETAILS

We evaluate language modelling on Paloma C4 and WikiText-103 (bits-per-byte). Commonsense is the mean over ARC-c, ARC-e, HellaSwag, PIQA, SIQA, and WinoGrande. Math is GSM8k accuracy and the OLMo3 base-easy math BPB average over Algebra, Counting, Geometry, Intermediate Algebra, Number Theory, Pre-algebra, and Pre-calculus. The aggregate BPB reported in Section 4.2 is the unweighted mean of four category values (C4, WikiText-103, the six-task commonsense BPB mean, and the seven-sub-task math BPB mean), so each category contributes equally.

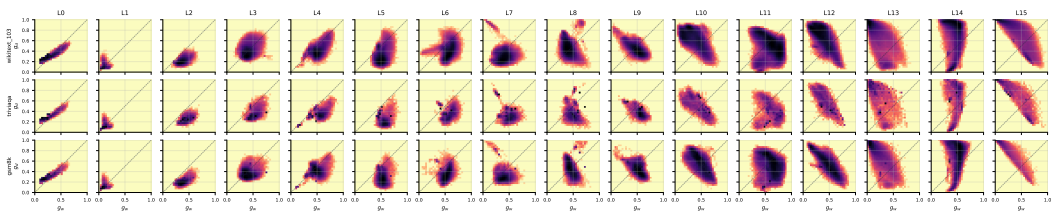


Figure 7: **Layer-wise joint density of raw gates** (g_w, g_d) across all layers ($L0$ to $L15$) and evaluation sources. The rows correspond to the three Paloma evaluation datasets (`wikitext_103`, `triviaqa`, and `gsm8k`), while columns correspond to layers index 0 to 15. The diagonal dashed line in each subplot represents equal routing preference ($g_d = g_w$). Lighter regions represent higher token concentration. The model has $L = 16$ layers, $K = 4$ loops, $d_{\text{model}} = 768$, deep FFN hidden width $= 4864$, wide FFN hidden width $= 24448$, dual $\alpha = 50$.

Intervention	**GSM8K**	**TriviaQA**	**Wiki-103**
Loss increase over learned gates (Δ CE, mean ± sd)			
Arith. forced wide	2.30 (±0.35)	0.14 (±0.02)	0.67 (±0.13)
Random ctrl (wide)	1.93 (±0.31)	0.14 (±0.02)	0.41 (±0.07)
Func. words forced deep	0.75 (±0.22)	0.71 (±0.19)	1.08 (±0.24)
Random ctrl (deep)	0.73 (±0.14)	0.65 (±0.13)	0.94 (±0.11)
Class-specific excess over random control			
Arith. forced wide	$+0.37^{***}$	$+0.00$	$+0.26^{***}$
Func. words forced deep	$+0.02$	$+0.06^{*}$	$+0.14^{*}$

Table 4: **Token-class gate overrides.** Per-token cross-entropy increase relative to the learned gates, aggregated over 18 trained checkpoints spanning both FLOP budgets, $\alpha \in \{25, 50, 75\}$, and $K \in \{2, 3, 4\}$. Arithmetic tokens (NUM, SYM) are forced wide (g_d=0, g_w=1); function words (ADJ, ADV, PART, PRON, VERB) are forced deep (g_d=1, g_w=0); each is compared against a size-matched random-token control under the same override. The bottom block reports the paired per-checkpoint excess of the class-targeted intervention over its control (Wilcoxon signed-rank, n=18; $^{*}p < 0.05$, $^{***}p < 0.001$).

Table 5: Full results at the **F80M** FLOP budget. ℓ denotes the loop count. Commonsense Acc. / BPB are means over ARC-c, ARC-e, HellaSwag, PIQA, SIQA, WinoGrande. The OLMo3 easy math avg is the OLMo3 base-easy math BPB average. Best value in each row is **bold** (highest for accuracy ↑, lowest for BPB ↓).

Metric	PureWide	PureLoop $\ell 2$	PureLoop $\ell 3$	PureLoop $\ell 4$	Dual-a25 $\ell 2$	Dual-a50 $\ell 2$	Dual-a75 $\ell 2$	Dual-a25 $\ell 3$	Dual-a50 $\ell 3$	Dual-a75 $\ell 3$	Dual-a25 $\ell 4$	Dual-a50 $\ell 4$	Dual-a75 $\ell 4$
Params	*719M*	*398M*	*294M*	*238M*	*644M*	*559M*	*483M*	*615M*	*511M*	*398M*	*606M*	*483M*	*360M*
Commonsense (avg)													
Acc. ↑	0.5203	0.5036	0.5067	0.5031	**0.5283**	0.5265	0.5218	0.5239	0.5269	0.5192	0.5163	0.5256	0.5249
BPB ↓	0.9920	1.0028	1.0236	1.0352	0.9888	0.9826	0.9866	0.9827	**0.9808**	0.9870	0.9902	0.9820	0.9892
Commonsense (per task, Acc. ↑ / BPB ↓)													
ARC-c Acc.	0.4053	0.3779	0.3770	0.3809	0.4307	0.4111	**0.4346**	0.4336	0.4004	0.3994	0.4014	0.4170	0.4072
ARC-c BPB	0.7967	0.8253	0.8546	0.8651	0.8001	0.7958	**0.7954**	0.7966	0.8016	0.8188	0.7974	0.7983	0.8090
ARC-e Acc.	0.7158	0.6953	0.6973	0.6875	0.7148	**0.7305**	0.7100	0.6973	0.7178	0.7119	0.7148	0.6982	0.7266
ARC-e BPB	0.5915	0.6134	0.6308	0.6801	0.5912	**0.5779**	0.5855	0.5907	0.5877	0.5820	0.5904	0.5912	0.6034
HellaSwag Acc.	0.3906	0.3730	0.3760	0.3643	0.4014	0.4131	0.4043	0.4121	**0.4150**	0.3965	0.3955	**0.4150**	0.3955
HellaSwag BPB	0.8947	0.8997	0.9156	0.9282	0.8872	0.8865	0.8928	**0.8847**	0.8888	0.8973	0.8940	0.8911	0.8946
PIQA Acc.	0.6475	0.6348	0.6436	0.6338	0.6533	0.6436	0.6455	0.6348	**0.6582**	0.6357	0.6289	0.6455	0.6367
PIQA BPB	1.1511	1.1816	1.1962	1.2069	**1.1441**	1.1524	1.1577	1.1504	1.1563	1.1646	1.1540	1.1627	1.1567
SIQA Acc.	0.4453	0.4346	0.4297	0.4326	**0.4492**	0.4346	0.4346	0.4385	0.4443	0.4453	0.4424	0.4424	**0.4492**
SIQA BPB	1.1863	1.2167	1.2252	1.2343	1.1983	1.2090	1.2101	1.1922	**1.1625**	1.1813	1.2147	1.1768	1.1975
WinoGrande Acc.	0.5176	0.5059	0.5166	0.5195	0.5205	0.5264	0.5020	0.5273	0.5254	0.5264	0.5146	**0.5352**	0.5342
WinoGrande BPB	1.3316	1.2803	1.3191	1.2964	1.3121	1.2741	1.2785	1.2819	1.2881	1.2782	1.2910	**1.2716**	1.2743
Math													
GSM8k Acc. ↑	0.0791	0.0723	0.0723	0.0527	0.0879	0.0801	**0.1006**	0.0879	**0.1006**	0.0820	0.0664	0.0898	0.0918
Algebra BPB ↓	0.5201	0.5306	0.5385	0.5468	0.5173	0.5162	0.5198	0.5178	**0.5115**	0.5186	0.5215	0.5173	0.5141
Counting BPB ↓	0.5240	0.5343	0.5479	0.5557	**0.5228**	0.5244	0.5289	0.5259	0.5232	0.5294	0.5289	0.5230	0.5252
Geometry BPB ↓	0.5622	0.5736	0.5860	0.6004	0.5571	0.5579	0.5667	0.5596	0.5568	0.5634	0.5579	**0.5567**	0.5605
Int. Algebra BPB ↓	0.5402	0.5538	0.5592	0.5707	0.5332	0.5281	0.5409	0.5358	0.5317	0.5377	0.5372	**0.5262**	0.5334
Number Theory BPB ↓	0.5940	0.6057	0.6137	0.6218	0.5926	0.5902	0.5937	0.5924	0.5892	0.5981	0.5950	**0.5870**	0.5885
Pre-algebra BPB ↓	0.5069	0.5155	0.5237	0.5322	0.5061	0.5033	0.5087	0.5051	**0.5005**	0.5078	0.5095	0.5036	0.5048
Pre-calculus BPB ↓	0.4357	0.4470	0.4488	0.4670	0.4306	0.4252	0.4391	0.4316	0.4315	0.4335	0.4272	**0.4247**	0.4325
Easy math avg BPB ↓	0.5261	0.5372	0.5454	0.5564	0.5228	0.5208	0.5282	0.5240	0.5206	0.5269	0.5253	**0.5198**	0.5227
Reading / QA													
LAMBADA Acc. ↑	0.2861	0.2803	0.2744	0.2793	0.2949	0.2725	0.2939	0.2832	**0.3135**	0.2881	0.2988	0.3037	0.3076
LAMBADA BPB ↓	0.9492	0.9769	0.9887	1.0032	0.9161	0.9454	0.9342	0.9167	**0.9146**	0.9462	0.9341	0.9154	0.9473
QASPER Acc. ↑	0.5987	0.6740	0.6771	0.6771	0.6646	0.5361	0.6677	0.6301	0.5862	**0.6803**	0.6301	0.4765	0.6771
QASPER BPB ↓	0.3063	0.3098	0.3150	0.3233	0.3012	0.3030	0.3010	0.3035	0.3027	0.3020	**0.2992**	0.3108	0.3081
Paloma (BPB ↓)													
C4	1.0370	1.0513	1.0652	1.0777	**1.0311**	1.0374	1.0395	1.0331	1.0358	1.0472	1.0374	1.0362	1.0451
WikiText-103	0.9461	0.9608	0.9757	0.9918	0.9373	0.9453	0.9477	**0.9344**	0.9382	0.9508	0.9420	0.9393	0.9480

Table 6: Full results at the **F160M** FLOP budget. ℓ denotes the loop count. Commonsense Acc. / BPB are means over ARC-c, ARC-e, HellaSwag, PIQA, SIQA, WinoGrande. The OLMo3 easy math avg is the OLMo3 base-easy math BPB average. Best value in each row is **bold** (highest for accuracy ↑, lowest for BPB ↓).

Metric	PureWide	PureLoop ℓ2	PureLoop ℓ3	PureLoop ℓ4	Dual-a25 ℓ2	Dual-a50 ℓ2	Dual-a75 ℓ2	Dual-a25 ℓ3	Dual-a50 ℓ3	Dual-a75 ℓ3	Dual-a25 ℓ4	Dual-a50 ℓ4	Dual-a75 ℓ4
Params	*1361M*	*719M*	*502M*	*398M*	*1200M*	*1040M*	*880M*	*1153M*	*936M*	*719M*	*1125M*	*880M*	*644M*
Commonsense (avg)													
Acc. ↑	0.5384	0.5303	0.5244	0.5137	0.5374	0.5358	0.5202	**0.5493**	0.5360	0.5384	0.5420	0.5326	0.5335
BPB ↓	0.9636	0.9814	0.9894	1.0010	0.9755	0.9608	0.9744	0.9655	0.9572	**0.9572**	0.9610	0.9704	0.9700
Commonsense (per task, Acc. ↑ / BPB ↓)													
ARC-c Acc.	0.4277	0.4170	0.4053	0.3896	0.4365	0.4160	0.4150	0.4346	0.4326	0.4316	0.4482	**0.4512**	0.4375
ARC-c BPB	0.7715	0.7880	0.8107	0.8178	0.7758	0.7674	0.7746	0.7839	**0.7652**	0.7704	0.7800	0.7673	0.7730
ARC-e Acc.	**0.7520**	0.7266	0.7109	0.7051	0.7207	0.7275	0.7129	0.7461	0.7305	0.7314	0.7373	0.7324	0.7363
ARC-e BPB	**0.5558**	0.5717	0.5961	0.6046	0.5634	0.5579	0.5682	0.5610	0.5633	0.5606	0.5624	0.5775	0.5668
HellaSwag Acc.	0.4229	0.4248	0.4004	0.4043	**0.4414**	0.4316	0.4111	0.4404	0.4258	0.4219	0.4277	0.4209	0.4170
HellaSwag BPB	0.8767	0.8809	0.8935	0.8929	0.8670	0.8723	0.8827	0.8696	0.8705	0.8738	**0.8661**	0.8796	0.8759
PIQA Acc.	0.6689	0.6445	0.6455	0.6367	**0.6777**	0.6592	0.6436	0.6660	0.6543	0.6611	0.6494	0.6602	0.6562
PIQA BPB	1.1263	1.1494	1.1660	1.1705	**1.1145**	1.1259	1.1336	1.1181	1.1185	1.1304	1.1367	1.1354	1.1349
SIQA Acc.	0.4541	0.4463	0.4414	0.4395	0.4395	0.4541	0.4463	**0.4658**	0.4434	0.4492	0.4502	0.4277	0.4443
SIQA BPB	1.1906	1.2077	1.2048	1.2272	**1.1524**	1.1695	1.1809	1.1780	1.1751	1.1639	1.1638	1.1794	1.1893
WinoGrande Acc.	0.5049	0.5225	**0.5430**	0.5068	0.5088	0.5264	0.4922	**0.5430**	0.5293	0.5352	0.5391	0.5029	0.5098
WinoGrande BPB	1.2609	1.2908	1.2650	1.2932	1.3800	1.2719	1.3063	1.2826	1.2509	**1.2439**	1.2572	1.2830	1.2802
Math													
GSM8k Acc. ↑	0.1250	0.1045	0.1191	0.0938	0.0996	0.1084	0.1055	0.1084	0.1074	0.1094	0.1250	0.1328	**0.1406**
Algebra BPB ↓	0.5038	0.5013	0.5101	0.5114	0.5074	0.5081	0.5049	0.5043	0.5007	0.5041	0.5018	**0.4920**	0.4976
Counting BPB ↓	0.5115	0.5112	0.5200	0.5242	0.5127	0.5147	0.5157	0.5135	0.5098	0.5129	0.5129	**0.5031**	0.5083
Geometry BPB ↓	0.5419	0.5426	0.5466	0.5585	0.5397	0.5457	0.5478	0.5383	0.5389	0.5440	0.5369	**0.5282**	0.5404
Int. Algebra BPB ↓	0.5137	0.5201	0.5250	0.5282	0.5208	0.5215	0.5243	0.5175	0.5186	0.5191	0.5154	**0.5003**	0.5145
Number Theory BPB ↓	0.5751	0.5783	0.5840	0.5872	0.5804	0.5840	0.5784	0.5804	0.5769	0.5758	0.5758	**0.5632**	0.5699
Pre-algebra BPB ↓	0.4923	0.4924	0.4974	0.5007	0.4942	0.4964	0.4956	0.4920	0.4913	0.4928	0.4917	**0.4831**	0.4864
Pre-calculus BPB ↓	0.4154	0.4177	0.4221	0.4289	0.4210	0.4231	0.4239	0.4135	0.4137	0.4165	0.4117	**0.4010**	0.4155
Easy math avg BPB ↓	0.5077	0.5091	0.5150	0.5199	0.5109	0.5134	0.5130	0.5085	0.5071	0.5093	0.5066	**0.4959**	0.5047
Reading / QA													
LAMBADA Acc. ↑	0.3086	0.3135	0.2920	0.2939	0.3115	0.2979	0.2822	0.2959	0.3115	0.3223	0.3105	**0.3271**	0.3203
LAMBADA BPB ↓	0.9169	0.9038	0.9408	0.9359	0.8811	0.8997	0.9184	0.9031	0.8985	0.8780	0.8854	**0.8646**	0.8945
QASPER Acc. ↑	0.5768	0.6301	0.6583	0.5987	0.6583	0.6207	0.6803	**0.6865**	0.5799	0.6301	0.4514	0.5925	0.6050
QASPER BPB ↓	0.2964	0.2990	**0.2916**	0.2941	0.3120	0.3405	0.2982	0.3112	0.3215	0.2963	0.3247	0.3309	0.2934
Paloma (BPB ↓)													
C4	1.0183	1.0270	1.0367	1.0446	1.0130	1.0163	1.0203	1.0138	1.0138	1.0188	**1.0129**	1.0166	1.0212
WikiText-103	0.9224	0.9312	0.9388	0.9453	0.9137	0.9208	0.9220	0.9177	0.9147	0.9214	**0.9108**	0.9201	0.9241

Ablation	**GSM8K**	**TriviaQA**	**WikiText-103**
Learned router (baseline)	2.148	4.129	3.026
Compute (force K loops; trained with K=4)			
K=1	3.592 (+1.44)	5.777 (+1.65)	4.846 (+1.82)
K=2	2.665 (+0.52)	4.694 (+0.57)	3.842 (+0.82)
K=3	2.221 (+0.07)	4.174 (+0.05)	3.178 (+0.15)
K=4 (*trained*)	2.148 (+0.00)	4.129 (+0.00)	3.026 (+0.00)
K=5	2.206 (+0.06)	4.187 (+0.06)	3.046 (+0.02)
K=6	2.271 (+0.12)	4.239 (+0.11)	3.074 (+0.05)
K=7	2.323 (+0.18)	4.283 (+0.15)	3.098 (+0.07)
K=8	2.366 (+0.22)	4.320 (+0.19)	3.120 (+0.09)
K=9	2.405 (+0.26)	4.351 (+0.22)	3.140 (+0.11)
K=10	2.442 (+0.29)	4.380 (+0.25)	3.160 (+0.13)
Gate overrides			
g_d=1, g_w=0 (deep only)	7.684 (+5.54)	8.291 (+4.16)	8.090 (+5.06)
g_d=0, g_w=1 (wide only)	7.574 (+5.43)	7.747 (+3.62)	8.655 (+5.63)
g_d=0.5, g_w=0.5 (uniform)	4.744 (+2.60)	6.707 (+2.58)	5.749 (+2.72)
g_d=1, g_w=1 (fully open)	8.541 (+6.39)	9.099 (+4.97)	9.211 (+6.19)
Shuffled gates	2.658 (+0.51)	5.071 (+0.94)	3.720 (+0.69)

Table 7: Inference-time ablations of the dual-path router. We report per-token cross-entropy loss on three Paloma sources. **Compute (top block):** forcing exactly K loop iterations confirms that loop budget matters, with diminishing returns past the trained value. Extra inference loops beyond training degrade the model monotonically rather than plateauing, indicating the loop schedule does not extrapolate past its training budget. **Gate overrides (bottom block):** both paths are necessary, disabling either (g_d=1, g_w=0 or g_d=0, g_w=1) collapses performance.

		$F_M = 80$M				$F_M = 160$M			
Config	K	d_{ffn}	$d_{\text{ffn}}^{\text{wide}}$	Params	Time	d_{ffn}	$d_{\text{ffn}}^{\text{wide}}$	Params	Time
PureWide	1	—	24576	719M	17.6h	—	50624	1361M	21.4h
PureLoop K=2	2	11392	—	398M	15.2h	24448	—	719M	19.9h
PureLoop K=3	3	7104	—	294M	16.6h	15744	—	502M	15.5h
PureLoop K=4	4	4864	—	238M	15.9h	11392	—	398M	17.7h
Dual α=25, K=2	2	1600	17920	644M	14.6h	4864	37440	1200M	14.8h
Dual α=25, K=3	3	576	17920	615M	14.7h	2752	37440	1153M	15.1h
Dual α=25, K=4	4	64	17920	606M	12.8h	1600	37440	1125M	15.0h
Dual α=50, K=2	2	4864	11392	559M	17.4h	11392	24448	1040M	21.3h
Dual α=50, K=3	3	2752	11392	511M	17.0h	7104	24448	936M	15.6h
Dual α=50, K=4	4	1600	11392	483M	16.5h	4864	24448	880M	19.0h
Dual α=75, K=2	2	8128	4864	483M	14.2h	17920	11392	880M	13.2h
Dual α=75, K=3	3	4864	4864	398M	13.3h	11392	11392	719M	14.5h
Dual α=75, K=4	4	3264	4864	360M	14.3h	8128	11392	644M	13.1h

Table 8: Per-configuration widths, parameter counts, and wall-clock training times at the $F_M = 80$M and $F_M = 160$M FFN-FLOP budgets.