Block-Level Recursion: Adaptive Test-Time Routing in Large Language Models

Anonymous Authors[1]

Abstract

Test-time routing improves frozen large language models (LLMs) by taking non-linear paths through their layers, without modifying weights or generating extra tokens. Existing approaches define route spaces that grow exponentially with depth, making them costly to search and hard to learn from. We therefore introduce **Block-Level Recursion** (BLR), a restricted route family that repeats a single contiguous block of transformer layers once. This reduces the number of routes from exponential to quadratic in the number of layers, making exhaustive per-instance evaluation tractable and the oracle upper bound directly measurable. Despite this restriction, BLR retains most of the routing potential. Across six model families and ten reasoning benchmarks, the optimal block varies across models, tasks, and individual inputs, with per-instance oracle gains of $+59.1\%$ on average and up to $+75.8\%$ on individual tasks. BLR also supports two practical policies: a single train-selected block (sBLR) that requires no router or per-input overhead, and a learned global router (aBLR) trained from dense per-instance rewards over all routes. sBLR already recovers a substantial fraction of the available gains, while aBLR improves further by selecting routes per input. With a frozen Qwen2.5-0.5B backbone, aBLR achieves higher accuracy than the unrouted Qwen2.5-7B model at lower FLOPs.

1. Introduction

Changing which layers a *frozen* model executes at inference time, without modifying weights or generating extra tokens, is a promising axis for improving accuracy. Layer-skipping methods reduce compute (Elhoushi et al., 2024; Luo et al., 2025a;b; Chen et al., 2025a), while layer-repetition methods improve it (Li et al., 2025; Heakl et al., 2026). Both take per-layer decisions, yielding action spaces of order 3^L that require heuristic search (Li et al., 2025) or approximate supervision (Heakl et al., 2026), and cannot exploit computation reuse across contiguous blocks. Recent recurrent-depth work shows that transformer layers naturally form functional blocks whose repetition unlocks latent reasoning capacity (Koishekenov et al., 2026; McLeish et al., 2025), suggesting block-level routing as the natural unit of computation reuse in frozen models.

We introduce **Block-Level Recursion** (BLR): a family of inference-time routes that repeat a contiguous block of transformer layers once before proceeding with the remaining layers. BLR defines $L(L-1)/2$ recursive paths, or $L(L-1)/2+1$ total routes including the default sequential path. This results in 277 total routes for a 24-layer model compared to the $3^{24} \approx 282$ billion routes in a per-layer action space. This compact structure makes exhaustive per-instance evaluation practical, converts routing into a reward-regression problem with dense per-sample supervision, and makes the *oracle gap* between the default and the best-route accuracy directly measurable.

Our exhaustive analysis across the Qwen2.5 model family (Qwen Team, 2025a) and five additional architectures reveals three consistent findings. First, the optimal BLR block is **model-specific**: within the same model family, the best block shifts substantially with scale. Second, it is **task-specific**: for a fixed model, different benchmarks favor different network regions. Third, and most importantly, it is **input-specific**: oracle BLR accuracy exceeds the unrouted baseline by an average of $+59.1\%$, with individual gains reaching $+75.8\%$, far beyond what any single fixed block can recover.

To close the oracle gap, we propose two routing policies, which are illustrated in Fig. 1. **Static BLR (sBLR)** selects a single train-optimal block once per backbone and applies it uniformly at inference, requiring no router and recovering a substantial fraction of the available gains. **Adaptive BLR (aBLR)** trains a lightweight global router that selects the BLR block per input from the frozen backbone's hidden states, recovering a further portion of the gap. With a Qwen2.5-0.5B backbone, aBLR achieves overall accuracy exceeding the unrouted Qwen2.5-7B model, a

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

$14\times$ reduction in parameter count and at lower FLOPs.

Our contributions are: (1) **BLR**, a structured, exhaustively enumerable family of $L(L-1)/2+1$ inference-time routes over frozen LLMs, (2) **an oracle analysis** showing the best BLR block is model-, task-, and input-specific with gains up to $+75.8\%$, and (3) **sBLR and aBLR**, a train-selected fixed block and a lightweight learned router. On Qwen2.5-0.5B, aBLR exceeds unrouted Qwen2.5-7B accuracy at lower FLOPs. We will release the per-instance per-route reward dataset.

2. Related Work

Routing and adaptive computation. Test-time routing methods in frozen LLMs include search over skip-or-loop choices (Li et al., 2025), learned per-layer skip/execute/repeat policies (Heakl et al., 2026), and layer-skipping for efficiency (Elhoushi et al., 2024; Luo et al., 2025a;b; Chen et al., 2025a; Men et al., 2024). All take local per-layer decisions over action spaces of order 3^L. Adaptive computation methods (Graves, 2016; Banino et al., 2021; Xin et al., 2020; Liu et al., 2020; Chen et al., 2024; Shan et al., 2024; Schuster et al., 2022; Hou et al., 2020; Raposo et al., 2024; Elbayad et al., 2020) reduce compute by halting early rather than reusing computation to improve accuracy.

Recurrence and internal computation. Universal Transformers (Dehghani et al., 2019) apply shared-weight layers iteratively, and later work adds computation through pause tokens (Goyal et al., 2024), latent reasoning (Hao et al., 2025; Chen et al., 2025b), and self-generated traces (Zelikman et al., 2024). Recurrent-depth architectures show that repeating layer subsets improves reasoning (McLeish et al., 2025; Geiping et al., 2025; Koishekenov et al., 2026), but require training or architectural changes rather than routing through an unmodified backbone.

Block-Level Recursion. Prior work spends additional computation through output tokens (CoT (Wei et al., 2022)), learned internal recurrence (Dehghani et al., 2019; Hao et al., 2025; McLeish et al., 2025; Geiping et al., 2025; Koishekenov et al., 2026), or large per-layer action spaces (Li et al., 2025; Heakl et al., 2026). Like recurrent-depth models, BLR allocates computation to a contiguous block without continued training, architectural changes, or additional tokens, and its $O(L^2)$ route family makes exhaustive evaluation tractable.

3. Block-Level Recursion

We frame inference-time adaptation as selecting a route through the layers of a frozen pretrained transformer. We begin by formalizing routes and route execution, then instantiate this framework with **Block-Level Recursion** (BLR), a structured family of routes that repeat a contiguous block of layers exactly once.

3.1. Routing Through Frozen Transformer Layers

Inputs and outputs. Each instance is a pair $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} = (x_1, \ldots, x_{T_x})$ is a prompt and $\boldsymbol{y} = (y_1, \ldots, y_{T_y})$ is a reference response.

Pretrained backbone. Let $f_{\boldsymbol{\theta}}$ be a frozen pretrained transformer with L layers and hidden dimension d. Each transformer block is a function

$$F_\ell : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}, \qquad \ell \in \{1, \ldots, L\},$$

where n is the sequence length. Let $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}|\times d}$ be the token embedding matrix and $\boldsymbol{W} \in \mathbb{R}^{d\times|\mathcal{V}|}$ the output projection. The standard forward pass computes hidden representations

$$\begin{aligned}\boldsymbol{H}_0(\boldsymbol{x}) &:= \boldsymbol{E}(\boldsymbol{x}),\\ \boldsymbol{H}_\ell(\boldsymbol{x}) &:= F_\ell\big(\boldsymbol{H}_{\ell-1}(\boldsymbol{x})\big), \quad \ell = 1, \ldots, L,\end{aligned}$$

yielding logits $\boldsymbol{z}_{\boldsymbol{\theta}}(\boldsymbol{x}) := \boldsymbol{W}^\top \boldsymbol{H}_L(\boldsymbol{x})$, from which the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$ is obtained autoregressively. Positional encodings and attention masks follow standard practice and are omitted from this notation for clarity.

Routes and route execution. A *route* $r = (\ell_1, \ldots, \ell_T)$ is an ordered sequence of layer indices that may revisit or skip layers; the *admissible route set* $\mathcal{R}$ collects all routes of interest. Executing r replaces the standard forward pass with $\boldsymbol{H}_t(\boldsymbol{x}; r) := F_{\ell_t}\big(\boldsymbol{H}_{t-1}(\boldsymbol{x}; r)\big)$, yielding logits $\boldsymbol{W}^\top \boldsymbol{H}_T(\boldsymbol{x}; r)$ and a routed distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, r)$. The *default route* $r_0 := (1, \ldots, L)$ recovers standard inference. Since $r_0 \in \mathcal{R}$, the per-instance oracle $\max_{r\in\mathcal{R}} p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, r)$ always weakly dominates unrouted inference (formal definitions and proof in App. A.1).

3.2. The BLR Route Family

We focus on a single-recursion route family. Given a start and end layer index pair (s, e) with $1 \le s < e \le L$, the recursive route $r_{s,e}$ inserts one additional pass over layers s through e:

$$r_{s,e} := (1, \ldots, e, \underbrace{s, \ldots, e}_{\text{BLR Block}}, e+1, \ldots, L).$$

We refer to (s, e) as the *BLR block*. The admissible route set is then

$$\mathcal{R}_{\text{BLR}} := \{r_0\} \cup \{r_{s,e} \mid 1 \le s < e \le L\},$$

which contains the default route together with $L(L-1)/2$ recursive paths. For typical transformer depths, this yields a few hundred total routes (e.g., 277 total routes for a

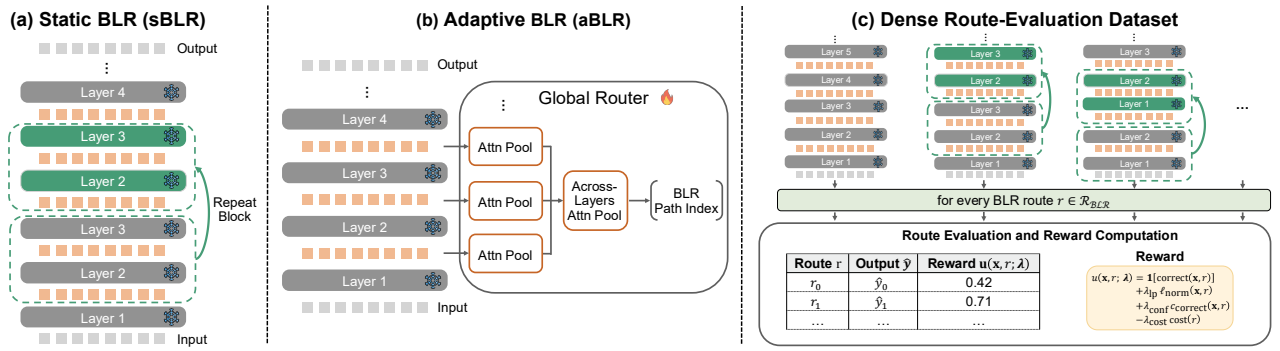


Figure 1. **Block-Level Recursion. (a)** `sBLR` repeats one fixed block for all inputs. **(b)** `aBLR` adds a lightweight router that selects the block per input from pooled hidden states. **(c)** Exhaustive per-route evaluation on training data produces dense supervision for `aBLR`.

24-layer model), making exhaustive per-instance evaluation practical. BLR thus introduces a single interpretable control choice: which contiguous block to repeat. BLR routes cannot be expressed as layer subsequences (layer-skipping) or as per-layer skip/execute/repeat decisions; see App. A.2 for a formal comparison.

3.3. Route-Selection Policies

The route set $\mathcal{R}_{\text{BLR}}$ specifies which inference-time paths are available, and choosing among them requires a separate policy. We consider two policies that share the same route set and frozen backbone.

Static BLR. `sBLR` selects a single recursive route using the training set and applies it to all inputs at inference, requiring no router and no per-input overhead (see Fig. 1a). Concretely, the selected route maximizes answer log-likelihood over the training set $\mathcal{D}$:

$$r^* = \arg \max_{r_{s,e} \in \mathcal{R}_{\text{BLR}} \setminus \{r_0\}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, r_{s,e}).$$

The default route is excluded during selection to isolate the effect of BLR, but can be included in practice if it performs best.

Adaptive BLR. `aBLR` selects the route separately for each input via a learned router $g_{\boldsymbol{\phi}}$ that maps $\boldsymbol{x}$ to scores over $\mathcal{R}_{\text{BLR}}$ (see Fig. 1b):

$$r(\boldsymbol{x}) = \arg \max_{r \in \mathcal{R}_{\text{BLR}}} g_{\boldsymbol{\phi}}(\boldsymbol{x})_r.$$

Both policies require supervision over which routes are effective for each input. We construct this supervision through the route-evaluation dataset described next.

3.4. Route-Evaluation Dataset

Since $\mathcal{R}_{\text{BLR}}$ contains only $L(L-1)/2 + 1$ routes, we can exhaustively evaluate every route on every labeled training example (see Fig. 1c). For each $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$ and each route $r \in \mathcal{R}_{\text{BLR}}$, we run the frozen backbone and record a scalar reward (defined in Sec. 3.6), producing a per-instance per-route reward vector $u(\boldsymbol{x}, \cdot) \in \mathbb{R}^{|\mathcal{R}_{\text{BLR}}|}$.

Route rewards are computed independently per input, so values are comparable within each example but do not leak information across the dataset. The dataset is built once on the training split with the frozen backbone, requiring no search or additional model training. It provides dense per-route supervision for `aBLR` and enables the per-instance oracle analysis.

3.5. Router Architecture

`aBLR` employs a *global* router that selects a route after observing representations from the full forward pass. The router follows a two-pass procedure. First, a preliminary pass through $f_{\boldsymbol{\theta}}$ collects hidden states for route selection, after which the model decodes with the selected BLR route. This is compatible with standard KV caching, as the selected route is fixed for the entire sequence.

To keep the router lightweight, input tokens are mean-pooled within W contiguous chunks per layer; the router $g_{\boldsymbol{\phi}}$ aggregates these across chunks and layers and maps to scores over $\mathcal{R}_{\text{BLR}}$, with the highest-scoring route selected (Fig. 1b; full details in App. E.1).

As an ablation, we also study a *local* router inspired by prior per-layer routing methods (App. E.2.1).

3.6. Training Objective

The route-evaluation dataset provides dense supervision, but multiple routes can produce correct answers for the same input. Since a binary correctness objective treats all such routes identically, we instead define a scalar route reward that scores prediction quality and penalizes compute.

Route reward. For $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$ and route $r \in \mathcal{R}_{\text{BLR}}$, the

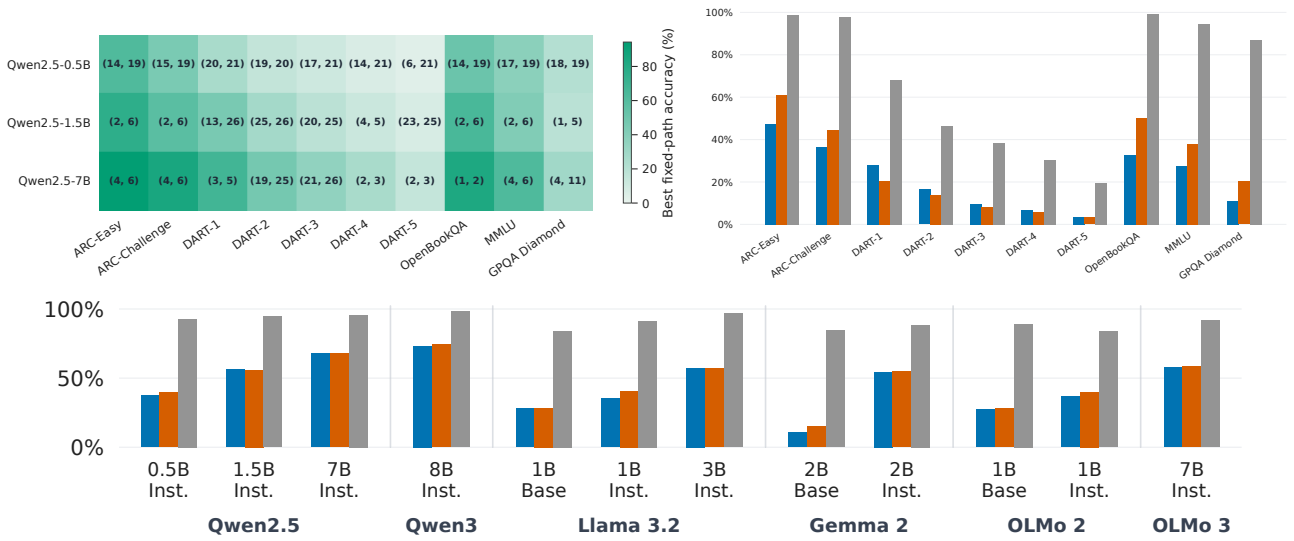


Figure 2. **Best BLR varies across tasks, inputs, and model families.** ■ baseline, ■ sBLR, ■ oracle. *Top-left*: optimal block (s, e) varies per benchmark. *Top-right*: instance-level oracle exceeds sBLR, showing input-dependent gaps. *Bottom*: same pattern across model families.

route reward is

$$u(\boldsymbol{x}, r; \boldsymbol{\lambda}) = \mathbf{1}[\text{correct}(\boldsymbol{x}, r)] + \lambda_{\text{lp}}\, \ell_{\text{norm}}(\boldsymbol{x}, r) + \lambda_{\text{conf}}\, c_{\text{correct}}(\boldsymbol{x}, r) - \lambda_{\text{cost}}\, \text{cost}(r) \quad (1)$$

where $\boldsymbol{\lambda} := (\lambda_{\text{lp}}, \lambda_{\text{conf}}, \lambda_{\text{cost}}) \in \mathbb{R}^3_{\geq 0}$. Here $\ell_{\text{norm}}(\boldsymbol{x}, r) \in [0, 1]$ is the normalized mean log probability of the answer tokens, $c_{\text{correct}}(\boldsymbol{x}, r) \in [0, 1]$ is an entropy-based confidence score masked to zero on incorrect routes, and $\text{cost}(r) \in [0, 1]$ is the route path length normalized by the maximum $2L$. The term ℓ_{norm} discriminates among incorrect routes by how close they come to the right answer, while c_{correct} separates correct but uncertain routes from correct and confident ones. Together, these terms form the target vector $u(\boldsymbol{x}, \cdot; \boldsymbol{\lambda}) \in \mathbb{R}^{|\mathcal{R}_{\text{BLR}}|}$ for each input.

Reward regression. A single-label classification objective is not appropriate, as it cannot distinguish among valid routes. Instead, the router performs reward regression: it predicts a scalar $g_{\boldsymbol{\phi}}(\boldsymbol{x})_r \approx u(\boldsymbol{x}, r; \boldsymbol{\lambda})$ for each route r, analogous to fitted Q iteration in batch reinforcement learning (Ernst et al., 2005), and the MSE loss directly optimizes this approximation. The global router predicts $g_{\boldsymbol{\phi}}(\boldsymbol{x}) \in \mathbb{R}^{|\mathcal{R}_{\text{BLR}}|}$ and is trained with

$$\mathcal{L}_{\text{global}}(\boldsymbol{\phi}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}} \| g_{\boldsymbol{\phi}}(\boldsymbol{x}) - u(\boldsymbol{x}, \cdot; \boldsymbol{\lambda}) \|_2^2 . \quad (2)$$

At inference, the global router selects $\arg\max_{r \in \mathcal{R}_{\text{BLR}}} g_{\boldsymbol{\phi}}(\boldsymbol{x})_r$.

4. Experiments

We first characterize BLR empirically at the model-, task-, and instance-level (Sec. 4.2) to quantify the input-dependent gains available through routing. We then train a router (Sec. 4.3) and measure how much of these gains a learned policy recovers in practice, including quantitative results and comparison to search-based routing. Finally, we provide qualitative analysis (Sec. 4.4) to examine when routing helps and how the learned router allocates computation across inputs.

4.1. Experimental Setup

Models. We use the Qwen2.5 (Qwen Team, 2025a) base model family (0.5B, 1.5B, and 7B) as the primary evaluation family. To test generality, we also analyze BLR across various model families including Qwen3 (Qwen Team, 2025b), Llama 3.2 (Dubey et al., 2024), Gemma 2 (Gemma Team, 2024), OLMo 2 (Walsh et al., 2025), and OLMo 3 (OLMo Team, 2025). We report the results in Sec. 4.2.

Benchmarks and split. We evaluate on reasoning benchmarks spanning commonsense, arithmetic, multitask understanding, and graduate-level science: ARC (Clark et al., 2018), DART (Tong et al., 2024), MMLU (Hendrycks et al., 2021), GPQA Diamond (Rein et al., 2024), and OpenBookQA (Mihaylov et al., 2018). ARC-Easy, ARC-Challenge, and DART-1 through DART-5 serve as in-domain training benchmarks for both sBLR and aBLR; OpenBookQA, MMLU, and GPQA Diamond are held out as out-of-distribution test sets with no training examples.

Exhaustive BLR evaluation. For each model, we exhaustively evaluate all valid BLR blocks (s, e) with $1 \leq s < e \leq L$, reporting accuracy under greedy decoding with no weight modifications. Benchmark-specific answer extraction follows the rules described in App. G.

Table 1. **Routing performance on Qwen2.5 base models.** Unrouted (r_0) denotes the default forward pass. Averages are weighted by dataset size. Deltas in parenthesis are absolute changes relative to Unrouted (r_0). Gray rows show Oracle BLR upper bounds.

Size	Method	In-domain			OOD				Overall
		ARC	DART	Avg.	OBQA	GPQA	MMLU	Avg.	Avg.
0.5B	Unrouted (r_0)	43.8	9.0	27.9	32.6	11.1	27.4	27.4	27.5
	Random BLR	25.3 (-18.5)	5.3 (-3.7)	16.1 (-11.7)	19.8 (-12.8)	13.6 (+2.5)	16.3 (-11.1)	16.4 (-11.0)	16.3 (-11.2)
	Dr.LLM (Heakl et al., 2026)	43.8 (+0.0)	**10.8 (+1.8)**	27.9 (+0.0)	32.8 (+0.2)	11.6 (+0.5)	27.4 (+0.0)	27.4 (+0.0)	27.5 (+0.0)
	sBLR	55.6 (+11.8)	7.5 (-1.6)	33.6 (+5.7)	50.2 (+17.6)	**20.2 (+9.1)**	37.9 (+10.5)	38.1 (+10.8)	36.7 (+9.2)
	aBLR	**56.4 (+12.6)**	8.8 (-0.2)	**34.6 (+6.7)**	**50.8 (+18.2)**	19.2 (+8.1)	**38.2 (+10.8)**	**38.4 (+11.0)**	**37.2 (+9.7)**
	Oracle BLR	98.4 (+54.6)	33.0 (+23.9)	68.4 (+40.5)	99.2 (+66.6)	86.9 (+75.8)	94.7 (+67.3)	94.7 (+67.4)	86.6 (+59.1)
1.5B	Unrouted (r_0)	36.6	16.4	27.3	33.8	13.1	24.6	24.8	25.6
	Random BLR	31.8 (-4.7)	9.3 (-7.1)	21.5 (-5.8)	29.0 (-4.8)	12.6 (-0.5)	21.0 (-3.7)	21.1 (-3.7)	21.3 (-4.3)
	Dr.LLM (Heakl et al., 2026)	36.6 (+0.0)	**18.9 (+2.5)**	27.3 (+0.0)	33.8 (+0.0)	13.1 (+0.0)	24.6 (+0.0)	24.8 (+0.0)	25.6 (+0.0)
	sBLR	70.2 (+33.6)	13.0 (-3.4)	44.0 (+16.7)	**65.0 (+31.2)**	**17.2 (+4.0)**	42.6 (+18.0)	43.0 (+18.2)	43.3 (+17.7)
	aBLR	**72.7 (+36.1)**	15.5 (-0.9)	**46.5 (+19.2)**	64.4 (+30.6)	**17.2 (+4.0)**	**43.8 (+19.1)**	**44.1 (+19.3)**	**44.8 (+19.3)**
	Oracle BLR	96.9 (+60.3)	47.7 (+31.3)	74.3 (+47.0)	95.6 (+61.8)	74.2 (+61.1)	87.8 (+63.1)	87.9 (+63.1)	83.7 (+58.1)
7B	Unrouted (r_0)	53.0	29.6	42.3	46.6	15.7	32.4	32.6	35.6
	Random BLR	45.5 (-7.5)	18.8 (-10.8)	33.3 (-9.0)	35.0 (-11.6)	14.1 (-1.5)	27.8 (-4.5)	27.9 (-4.7)	29.6 (-6.0)
	Dr.LLM (Heakl et al., 2026)	55.7 (+2.7)	**32.2 (+2.6)**	43.7 (+1.4)	48.4 (+1.8)	15.7 (+0.0)	34.6 (+2.2)	36.1 (+3.5)	37.9 (+2.3)
	sBLR	89.2 (+36.2)	27.9 (-1.7)	61.1 (+18.8)	**82.6 (+36.0)**	18.2 (+2.5)	56.7 (+24.3)	57.0 (+24.4)	58.3 (+22.7)
	aBLR	**90.9 (+37.9)**	27.7 (-1.9)	**62.0 (+19.7)**	80.8 (+34.2)	**29.3 (+13.6)**	**61.7 (+29.3)**	**61.9 (+29.3)**	**61.9 (+26.3)**
	Oracle BLR	99.5 (+46.4)	63.6 (+34.0)	83.0 (+40.7)	99.2 (+52.6)	90.4 (+74.7)	96.2 (+63.8)	96.2 (+63.6)	92.2 (+56.6)

Baselines and reference points. Random routing samples uniformly from $\mathcal{R}_{\text{BLR}}$. sBLR applies a single train-selected BLR block per model (13–21 for 0.5B, 2–6 for 1.5B, and 1–2 for 7B). We compare to Dr.LLM (Heakl et al., 2026), a state-of-the-art learned routing approach for frozen LLMs, and to MCTS (Li et al., 2025), which provides an oracle search baseline in a larger per-layer route space. We do not include layer-skipping methods, as they target efficiency rather than accuracy, or CoT and self-consistency, which increase computation through additional output tokens and operate along a complementary axis.

Metrics. We report per-dataset accuracies along with three summary metrics. The *in-domain average* aggregates ARC and DART, the *OOD average* aggregates OpenBookQA, MMLU, and GPQA Diamond, and the *overall average* aggregates all datasets.

Adaptive BLR training. aBLR is trained on the route-evaluation dataset from Sec. 3.4, using only examples from the in-domain training benchmarks. The router is trained offline with the loss in Eq. (2) and the reward in Eq. (1), so no online generation or test-time search is used during router training. At inference, the trained router selects one route per input, and the frozen backbone decodes with that route using greedy decoding.

Training details. Unless noted otherwise, reported aBLR results use a global router trained with AdamW, exponential moving average, and a cosine learning rate schedule. The checkpoint is selected based on the highest accuracy gain. We use a single fixed aBLR configuration across the Qwen2.5 base models: hidden size 256, pooling window 32, and a confidence-heavy reward profile $\boldsymbol{\lambda} = (0.1, 0.5, 0.2)$ (see App. E.2.2).

4.2. Evaluating BLR Across Models, Tasks, and Instances

Frozen LLMs exhibit route-dependent variation that standard single-pass inference does not exploit. We characterize this structure on the Qwen2.5 family by exhaustively evaluating every BLR block on each benchmark. The findings proceed from fixed to input-dependent choices: variation across models (Finding 1), variation across tasks for a fixed model (Finding 2), and the remaining gains when the BLR block varies by instance (Finding 3).

Finding 1: the best BLR block depends on the model. We first ask whether one BLR block works well across model scales. Table 1 shows that sBLR improves over the baseline at all scales, but the best-performing block differs across models. Specifically, Qwen2.5-0.5B achieves its best result with layers 13–21, Qwen2.5-1.5B with layers 2–6, and Qwen2.5-7B with layers 1–2. Thus, there is no single BLR block that works best across the model family, and BLR must be selected separately for each backbone.

Finding 2: the best BLR block depends on the task. We next fix the backbone and vary the benchmark. The best BLR block again changes. Figure 2 shows that Qwen2.5-0.5B achieves its best result with layers 14–19 on ARC-Easy, layers 6–21 on DART-5, and layers 17–19 on MMLU. These blocks cover early, middle, and late parts of the network. A single model-level block therefore misses task-level variation. When task identity is available, BLR should be selected at the task level rather than only once per model.

Finding 3: the best BLR block depends on the input. Model-level and task-level choices still assign one block to many examples. We therefore measure how much accuracy remains available when the BLR block can vary per input. For each evaluation example, an oracle selects the best route in $\mathcal{R}_{\text{BLR}}$ using the reference answer. This oracle is not a deployable method, but it measures the maximum gain available within the BLR route family. The results substantially exceed both the unrouted baseline and sBLR (see Fig. 2 and Tab. 1). For Qwen2.5-0.5B, the overall average rises from 27.5% to 86.6%, with the largest single-task gain reaching +75.8 points on GPQA Diamond. Thus, the best BLR block is not only model-specific and task-specific, but also input-specific. This motivates aBLR, where a router predicts the BLR block per input without access to the reference answer.

Generality across Model Families. The same three-finding pattern holds across Qwen3 (Qwen Team, 2025b), Llama 3.2 (Dubey et al., 2024), Gemma 2 (Gemma Team, 2024), OLMo 2 (Walsh et al., 2025), and OLMo 3 (OLMo Team, 2025) (see Fig. 2 and App. D.1 for per-dataset results).

4.3. Adaptive BLR and Its Generalization

Findings 1–3 establish that frozen LLMs contain large instance-specific routing gaps but sBLR closes only a fraction of them. We now train a router g_ϕ to implement aBLR, selecting routes from $\mathcal{R}_{\text{BLR}}$ per input without oracle access.

Across all eight datasets, aBLR achieves the best non-oracle overall average at every scale (see Tab. 1): 37.2% at 0.5B, 44.8% at 1.5B, and 61.9% at 7B. These results substantially outperform the unrouted baseline, Dr.LLM, and sBLR.

In-domain Results. Table 1 shows three consistent patterns. First, random routing is substantially weaker than the unrouted baseline, confirming that gains require structured route selection rather than arbitrary route perturbation. Second, although Dr.LLM improves some DART settings, these gains do not translate into the same improvements on ARC or on the in-domain average. Third, sBLR is much stronger than random routing and the Dr.LLM on the in-domain average, showing that sBLR already recovers part of the available gains. aBLR further improves on sBLR across all three model sizes, raising the in-domain average from 33.6% to 34.6% at 0.5B, from 44.0% to 46.5% at 1.5B, and from 61.1% to 62.0% at 7B.

OOD Generalization. We next test whether these gains transfer beyond the training benchmarks. Table 1 shows that BLR generalizes out of distribution. The Dr.LLM baseline provides limited OOD improvements, with gains mainly at the largest scale, while BLR improves the OOD average more consistently. aBLR improves the OOD average over sBLR at all three scales (38.1% → 38.4%, 43.0% → 44.1%, and 57.0% → 61.9%), leading to higher overall averages (37.2%, 44.8%, 61.9%). These results indicate that both the learned routing policy and the BLR route family transfer to unseen tasks.

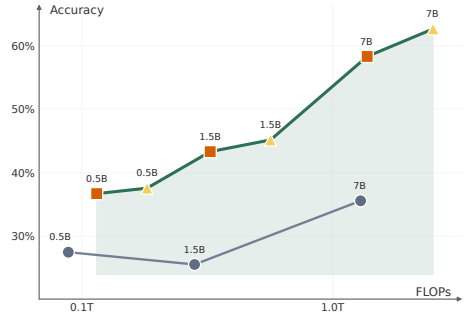


Figure 3. **Accuracy–FLOPs tradeoff on all selected datasets.** ● baseline (unrouted), ■ sBLR, and ▲ aBLR. Routed models extend the Pareto frontier by reaching higher accuracy at competitive compute.

Efficiency Analysis. sBLR increases computation by re-executing a block of layers, while aBLR adds further cost through the router and a preliminary pass. Figure 3 shows that both methods extend the accuracy-FLOPs Pareto frontier of the Qwen2.5 family. aBLR improves further over sBLR at both the 0.5B and 7B scales.

Oracle BLR consistently outperforms MCTS (Li et al., 2025) on ARC and DART across all three Qwen2.5 base models, confirming that search difficulty dominates the expressivity advantage of larger action spaces (see App. E.2.4).

4.4. Qualitative Analysis

Qualitative analysis of aBLR routing behavior, format repairs, and semantic fixes appears in Apps. F.1 to F.3.

5. Conclusion

Block-Level Recursion (BLR) enables inference-time adaptation in frozen language models by re-executing a contiguous block of layers without modifying weights or generating additional tokens. Our exhaustive evaluation reveals large model-, task-, and instance-specific routing gaps across multiple model families. A single train-selected block (sBLR) already recovers a substantial fraction of this gap, while a learned global router (aBLR) improves further by selecting routes per input, shifting the accuracy–compute frontier of the Qwen2.5 family. Unlike CoT, BLR operates entirely within the latent space of the frozen backbone and requires no additional tokens. The large instance-level oracle gap suggests that pretrained models contain latent, input-dependent computation patterns that fixed execution routes do not exploit.

Limitations and future directions are discussed in App. B.

References

Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder, 2021.

Chen, L., Shan, R., Wang, H., Wang, L., Liu, Z., Luo, R., Wang, J., Alinejad-Rokny, H., and Yang, M. CLaSp: In-context layer skip for self-speculative decoding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31608–31618, Vienna, Austria, 2025a. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1525.

Chen, Y., Pan, X., Li, Y., Ding, B., and Zhou, J. EE-LLM: Large-scale training and inference of early-exit large language models with 3d parallelism. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 7163–7189. PMLR, 2024.

Chen, Y., Shang, J., Zhang, Z., Xie, Y., Sheng, J., Liu, T., Wang, S., Sun, Y., Wu, H., and Wang, H. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259, Vienna, Austria, 2025b. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1369.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. Universal transformers. In *International Conference on Learning Representations*, 2019.

Dubey, A. et al. The llama 3 herd of models, 2024.

Elbayad, M., Gu, J., Grave, E., and Auli, M. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

Elhoushi, M., Shrivastava, A., Liskovich, D., Hosmer, B., Wasti, B., Lai, L., Mahmoud, A., Acun, B., Agarwal, S., Roman, A., Aly, A., Chen, B., and Wu, C.-J. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Geiping, J., McLeish, S. M., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Gemma Team. Gemma 2: Improving open language models at a practical size, 2024.

Goyal, S., Ji, Z., Rawat, A. S., Menon, A. K., Kumar, S., and Nagarajan, V. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024.

Graves, A. Adaptive computation time for recurrent neural networks, 2016.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J. E., and Tian, Y. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025.

Heakl, A., Gubri, M., Khan, S., Yun, S., and Oh, S. J. Dr.LLM: Dynamic layer routing in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Hou, L., Huang, Z., Shang, L., Jiang, X., Chen, X., and Liu, Q. DynaBERT: Dynamic BERT with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Koishekenov, Y., Lipani, A., and Cancedda, N. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts. In *Workshop on Latent & Implicit Thinking – Going Beyond CoT Reasoning*, 2026.

Li, Z., Li, Y., and Zhou, T. Skip a layer or loop it? test-time depth adaptation of pretrained LLMs, 2025.

Liu, W., Zhou, P., Wang, Z., Zhao, Z., Deng, H., and Ju, Q. FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6035–6044, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.537.

Luo, X., Wang, W., and Yan, X. Adaptive layer-skipping in pre-trained LLMs. In *Second Conference on Language Modeling*, 2025a. doi: 10.48550/arXiv.2503.23798.

Luo, X., Wang, W., and Yan, X. DiffSkip: Differential layer skipping in large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 7221–7231, Vienna, Austria, 2025b. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.377.

McLeish, S., Li, A., Kirchenbauer, J., Kalra, D. S., Bartoldson, B. R., Kailkhura, B., Schwarzschild, A., Geiping, J., Goldstein, T., and Goldblum, M. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025. doi: 10.48550/arXiv.2511.07384.

Men, X., Xu, M., Wang, B., Zhang, Q., Lin, H., Lu, Y., Han, X., and Chen, W. ShortGPT: Layers in large language models are more redundant than you expect, 2024.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260.

OLMo Team. OLMo 3 technical report, 2025.

Qwen Team. Qwen2.5 technical report, 2025a.

Qwen Team. Qwen3 technical report, 2025b.

Raposo, D., Ritter, S., Richards, B., Lillicrap, T., Humphreys, P. C., and Santoro, A. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Shan, W., Meng, L., Zheng, T., Luo, Y., Li, B., Wang, J., Xiao, T., and Zhu, J. Early exit is a natural capability in transformer-based models: An empirical study on early exit without joint optimization, 2024.

Tong, Y., Zhang, X., Wang, R., Wu, R., and He, J. DART-Math: Difficulty-aware rejection tuning for mathematical problem-solving. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7821–7846, 2024. doi: 10.52202/079017-0251.

Walsh, E. P. et al. 2 OLMo 2 furious (COLM's version). In *Second Conference on Language Modeling*, 2025.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.204.

Zelikman, E., Harik, G. R., Shao, Y., Jayasiri, V., Haber, N., and Goodman, N. D. Quiet-star: Language models can teach themselves to think before speaking. In *First Conference on Language Modeling*, 2024.

A. Formal Route Framework

A.1. Route Definitions and Oracle Bound

We provide the formal definitions and proof omitted from the main text for brevity.

Definition A.1 (Route and route execution)**.** A *route* is an ordered sequence of layer indices $r = (\ell_1, \ldots, \ell_T) \in \{1, \ldots, L\}^T$, where $T \geq 1$: routes may revisit layers or skip layers depending on the sequence of indices. The *admissible route set* $\mathcal{R} \subseteq \bigcup_{T \geq 1} \{1, \ldots, L\}^T$ collects all routes of interest. Route execution computes

$$\begin{aligned} \boldsymbol{H}_0(\boldsymbol{x}; r) &:= \boldsymbol{E}(\boldsymbol{x}), \\ \boldsymbol{H}_t(\boldsymbol{x}; r) &:= F_{\ell_t}\big(\boldsymbol{H}_{t-1}(\boldsymbol{x}; r)\big), \quad t = 1, \ldots, T, \end{aligned}$$

with logits $\boldsymbol{z}_{\boldsymbol{\theta}}(\boldsymbol{x}; r) := \boldsymbol{W}^\top \boldsymbol{H}_T(\boldsymbol{x}; r)$, inducing the routed conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, r)$. The *default route* $r_0 := (1, 2, \ldots, L)$ recovers standard transformer computation; we assume $r_0 \in \mathcal{R}$.

Proposition A.2 (Best admissible route dominates the default route)**.** *Fix an input–output pair $(\boldsymbol{x}, \boldsymbol{y})$ and an admissible route set $\mathcal{R}$ with $r_0 \in \mathcal{R}$. Then*

$$\max_{r \in \mathcal{R}} \; p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, r) \;\geq\; p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}, r_0).$$

Proof. Since $r_0 \in \mathcal{R}$, the maximum over $\mathcal{R}$ is at least the value attained by the default route r_0. □

A.2. Route Family Comparison

Prior methods as route families. The choice of $\mathcal{R}$ determines which compute patterns are available at inference time. Layer-skipping methods define $\mathcal{R}_{\text{skip}}$ as the set of ordered subsequences of the default route,

$$\mathcal{R}_{\text{skip}} = \big\{ r \in \{1, \ldots, L\}^T : T \leq L,\ \ell_i < \ell_j \text{ for all } i < j \big\},$$

so every admissible route only removes computation (Elhoushi et al., 2024; Luo et al., 2025a;b; Chen et al., 2025a). More general methods allow per-layer actions such as skip, execute, or repeat, yielding a combinatorial route space that grows exponentially with depth (on the order of 2^L or 3^L) (Heakl et al., 2026; Li et al., 2025). In contrast, the set of BLR routes introduces a distinct design space: it inserts a single contiguous repeated segment into the forward pass, which cannot be expressed by pure skipping and is not enforced by per-layer action spaces.

B. Limitations and Future Work

Limitations. BLR operates on a frozen backbone and can only reuse capabilities already present in the pretrained weights. It does not match methods that adapt model parameters to the target domain, such as instruction tuning or parameter-efficient fine-tuning. In addition, BLR is restricted to a single repeated block, which enables exhaustive evaluation but limits the expressivity of admissible routes.

Future work. Several directions follow from this work. First, future work should identify alternative route families that retain the tractability of BLR while increasing expressivity. This includes extensions such as multiple block repetitions as well as entirely different route structures that remain small enough to enable exhaustive evaluation. Second, our results raise the question of *routability*: how pretrained models can be designed or trained so that useful alternative computation paths emerge more reliably. Third, rather than applying BLR post hoc, routing could be integrated directly into model training, allowing the model to learn representations that are inherently compatible with Block-Level Recursion and input-dependent execution paths.

C. Broader Impacts

This work studies inference-time routing for frozen language models. Its main potential benefit is improved accuracy without modifying model weights or increasing the length of generated outputs. This could make stronger reasoning performance more accessible when retraining or deploying larger models is impractical. BLR may also reduce some forms of compute waste by reusing existing model parameters more effectively.

The same capability can also improve systems that are used in harmful settings, since BLR is a general method for increasing the accuracy of frozen LLMs. It does not introduce new data sources, user modeling, or direct deployment mechanisms, but it could be combined with models used for misinformation, automation, or other dual-use applications. We therefore view BLR as a general inference-time scaling technique whose risks are inherited from the underlying model and deployment context. Standard safeguards for LLM deployment, including model access controls, monitoring, and task-specific safety evaluation, remain necessary.

D. Additional Empirical Analysis

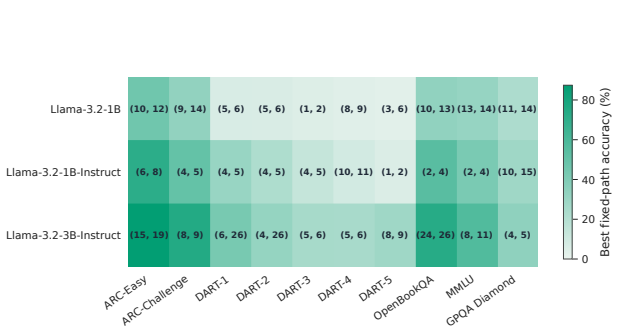
D.1. Generality Across Model Families

Section 4.2 establishes that the optimal BLR block is model-, task-, and instance-specific across six model families using weighted-average summaries. Here we provide the per-dataset figures underlying those summaries, showing that the same patterns hold at the level of individual benchmarks rather than only in aggregate.

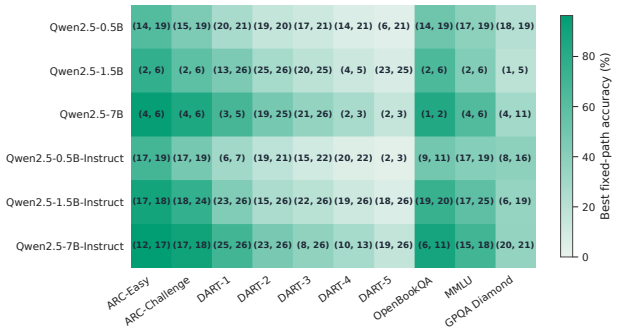
Models. We report per-dataset results for the following checkpoints:

- **Qwen2.5** (Qwen Team, 2025a): 0.5B, 1.5B, and 7B base and instruction-tuned models.
- **Llama-3.2** (Dubey et al., 2024): 1B base, 1B-Instruct, and 3B-Instruct.
- **Qwen3** (Qwen Team, 2025b): Qwen3-8B.
- **OLMo-3-7B-Instruct** (OLMo Team, 2025): an instruction-tuned model from the fully open OLMo series.
- **Gemma-2-2B** (Gemma Team, 2024): base and instruction-tuned variants.
- **OLMo-2** (Walsh et al., 2025): base and instruction-tuned variants.

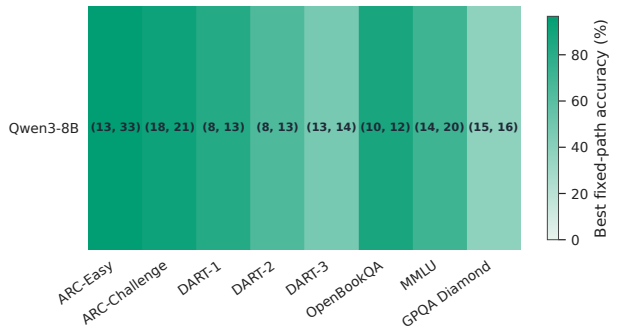
Results. Figures 4 to 6 report the per-dataset results across all six model families. The task-specific figures show that the optimal BLR block varies substantially across benchmarks for every model, with no architecture or training recipe concentrating on a single layer range. The oracle figures show a consistent gap between the unrouted baseline, the best `sBLR` block, and the instance-level oracle on each individual benchmark. The model-, task-, and instance-specific patterns reported in Sec. 4.2 therefore hold at the dataset level as well, across both base and instruction-tuned checkpoints.



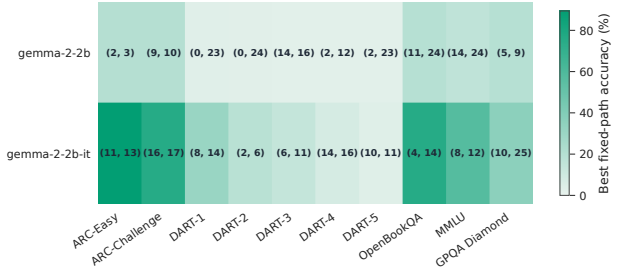
(a) Llama-3.2 family



(b) Qwen2.5 family



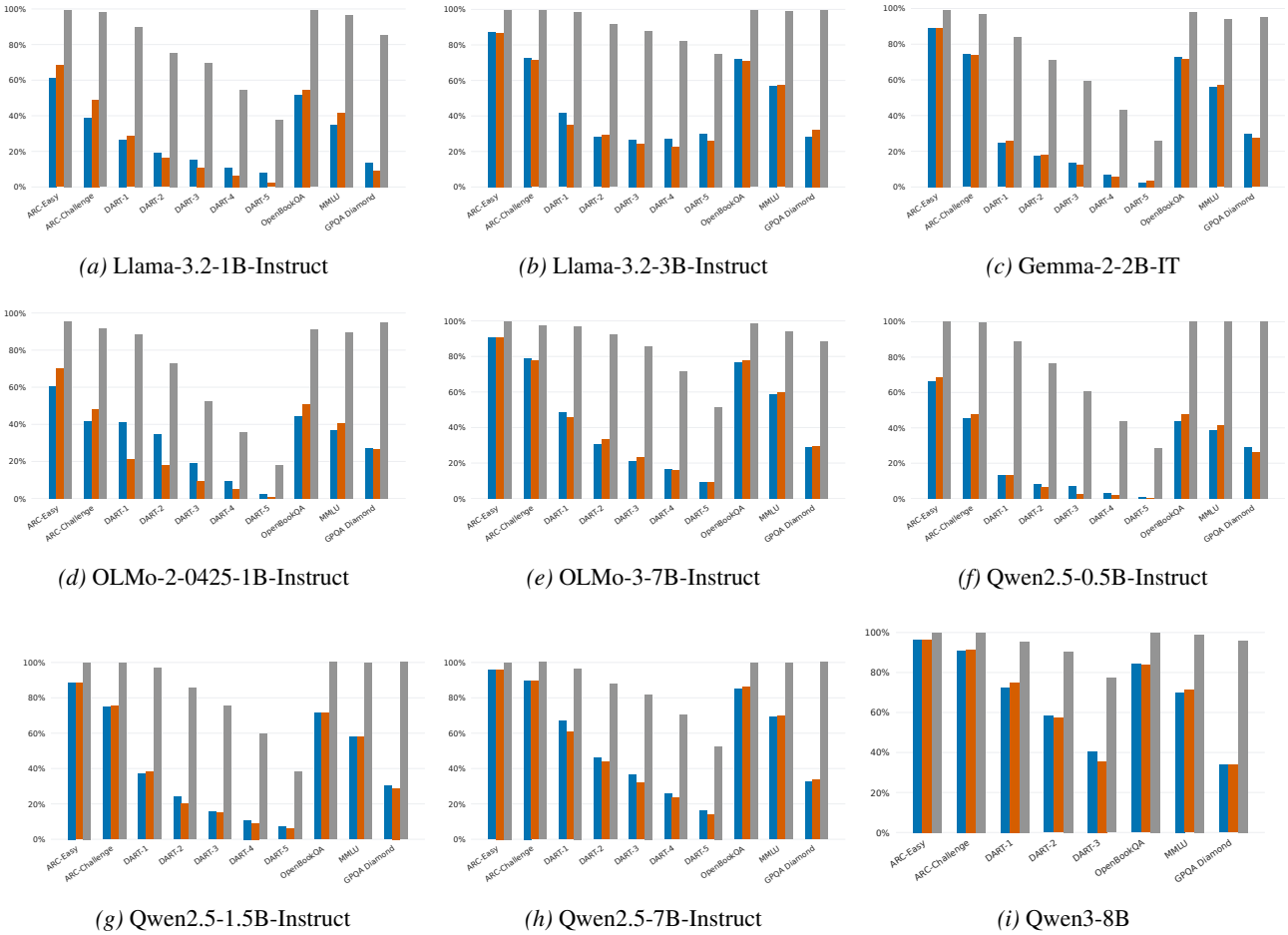
(c) Qwen3 family



(d) Gemma-2 family

(e) OLMo family

Figure 4. **The optimal BLR block is task-specific across model families.** For every model family, the optimal BLR block changes across benchmarks rather than concentrating on a single layer range. This decomposes the aggregate result in Sec. 3.6 into per-checkpoint, per-benchmark heatmaps.

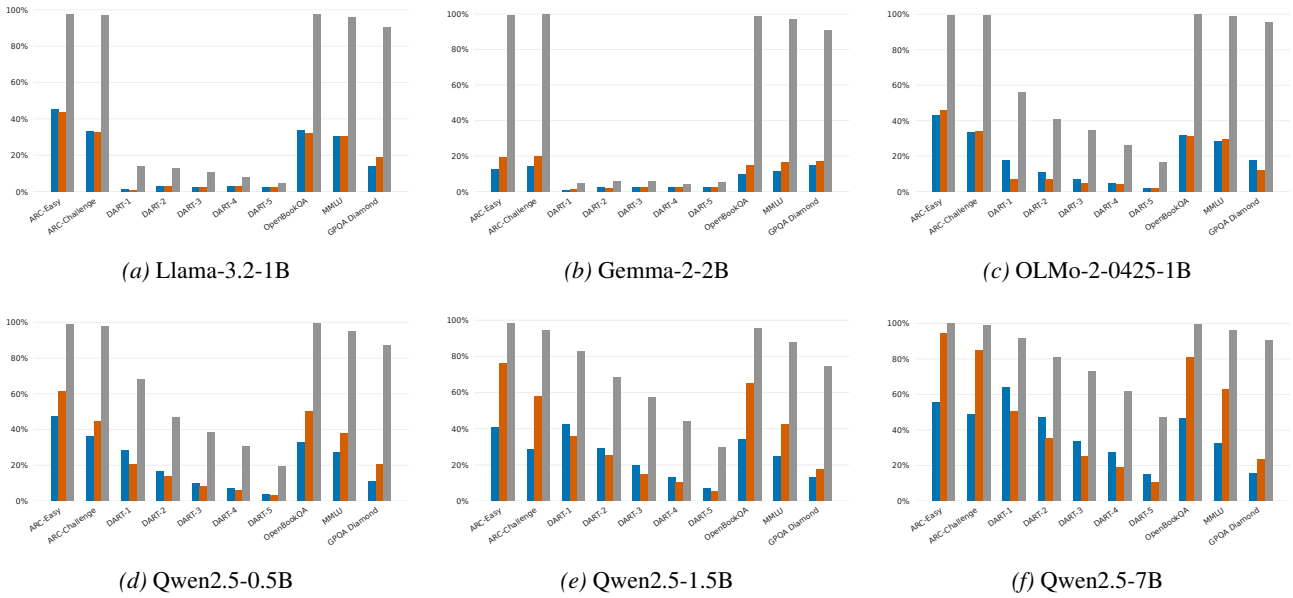


(a) Llama-3.2-1B-Instruct *(b)* Llama-3.2-3B-Instruct *(c)* Gemma-2-2B-IT

(d) OLMo-2-0425-1B-Instruct *(e)* OLMo-3-7B-Instruct *(f)* Qwen2.5-0.5B-Instruct

(g) Qwen2.5-1.5B-Instruct *(h)* Qwen2.5-7B-Instruct *(i)* Qwen3-8B

Figure 5. **Instance-specific BLR gaps persist across instruction-tuned checkpoints.** ■ baseline (unrouted), ■ best `sBLR` block, and ■ oracle. On every benchmark, the instance-level oracle substantially exceeds both the baseline and the best `sBLR` block, showing that input-dependent BLR gains persist after instruction tuning.



(a) Llama-3.2-1B *(b)* Gemma-2-2B *(c)* OLMo-2-0425-1B

(d) Qwen2.5-0.5B *(e)* Qwen2.5-1.5B *(f)* Qwen2.5-7B

Figure 6. **Instance-specific BLR gaps across base checkpoints.** ■ baseline (unrouted), ■ best `sBLR` block, and ■ oracle. On every benchmark, the best `sBLR` block improves over the baseline while the instance-level oracle remains substantially stronger, indicating that input-dependent BLR gains hold at the per-dataset level across architectures.

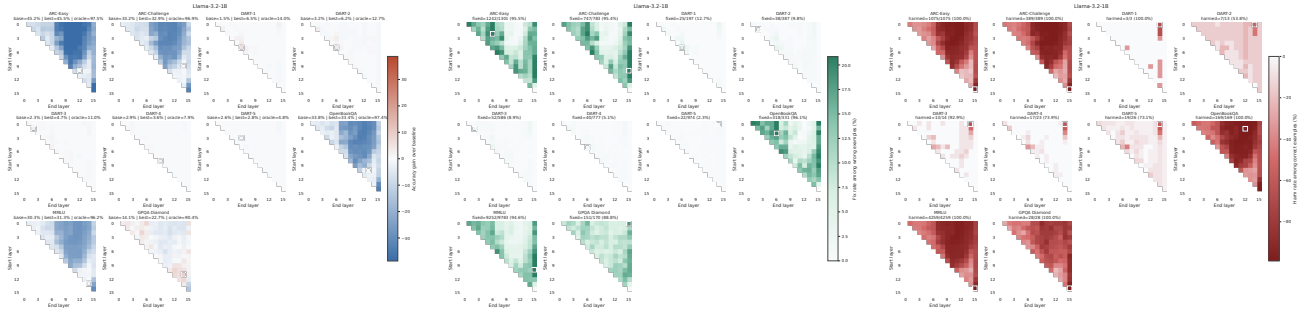
D.2. Heatmaps

This section reports the full BLR search space for each checkpoint. For each model, we show three groups of heatmaps. **Accuracy gain** reports the change in accuracy relative to the no-recursion baseline. **Fix count** reports the number of baseline-wrong examples that a recursive block corrects. **Harm count** reports the number of baseline-correct examples that a recursive block flips to incorrect. Each panel fixes one benchmark and plots the recursive block by start and end layer. The figures cover ARC-Easy, ARC-Challenge, DART-1 through DART-5, OpenBookQA, MMLU, and GPQA Diamond.

These three views separate the net effect of recursion into its two components. Accuracy gain captures the overall impact of a fixed block. Fix count identifies where recursion recovers missed examples. Harm count identifies where recursion degrades correct predictions. Reading the three heatmaps together shows whether a region is useful, merely active, or brittle.

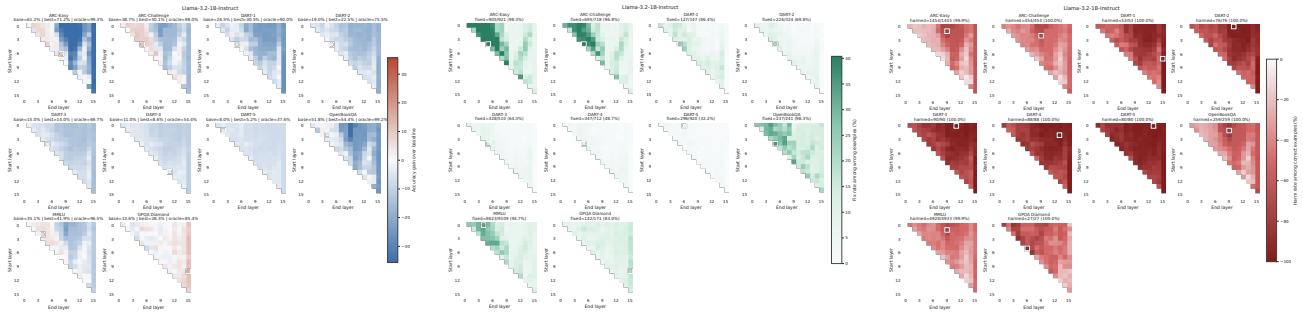
The heatmaps make the empirical findings from Sec. 4.2 explicit. Useful recursive blocks form structured regions rather than isolated cells, but these regions shift across tasks and models. Many blocks are harmful, especially when recursion starts early and spans a large portion of the network. The oracle values remain substantially higher than the best fixed-block values, even when clear high-gain regions exist. This gap supports input-dependent routing: no single recursive block explains the gains.

Across model families, the same qualitative pattern appears. On the Qwen2.5 base models, ARC, OpenBookQA, MMLU, and GPQA often exhibit clear high-gain regions, while several DART variants show weaker or negative fixed-block gains despite large oracle values. Outside Qwen2.5, structured regions persist, but their location and strength vary across checkpoints. The fix-count heatmaps show that gains arise from task-specific regions of the layer triangle, while the harm-count heatmaps reveal broad regions where recursion flips correct answers. This explains why random BLR performs poorly in Tab. 1: recursion is not uniformly beneficial, and effective routing must avoid blocks that introduce more harms than fixes.



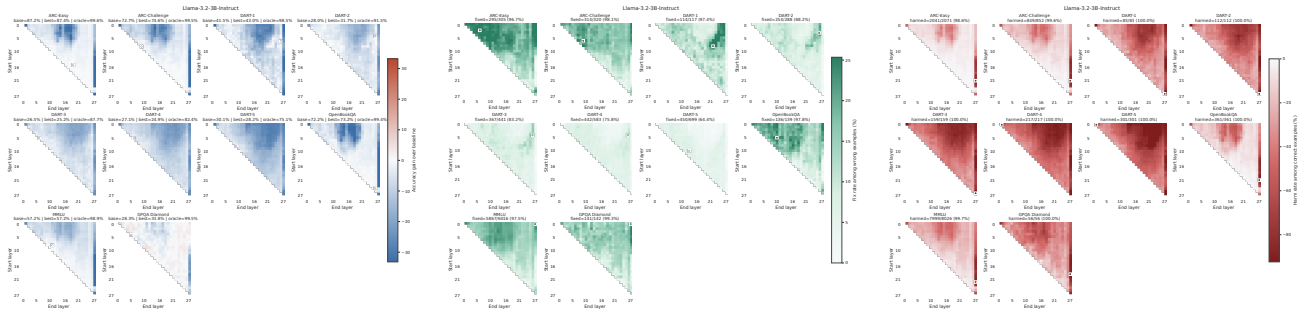
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 7. **Llama-3.2-1B.** Full BLR heatmaps for Llama-3.2-1B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



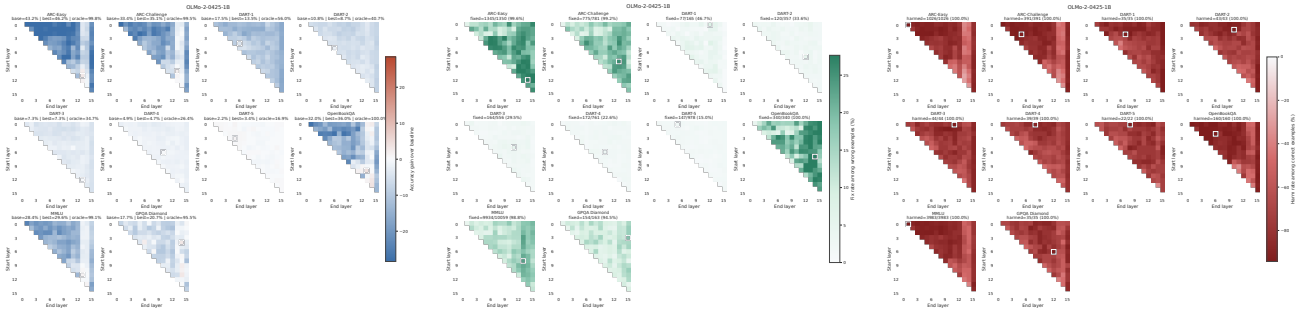
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 8. **Llama-3.2-1B-Instruct.** Full BLR heatmaps for Llama-3.2-1B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



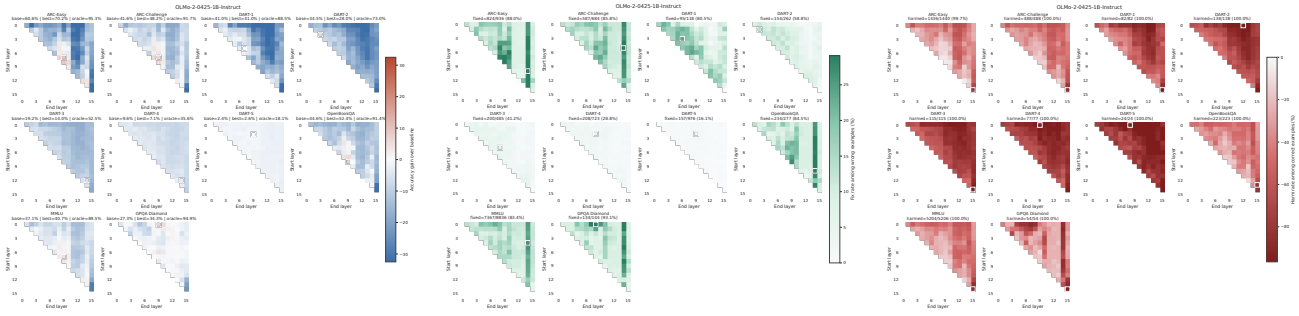
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 9. **Llama-3.2-3B-Instruct.** Full BLR heatmaps for Llama-3.2-3B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



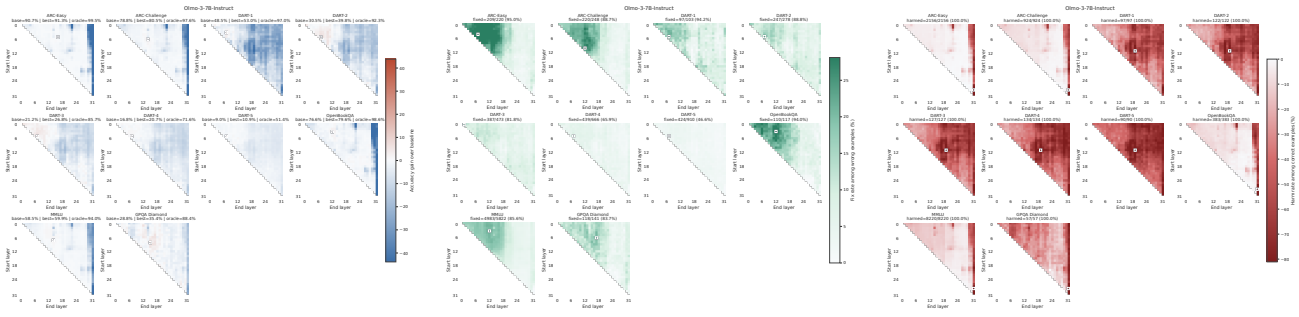
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 10. **OLMo-2-0425-1B.** Full BLR heatmaps for OLMo-2-0425-1B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



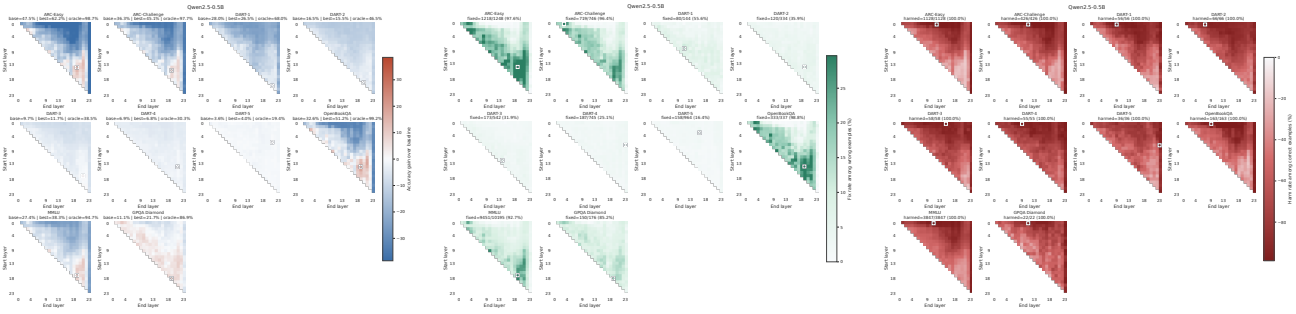
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 11. **OLMo-2-0425-1B-Instruct.** Full BLR heatmaps for OLMo-2-0425-1B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



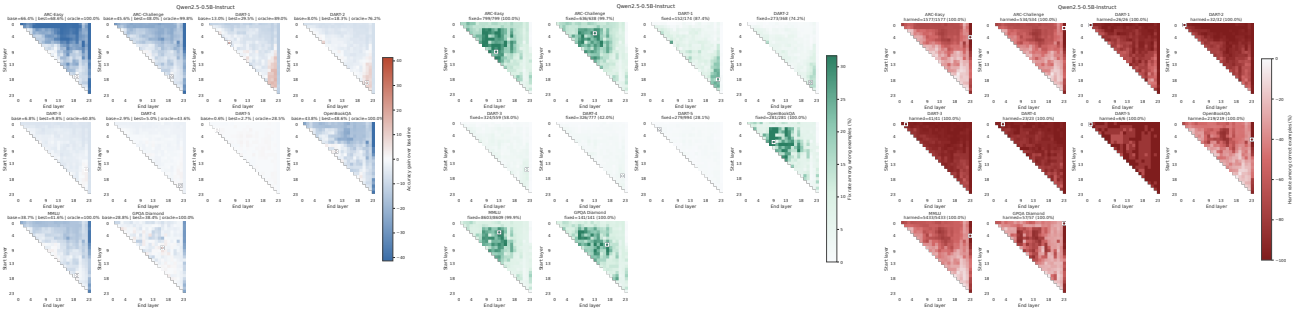
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 12. **OLMo-3-7B-Instruct.** Full BLR heatmaps for OLMo-3-7B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



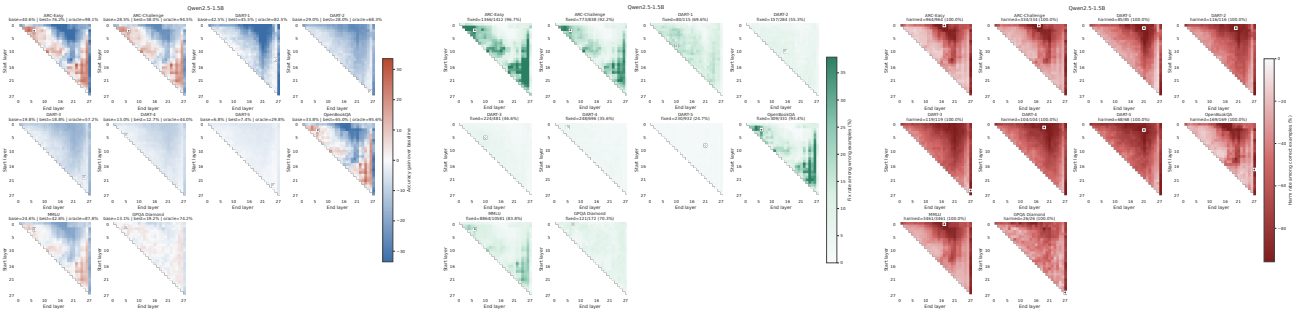
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 13. **Qwen2.5-0.5B.** Full BLR heatmaps for Qwen2.5-0.5B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



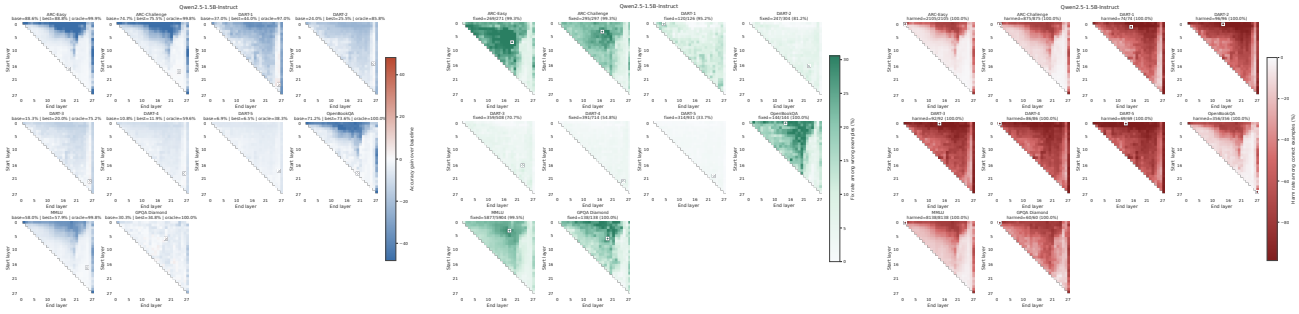
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 14. **Qwen2.5-0.5B-Instruct.** Full BLR heatmaps for Qwen2.5-0.5B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



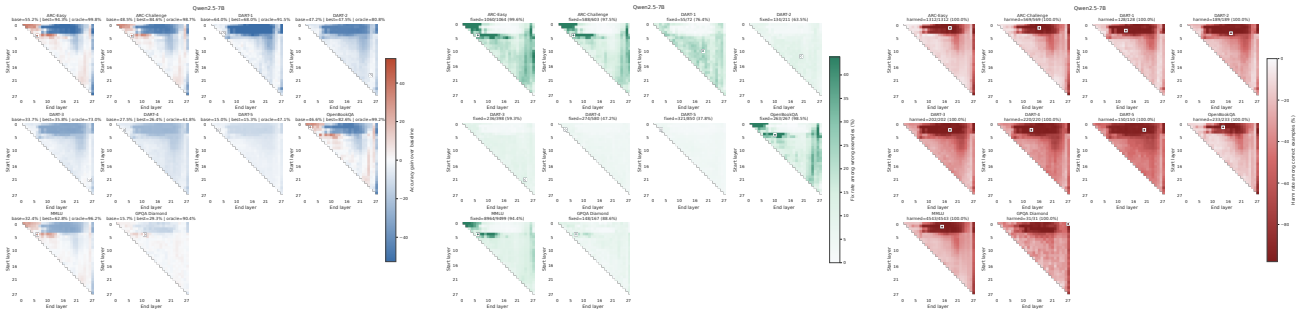
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 15. **Qwen2.5-1.5B.** Full BLR heatmaps for Qwen2.5-1.5B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



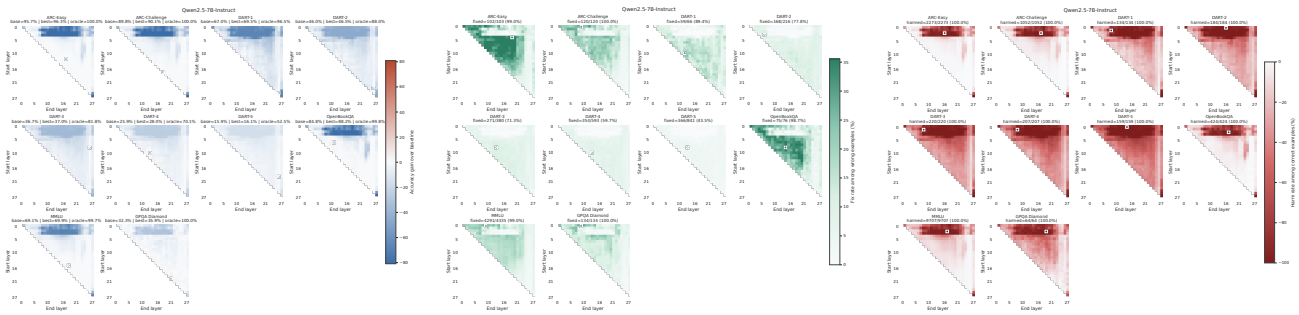
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 16. **Qwen2.5-1.5B-Instruct.** Full BLR heatmaps for Qwen2.5-1.5B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



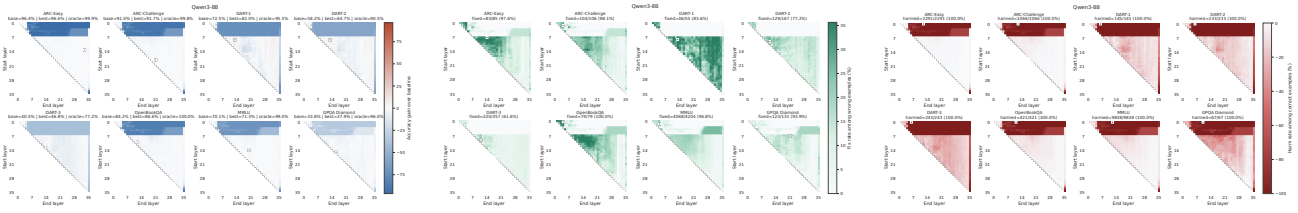
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 17. **Qwen2.5-7B.** Full BLR heatmaps for Qwen2.5-7B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



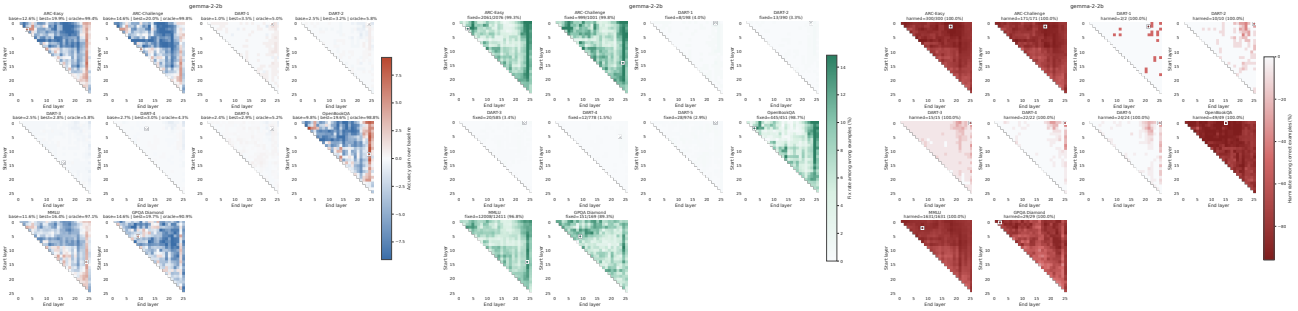
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 18. **Qwen2.5-7B-Instruct.** Full BLR heatmaps for Qwen2.5-7B-Instruct. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



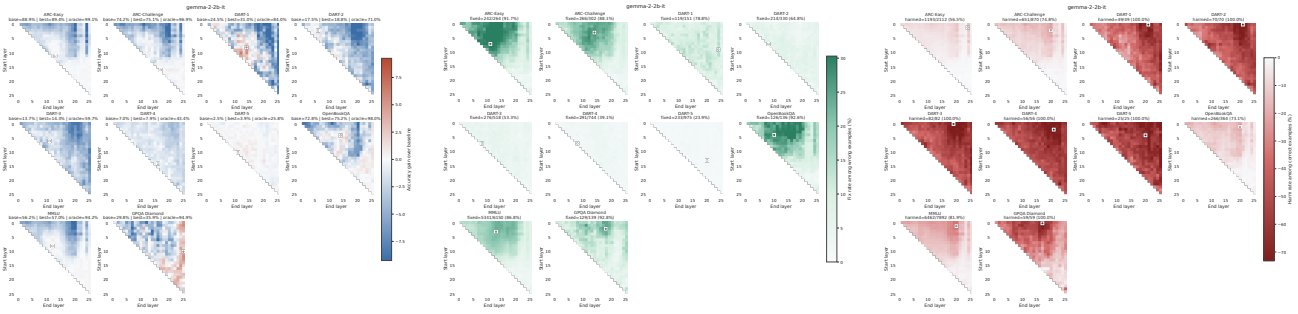
(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 19. **Qwen3-8B.** Full BLR heatmaps for Qwen3-8B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 20. **Gemma-2-2B.** Full BLR heatmaps for Gemma-2-2B. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.



(a) Accuracy gain *(b)* Fix count *(c)* Harm count

Figure 21. **Gemma-2-2B-IT.** Full BLR heatmaps for Gemma-2-2B-IT. Accuracy gain is measured relative to the no-recursion baseline. Fix counts denote baseline-wrong examples corrected by a recursive block. Harm counts denote baseline-correct examples flipped to incorrect by a recursive block.

E. Model Architecture, Implementation Details, and Ablations

E.1. Model Architecture and Implementation Details

Both router variants take hidden states collected during a preliminary pass through $f_{\boldsymbol{\theta}}$ as input. Input tokens are split into W contiguous chunks and mean-pooled within each chunk, producing per-layer sequences of length W (the *pooling window*). We denote the router hidden size by D.

Local router. A two-layer MLP with hidden size D is applied to each pooled hidden state at each layer. At layer ℓ, it predicts action values for (i) continuing on the default route and (ii) jumping back to start a recursive block at some earlier layer $s \leq \ell$. The MLP is applied to each of the W chunks independently and the resulting logits are averaged across chunks. If the best jump action has positive margin over the continue action, the model executes the corresponding BLR route. Otherwise it keeps the default route.

Global router. A two-stage attention-pooling network aggregates evidence from all layers before committing to a route. After the preliminary pass, pooled hidden states form a tensor of shape $L \times W \times d$. The first stage pools across the W chunks within each layer using a single-head self-attention with a learned query; the second stage pools across the L layers using the same mechanism. The resulting vector in $\mathbb{R}^D$ is mapped by a linear projection to scores over $\mathcal{R}_{\text{BLR}}$, and the argmax route is selected.

Local-router training objective. The local router predicts action values at each layer. The valid actions $\mathcal{A}_\ell$ at layer ℓ are continuing on the default route or jumping back to start a recursive block at some earlier layer $s \leq \ell$. Each action $a \in \mathcal{A}_\ell$ induces a route $r(a) \in \mathcal{R}_{\text{BLR}}$ and defines the target $q^*(\boldsymbol{x}, \ell, a) := u(\boldsymbol{x}, r(a); \boldsymbol{\lambda})$. The local router outputs $g_{\boldsymbol{\phi}}(\boldsymbol{x}_\ell) \in \mathbb{R}^{|\mathcal{A}_\ell|}$ and is trained with

$$\mathcal{L}_{\text{local}}(\boldsymbol{\phi}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \sum_{\ell=1}^{L} \| g_{\boldsymbol{\phi}}(\boldsymbol{x}_\ell) - q^*(\boldsymbol{x}, \ell, \cdot) \|_2^2 . \tag{3}$$

At inference, we perform a full preliminary forward pass to compute $g_{\boldsymbol{\phi}}(\boldsymbol{x}_\ell)$ for all layers. We then select the single highest-value action across all (ℓ, a) pairs. This may introduce a recursive block, or recover the default route if no recursive action exceeds the value of continuing at all layers. Only one recursion is executed.

KV cache. `aBLR` selects a route once per input sequence and uses the same route for every generated token, making standard KV caching compatible with routed execution. We treat the routed forward pass as a sequence of *virtual* layer steps and store cache entries by virtual step index. If the route repeats layers 2–3, the next executed step after layer 3 is physical layer 2 again, but it receives a new virtual index and therefore a separate cache slot. The cache is extended to match the routed sequence length, while each virtual step still calls the corresponding physical transformer layer.

Default values. Unless stated otherwise, training uses AdamW with an exponential moving average and a cosine learning-rate schedule. The default pooling window is $W = 32$ (Fig. 22b) and the default hidden size is $D = 256$ (Fig. 22a).

Hardware Details. All experiments were run on internal GPU clusters using either NVIDIA GH200 GPUs with 120GB memory or NVIDIA H100 GPUs with 80GB HBM3 memory, depending on cluster availability. Each experimental run used a single GPU, and we ran one experiment per GPU.

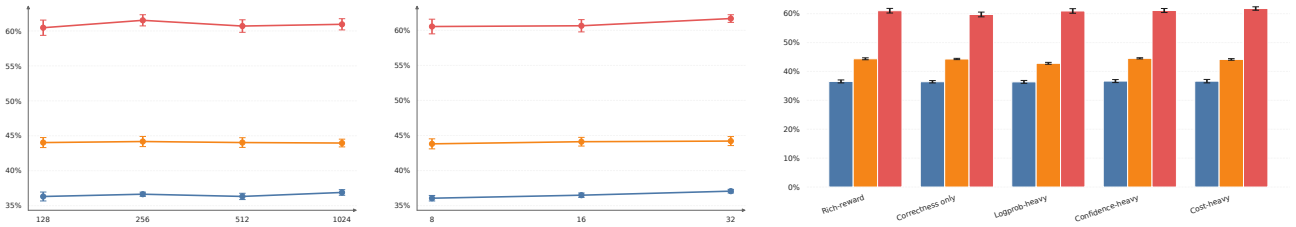
E.2. Ablations

E.2.1. Local Router Ablation

Table 2. **Router-family ablation on Qwen2.5 base models.** Entries are weighted average accuracy over all eight datasets. Unrouted (r_0) denotes the default forward pass without BLR.

Method	0.5B	1.5B	7B
Unrouted (r_0)	27.5	25.6	35.6
Local router (`aBLR`)	34.6 (+7.1)	41.2 (+15.6)	55.3 (+19.7)
Global router (`aBLR`)	**37.2** (**+9.7**)	**44.8** (**+19.3**)	**61.9** (**+26.3**)

Both routers improve over the unrouted baseline (Tab. 2), but the global router is stronger at every scale on the all-dataset weighted average, and the gap widens with scale: $+2.6$ points at 0.5B, $+3.6$ points at 1.5B, and $+6.6$ points at 7B. The



(a) Hidden dimension. *(b)* Window size. *(c)* Reward profile.

Figure 22. **Global router hyperparameter ablations on Qwen2.5 base models.** We vary one router design choice at a time and report all-weighted routed accuracy and standard deviation. Colors denote model scale: ■ Qwen2.5-0.5B, ■ Qwen2.5-1.5B, and ■ Qwen2.5-7B.

global router's two-stage attention pooling therefore makes increasingly better use of available evidence as model depth grows.

E.2.2. Reward Profiles

The scalar route reward in Eq. (1) has three weights $\boldsymbol{\lambda} = (\lambda_{\text{lp}}, \lambda_{\text{conf}}, \lambda_{\text{cost}})$. Table 3 lists the profiles used in the reward ablation of Fig. 22c. The *correctness-only* profile replaces the scalar reward with the binary correctness indicator $\mathbf{1}[\text{correct}(\boldsymbol{x}, r)]$ and ignores $\boldsymbol{\lambda}$.

Table 3. **Reward profiles and their $\boldsymbol{\lambda}$ values.** All profiles except *correctness only* use the full reward in Eq. (1) with indicator weight fixed at one; the table lists the remaining three weights.

Reward profile	λ_{lp}	λ_{conf}	λ_{cost}
Correctness only	0.0	0.0	0.0
Rich-reward	0.2	0.2	0.2
Logprob-heavy	0.5	0.1	0.2
Confidence-heavy	0.1	0.5	0.2
Cost-heavy	0.2	0.2	0.5

E.2.3. Hyperparameter Ablation

We ablate three global router hyperparameters on the Qwen2.5 base models, varying one at a time and reporting all-weighted routed accuracy across ARC, DART, OpenBookQA, GPQA Diamond, and MMLU.

Figure 22 shows that router performance is moderately sensitive to these choices, but no single setting dominates for every model scale. A hidden dimension of 256 gives the best macro average across base models. A pooling window of 32 gives the best routed accuracy at every base scale, raising the macro average to 47.7% versus 47.1% for window size 16 and 46.8% for window size 8. For the reward profile, *confidence-heavy* performs best for Qwen2.5-0.5B and Qwen2.5-1.5B, while *cost-heavy* performs best for Qwen2.5-7B. We use one fixed configuration for all main experiments, chosen by macro-average performance across the Qwen2.5 base models: hidden dimension 256, pooling window 32, and the *confidence-heavy* profile ($\boldsymbol{\lambda} = (0.1, 0.5, 0.2)$).

E.2.4. MCTS vs. Oracle BLR

Both Oracle BLR and MCTS (Li et al., 2025) use gold labels and are therefore upper-bound comparisons, differing in route space and selection procedure: exhaustive enumeration over the BLR family versus heuristic search over the larger per-layer space. Due to compute constraints, MCTS is evaluated only on the in-domain benchmarks. Oracle BLR consistently outperforms MCTS on all three Qwen2.5 base models on ARC and DART, indicating that search difficulty dominates the expressivity advantage of larger action spaces.

E.2.5. LoRA Fine-Tuning Comparison

LoRA and BLR address different problems. LoRA modifies the model by training low-rank adapters on each target task. BLR keeps all weights frozen and only changes which layers execute at inference time. The two methods are therefore not

Table 4. **MCTS vs. Oracle BLR.** Oracle BLR uses a smaller structured route space, but achieves larger in-domain gains than MCTS. Unrouted (r_0) denotes the default forward pass.

Size	Method	ARC	DART	Avg.
0.5B	Unrouted (r_0)	43.8	9.0	27.9
	MCTS (Li et al., 2025)	72.0 (+28.2)	15.5 (+6.5)	46.1 (+18.2)
	Oracle BLR	**98.4 (+54.6)**	**33.0 (+23.9)**	**68.4 (+40.5)**
1.5B	Unrouted (r_0)	36.6	16.4	27.3
	MCTS (Li et al., 2025)	68.6 (+32.0)	26.0 (+9.6)	49.1 (+21.8)
	Oracle BLR	**96.9 (+60.3)**	**47.7 (+31.3)**	**74.3 (+47.0)**
7B	Unrouted (r_0)	53.0	29.6	42.3
	MCTS (Li et al., 2025)	81.7 (+28.7)	42.2 (+12.6)	63.6 (+21.3)
	Oracle BLR	**99.5 (+46.4)**	**63.6 (+34.0)**	**83.0 (+40.7)**

directly comparable. LoRA can adapt the model function to the training tasks, whereas BLR is restricted to computation paths already available in the pretrained model.

We nevertheless include LoRA in Tab. 5 as a parametric adaptation reference. LoRA achieves strong overall accuracy at every scale, while degrading DART accuracy. At the same time, Oracle BLR is consistently higher than LoRA across all scales. This shows that a large gap remains between current methods and the best route available within the frozen BLR route family. The results therefore highlight that pretrained models contain substantial route-dependent potential that is not yet fully exploited.

Table 5. **LoRA fine-tuning on Qwen2.5 base models.** Results use the same evaluation groups as Tab. 1: ARC and DART form the in-domain average, while OpenBookQA, GPQA Diamond, and MMLU form the OOD average. Unrouted (r_0) denotes the default forward pass without BLR. 'LoRA' denotes the same backbone after LoRA fine-tuning. Deltas are absolute accuracy changes relative to Unrouted (r_0) at the same scale. Gray rows show Oracle BLR upper bounds.

Size	Method	In-domain			OOD				Overall
		ARC	DART	Avg.	OBQA	GPQA	MMLU	Avg.	Avg.
0.5B	Unrouted (r_0)	43.8	9.0	27.9	32.6	11.1	27.4	27.4	27.5
	LoRA	68.0 (+24.2)	7.4 (-1.6)	40.2 (+12.3)	54.8 (+22.2)	22.7 (+11.6)	44.5 (+17.1)	44.6 (+17.2)	43.2 (+15.7)
	Oracle BLR	98.4 (+54.6)	33.0 (+23.9)	68.4 (+40.5)	99.2 (+66.6)	86.9 (+75.8)	94.7 (+67.3)	94.7 (+67.4)	86.6 (+59.1)
1.5B	Unrouted (r_0)	36.6	16.4	27.3	33.8	13.1	24.6	24.8	25.6
	LoRA	87.3 (+50.7)	12.1 (-4.3)	52.8 (+25.5)	80.6 (+46.8)	26.3 (+13.2)	60.0 (+35.4)	60.2 (+35.4)	58.0 (+32.4)
	Oracle BLR	96.9 (+60.3)	47.7 (+31.3)	74.3 (+47.0)	95.6 (+61.8)	74.2 (+61.1)	87.8 (+63.1)	87.9 (+63.1)	83.7 (+58.1)
7B	Unrouted (r_0)	53.0	29.6	42.3	46.6	15.7	32.4	32.6	35.6
	LoRA	94.4 (+41.4)	21.6 (-8.0)	61.1 (+18.8)	91.2 (+44.6)	35.9 (+20.2)	71.8 (+39.4)	72.0 (+39.4)	68.6 (+33.0)
	Oracle BLR	99.5 (+46.4)	63.6 (+34.0)	83.0 (+40.7)	99.2 (+52.6)	90.4 (+74.7)	96.2 (+63.8)	96.2 (+63.6)	92.2 (+56.6)

E.2.6. Instruction-Tuned Qwen2.5 Results

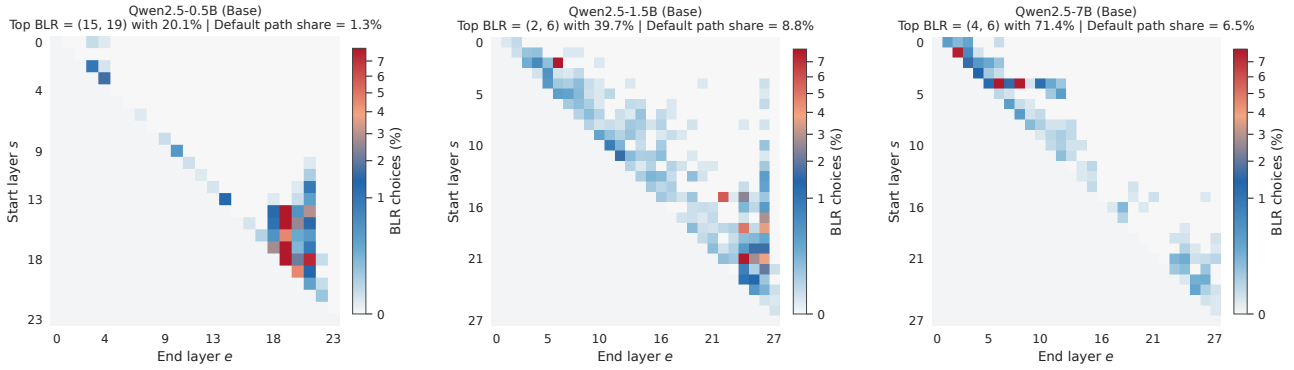
Table 6 repeats the main evaluation on instruction-tuned Qwen2.5 checkpoints. Two findings stand out. First, the instance-level oracle remains very high at every scale (92.8%, 94.4%, and 95.7% overall), substantially exceeding the unrouted baseline (37.3%, 56.1%, 67.8%) and confirming that large input-dependent routing potential persists after instruction tuning. Second, this potential is harder to recover with the current learned router: `aBLR` improves over the unrouted baseline at 0.5B (37.3% $\rightarrow$ 40.4% overall) but matches it at 1.5B and 7B. Closing the oracle gap on instruction-tuned models therefore appears to require routing supervision that better captures the more complex decision boundaries induced by instruction tuning, which we leave to future work.

Table 6. **Instruction-tuned Qwen2.5 results.** Weighted accuracy is reported for in-domain, OOD, and all datasets. Unrouted (r_0) denotes the default forward pass without BLR. Deltas are absolute changes relative to Unrouted (r_0) at the same scale. Gray rows show Oracle BLR upper bounds.

Size	Method	In-domain			OOD				Overall
		ARC	DART	Avg.	OBQA	GPQA	MMLU	Avg.	Avg.
0.5B	Unrouted (r_0)	59.5	4.3	34.2	43.8	**28.8**	38.7	38.7	37.3
	Random BLR	37.8 (-21.7)	2.4 (-1.9)	21.5 (-12.6)	32.8 (-11.0)	**28.8** (+0.0)	29.0 (-9.7)	29.1 (-9.6)	26.8 (-10.6)
	sBLR	**61.8** (+2.3)	2.9 (-1.4)	34.8 (+0.6)	47.8 (+4.0)	26.3 (-2.5)	41.6 (+2.9)	41.6 (+2.9)	39.5 (+2.2)
	aBLR	61.6 (+2.1)	**7.7** (+3.4)	**36.9** (+2.7)	**49.0** (+5.2)	26.3 (-2.5)	**42.0** (+3.3)	**42.0** (+3.3)	**40.4** (+3.1)
	Oracle BLR	99.9 (+40.4)	49.4 (+45.1)	76.8 (+42.6)	100.0 (+56.2)	100.0 (+71.2)	100.0 (+61.3)	100.0 (+61.2)	92.8 (+55.5)
1.5B	Unrouted (r_0)	**84.0**	13.9	**51.9**	**71.2**	30.3	**58.0**	**58.0**	**56.1**
	Random BLR	66.5 (-17.5)	6.5 (-7.4)	39.0 (-12.9)	56.0 (-15.2)	26.3 (-4.0)	47.2 (-10.8)	47.2 (-10.8)	44.7 (-11.5)
	sBLR	83.8 (-0.2)	13.7 (-0.2)	51.6 (-0.2)	71.0 (-0.2)	28.3 (-2.0)	57.3 (-0.7)	57.3 (-0.7)	55.6 (-0.6)
	aBLR	83.0 (-1.0)	**15.0** (+1.1)	51.8 (-0.0)	71.0 (-0.2)	**30.8** (+0.5)	57.3 (-0.7)	57.4 (-0.6)	55.7 (-0.4)
	Oracle BLR	99.9 (+15.9)	61.6 (+47.7)	82.3 (+30.5)	100.0 (+28.8)	100.0 (+69.7)	99.8 (+41.9)	99.8 (+41.8)	94.4 (+38.3)
7B	Unrouted (r_0)	**93.7**	**30.1**	**64.6**	84.8	**32.3**	69.1	69.2	**67.8**
	Random BLR	77.4 (-16.3)	17.8 (-12.3)	50.1 (-14.5)	70.6 (-14.2)	28.8 (-3.5)	56.7 (-12.4)	56.8 (-12.4)	54.7 (-13.0)
	sBLR	**93.7** (-0.0)	29.8 (-0.4)	64.4 (-0.2)	**85.0** (+0.2)	**32.3** (+0.0)	69.0 (-0.1)	69.0 (-0.1)	67.6 (-0.2)
	aBLR	93.5 (-0.2)	28.8 (-1.3)	63.9 (-0.7)	84.8 (+0.0)	**32.3** (+0.0)	**69.4** (+0.2)	**69.4** (+0.2)	67.7 (-0.1)
	Oracle BLR	100.0 (+6.3)	70.7 (+40.6)	86.6 (+22.0)	99.8 (+15.0)	100.0 (+67.7)	99.7 (+30.6)	99.7 (+30.5)	95.7 (+27.9)

F. Qualitative Analysis

F.1. Router Outputs



(a) Qwen2.5-0.5B *(b)* Qwen2.5-1.5B *(c)* Qwen2.5-7B

Figure 23. **aBLR choices across model scales.** The router assigns probability mass to multiple BLR blocks rather than collapsing to a single route.

Fig. 23 show that aBLR does not collapse to a single route across model scales. The share of the most selected BLR block increases with model size, from 20.1% at 0.5B to 71.4% at 7B, but the router still assigns meaningful mass to several alternative blocks, and the default route remains rare. This shows that routing does not converge to one fixed computation path, but instead uses a set of structured, input-dependent routes. The fact that some blocks are selected much more often than others also argues against a noise-based explanation, since the distribution is not uniform but concentrated on specific parts of the network.

F.2. Gain Decomposition

Table 7. **aBLR gain decomposition on Qwen2.5 base models.** FR is the fraction of the gain explained by format repair. Remaining is the gain after removing FR. SF is the percentage of Remaining explained by semantic fixes.

Size	Gain	FR	Remaining	SFs
0.5B	+9.7	45%	+5.3	94%
1.5B	+19.3	40%	+11.6	94%
7B	+26.3	27%	+19.1	92%

Format Repairs (FR). On base models, the default route often produces a plausible explanation without emitting a parsable answer, while a recursive route yields the correct format and option. Table 7 shows that augmenting the baseline parser with a strict numeric fallback ($1 \mapsto A, \ldots, 4 \mapsto D$) recovers 27–45% of the aBLR gain across scales. Format repair accounts for less than half of the gain, and its share decreases as larger models more reliably emit the expected format.

Semantic Fixes (SF). The remaining 55–73% of the gain comes from routes that change the predicted answer itself. Table 7 shows that in 92–94% of these cases, the baseline selects an interpretable but incorrect option and aBLR replaces it with the correct one. aBLR therefore changes both the output format and the semantic decision. Concrete examples appear in App. F.3.

F.3. LLM Generation

This section provides representative generations for the format repair and semantic fix analysis in Sec. 4.4.

Format repair on MMLU. Examples 1 and 2 illustrate format repair on Qwen2.5-1.5B: the baseline produces a plausible explanation but no parsable answer, sBLR enforces the format but selects the wrong option, and aBLR produces both the correct format and the correct answer.

Example 1 (MMLU 227). *Where are the vital centres located in the brainstem?* Gold: **C**

Baseline — *unparsable*

```
1.  The vital centers are located in the brainstem, specifically in the
med
```

sBLR (layers 2–6) — *wrong*

```
B. pons.  The pons is located at the base of the brain
```

aBLR (layers 21–25) — *correct*

```
C. medulla oblongata.  The medulla oblongata is located
```

Example 2 (MMLU 544). *The cardiac cycle consists of the phases:* Gold: **C**

Baseline — *unparsable*

```
1.  The cardiac cycle consists of the phases:  systole, diastole
```

sBLR (layers 2–6) — *wrong*

```
B. The cardiac cycle consists of the phases:  contraction, relaxation, and
rest
```

aBLR (layers 18–24) — *correct*

```
C. diastole and systole.  The cardiac cycle consists of two phases
```

Semantic fixes on DART. Examples 3 and 4 illustrate semantic fixes on Qwen2.5-7B, where all methods produce parsable outputs and routing changes the predicted answer itself.

Example 3 (DART-1 48). *Find the value of the first term in the geometric sequence* $a, b, c, 32, 64$. Gold: 4

Baseline

```
2
```

sBLR (layers 1–2)

```
2
```

aBLR (layers 9–12)

```
4
```

Example 4 (DART-1 91). *Solve* $\frac{x+4}{x-2} = 3$ *for* x. Gold: 5

Baseline

```
10
```

sBLR (layers 1–2)

```
10
```

aBLR (layers 3–4)

```
5
```

G. Answer Extraction Rules

Accuracy is computed by extracting a predicted answer from the model's free-form output and comparing it to the ground-truth label. We consider two types of benchmarks: multiple-choice questions (MCQ) and numeric reasoning tasks.

G.1. Multiple-Choice Extraction

We evaluate on ARC-Easy, ARC-Challenge, MMLU, GPQA Diamond, and OpenBookQA. Each question has a fixed set of lettered choices (A, B, C, ...). We extract the predicted letter using the following procedure.

Step 1: Pattern matching. We apply regular-expression patterns for answer phrases such as `Answer: X`, `Correct answer is X`, standalone options (e.g. `(B)`, `C.`), and `Option X`. If multiple matches occur, we select the *last* occurrence in the output.

Step 2: Fallback matching. If Step 1 fails, we search for a standalone answer letter anywhere in the output. We restrict matches to valid answer options and prefer occurrences near the end of the output or following answer-indicating phrases.

Step 3: Choice-text overlap. If no letter is found, we match the output against the choice texts using normalized token overlap and accept a match only if a single choice exceeds a threshold of 0.8.

A prediction is correct if the extracted letter matches the ground-truth.

G.2. Numeric Extraction

For DART benchmarks, we extract numeric answers using the following deterministic procedure.

Boxed content. If present, we extract the content inside `\boxed{...}` as the final answer. Otherwise, the output is considered unparsable and counted as incorrect.