# BLOCK-LEVEL RECURSION: ADAPTIVE TEST-TIME ROUTING IN LARGE LANGUAGE MODELS

**Anonymous authors**
Paper under double-blind review

## 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 yields large accuracy gains while keeping route selection practical. Across six model families and ten reasoning benchmarks, the optimal block varies across models, tasks, and individual inputs, with per-instance oracle gains of $+59.1\%$ overall 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 improves accuracy substantially, 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

LLM accuracy scales with model size (Kaplan et al., 2020; Brown et al., 2020) and can also improve with longer generation through chain-of-thought (CoT) prompting (Wei et al., 2022) and reinforcement-trained reasoning traces (Guo et al., 2025). Both come at a cost: larger models require more memory and compute, while longer generations increase latency and token budget. A complementary approach changes which layers execute in a *fixed* model at inference time, without modifying weights or extending the output sequence. Layer-skipping methods reduce compute (Elhoushi et al., 2024; Luo et al., 2025a;b; Chen et al., 2025a), while layer-repetition methods can improve accuracy (Li et al., 2025; Heakl et al., 2026). Existing routing methods make layer- or block-level skip, execute, or repeat decisions, yielding spaces that can grow exponentially with depth. Three independent actions across $L$ layers already give $3^L$ assignments, making exhaustive exploration impractical and requiring heuristic search (Heakl et al., 2026). CoLa (Li et al., 2025) expands this space further by also allowing short contiguous blocks to be repeated, making it even harder to explore. This motivates a smaller route family defined directly over contiguous blocks.

Recent recurrent-depth work suggests that pretrained layers form functional stages and that repeating subsets of layers can improve reasoning (Koishekenov et al., 2026; McLeish et al., 2025). We therefore introduce **Block-Level Recursion** (BLR), which repeats a contiguous block of transformer layers once. For $L$ layers, BLR contains $L(L-1)/2$ recursive routes plus the default path, giving only 277 routes for a 24-layer model instead of $3^{24} \approx 282$ billion per-layer assignments. This compact space makes exhaustive per-instance evaluation practical, provides dense supervision for reward regression, and lets us measure the oracle gap directly. Compared with CoT (Wei et al., 2022) and looped latent reasoning (Saunshi et al., 2025; Zhu et al., 2025), BLR adds computation inside the frozen backbone and without additional tokens. Its input-specific oracle gap also mirrors input-dependent depth allocation in adaptive looped models (Zhu et al., 2025), motivating per-input route selection.

Across Qwen2.5 (Qwen Team, 2025a) and five additional model families, the best block to repeat varies across model scales, tasks, and individual inputs. Since gold answers are not available at



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.

test time, we introduce two practical route-selection policies. **Static `BLR` (`sBLR`)** applies one train-selected block per backbone, providing a fixed-route baseline. Building on the observed variation across inputs, **Adaptive `BLR` (`aBLR`)** uses a lightweight global router to select a route per input from frozen hidden states. In practice, `sBLR` already yields substantial gains, while `aBLR` improves further: with Qwen2.5-0.5B, it exceeds the overall accuracy of the unrouted Qwen2.5-7B model with $14\times$ fewer parameters and lower FLOPs.

We summarize our contributions as follows: (1) **Block-Level Recursion.** We introduce a structured, exhaustively enumerable family of inference-time routes over frozen pretrained LLMs that makes exhaustive per-instance evaluation practical. (2) **Oracle `BLR` upper bound.** This tractability lets us directly measure the best accuracy achievable within the `BLR` route family under gold-answer selection. This optimistic upper bound reveals high route-set coverage, with gains up to $+75.8\%$ across multiple model families and diverse tasks. (3) `sBLR` **and** `aBLR`. To select routes without gold answers, we introduce two practical policies. A single train-selected block substantially improves accuracy, while a lightweight learned router selects routes per input and improves further. We plan to release the per-instance per-route reward dataset to support further research on routing supervision in frozen models.

## 2 Related Work

**Routing and layer reuse in pretrained LLMs.** Several recent methods modify execution paths in frozen transformers. CoLa (Li et al., 2025) uses Monte Carlo tree search (MCTS) to compose per-sample edits of the layer sequence, where each action skips $k$ layers or repeats a block of $k$ layers $r$ times with $k, r \in \{1, 2, 3, 4\}$, while Heakl et al. (2026) learn a per-layer policy that decides whether to skip, execute, or repeat each layer at inference time. A parallel line of work focuses on skipping layers to reduce compute (Luo et al., 2025a;b; Chen et al., 2025a), trains the backbone for early exit (Elhoushi et al., 2024), or identifies redundancy in transformer depth (Men et al., 2024). These methods either perform online search or take local, per-layer decisions over large action spaces.

**Adaptive computation and internal recurrence.** Adaptive-computation methods vary depth or halting time per input, including Adaptive Computation Time (Graves, 2016), PonderNet (Banino et al., 2021), depth-adaptive transformers (Elbayad et al., 2020), early exit (Xin et al., 2020; Liu et al., 2020; Chen et al., 2024; Shan et al., 2024), confidence-based decoding (Schuster et al., 2022), and dynamic width or depth allocation (Hou et al., 2020; Raposo et al., 2024). These methods mainly reduce computation by halting or skipping. A complementary line increases internal computation through recurrent or latent reasoning, including Universal Transformers (Dehghani et al., 2019), pause tokens (Goyal et al., 2024), self-generated reasoning traces (Zelikman et al., 2024), latent reasoning (Hao et al., 2025; Chen et al., 2025b), and recurrent-depth architectures (McLeish et al., 2025; Geiping et al., 2025; Koishekenov et al., 2026). These approaches show that additional internal computation can improve reasoning, but typically require training, retrofitting, or architectural changes rather than routing through an unchanged pretrained layer graph.

**Block-Level Recursion.** Prior work spends additional computation through output tokens in 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 search over layer and block edits (Li et al., 2025; Heakl et al., 2026). `BLR` instead re-executes one contiguous block of frozen layers, requiring no continued backbone training, architectural modification, or additional tokens. Rather than making independent per-layer decisions or searching bounded edit operations, it explicitly defines all single-repeat contiguous blocks, yielding only $O(L^2)$ candidates. This makes exhaustive evaluation practical and allows the gains from additional internal computation to be measured and learned directly.

## 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.

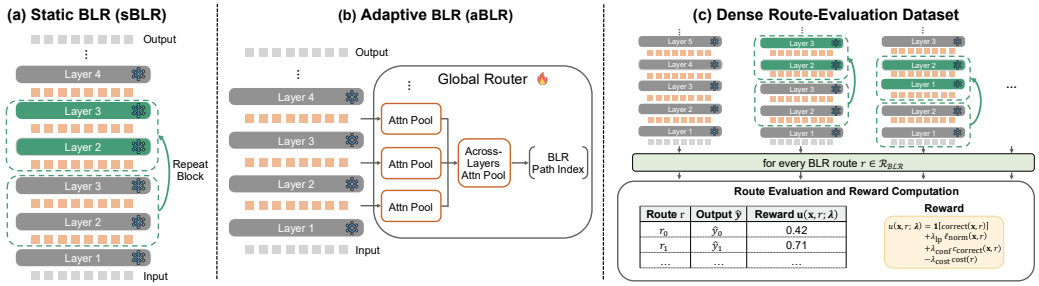



Figure 1: **Block-Level Recursion.** BLR changes the inference-time route through a frozen transformer without modifying weights. **(a)** Static BLR (sBLR) repeats one fixed contiguous block of layers for all inputs. **(b)** Adaptive BLR (aBLR) adds a lightweight global router that selects the block per input from pooled hidden states collected during a preliminary pass. **(c)** For each training input, every route $r \in \mathcal{R}_{\text{BLR}}$ is executed once on the frozen backbone to record outputs and a scalar reward defined in Sec. 3.6, yielding dense per-route supervision used to train aBLR.

### 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 $\mathcal{V}$ denote the vocabulary, $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$ the token embedding matrix, and $\boldsymbol{W} \in \mathbb{R}^{d \times |\mathcal{V}|}$ the output projection. The standard forward pass computes hidden representations

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

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

**Definition 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 $T = |r| \geq 1$ counts layer executions. Routes may revisit or skip layers depending on the sequence of indices. The* admissible route set $\mathcal{R} \subseteq \bigcup_{T \geq 1} \{1, \ldots, L\}^T$ *is a finite collection of allowed routes. The index $t$ counts execution steps, while $\ell_t$ denotes the physical layer called at step $t$. Route execution computes*

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

*with logits $\boldsymbol{z}_{\boldsymbol{\theta}}(\boldsymbol{x}; r) := \boldsymbol{H}_T(\boldsymbol{x}; r)\boldsymbol{W}$, 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 $r_0 \in \mathcal{R}$.*

**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, $\mathcal{R}_{\text{skip}} = \big\{r \in \{1, \ldots, L\}^T : 1 \leq T \leq L,\ \ell_i < \ell_j \text{ for all } i < j\big\}$, so every admissible route only removes computation, giving $2^L$ subsequences (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 (for example, $3^L$ assignments for three independent actions per layer) (Heakl et al., 2026). BLR restricts this design space to a single extra traversal of one contiguous block, making the complete family enumerable.

### 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 $1 \leq s < e \leq L$, the recursive route $r_{s,e}$ inserts one additional pass over layers $s$ through $e$:

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

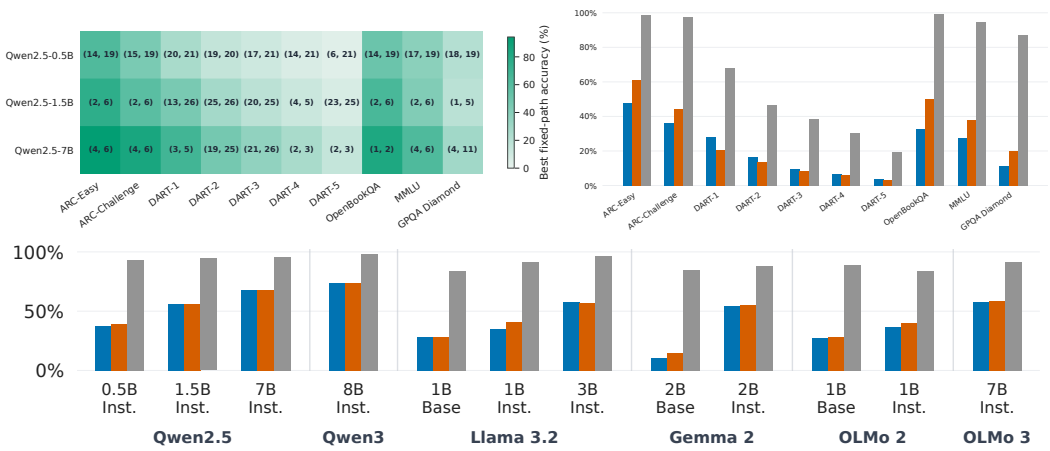



Figure 2: **Best BLR varies across tasks, inputs, and model families.** ■ baseline, ■ sBLR, and ■ oracle. Task specific BLR (top-left). For a fixed model, the optimal BLR block changes across benchmarks. Each cell reports the best $(s, e)$ pair found by exhaustive search. Instance-level oracle BLR for Qwen2.5-0.5B (top-right). The oracle exceeds both the baseline and sBLR, showing large input-dependent gains. Generality across model families (bottom). The same pattern holds across model families, with consistent gains from BLR and a large oracle gap, averaged over all datasets.

We refer to $(s, e)$ as the *BLR block*. The endpoints are inclusive, with $s < e$, and the route executes $|r_{s,e}| = L + e - s + 1 \leq 2L$ layers in total. The admissible route set is then

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

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 24-layer model), making exhaustive per-instance evaluation practical. BLR thus introduces a single interpretable control choice: which contiguous block to repeat.

### 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 propose two policies that share the same route set and frozen backbone.

**Static BLR.** Let $\mathcal{D}$ denote the labeled training set. 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 route selection 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. The repeated block still adds layer computation.

**Adaptive BLR.** aBLR selects the route separately for each input via a router $g_{\boldsymbol{\phi}}$ with trainable parameters $\boldsymbol{\phi}$ that maps pooled prompt representations 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 $u(\boldsymbol{x}, \cdot; \boldsymbol{\lambda}) \in \mathbb{R}^{|\mathcal{R}_{\texttt{BLR}}|}$.

Route rewards are computed independently per input, with normalization performed across routes for the same example. 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 prompt hidden states without generating answer tokens. One router call then selects a route, and the model runs the prompt again from layer 1 before decoding with that route. This is compatible with standard KV caching, as the selected route is fixed for the entire sequence.

To keep the router lightweight, hidden states are compressed before routing. Input tokens are partitioned into $W$ contiguous, non-overlapping chunks of $\lfloor T/W \rfloor$ tokens each, and valid tokens within each chunk are mean-pooled, yielding $W$ vectors per layer. The router $g_{\boldsymbol{\phi}}$ aggregates these representations in two stages, first across chunks within each layer and then across layers, and maps the resulting vector to scores over $\mathcal{R}_{\texttt{BLR}}$, from which the highest-scoring route is selected (Fig. 1b). We provide additional details in App. C and further study a *local* router inspired by prior per-layer routing methods in App. C.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 each training example $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$ and route $r \in \mathcal{R}_{\texttt{BLR}}$, we define a scalar reward that combines answer correctness, reference likelihood, confidence, and computational cost:

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

where $\boldsymbol{\lambda} := (\lambda_{\mathrm{lp}}, \lambda_{\mathrm{conf}}, \lambda_{\mathrm{cost}}) \in \mathbb{R}^3_{\geq 0}$. Correctness is determined using the benchmark-specific answer extraction and grading rules. The term $\ell_{\mathrm{norm}}(\boldsymbol{x}, r) \in [0, 1]$ is the mean reference-token log probability normalized across routes for the same example (see App. B.3 for details), while $c_{\mathrm{correct}}(\boldsymbol{x}, r) \in [0, 1]$ measures confidence along the reference response and is set to zero for incorrect routes. Finally, $\mathrm{cost}(r) = |r|/(2L)$ penalizes longer routes. Thus, $\ell_{\mathrm{norm}}$ distinguishes routes by their likelihood of the reference answer, while $c_{\mathrm{correct}}$ further separates correct but uncertain routes from confident ones. Together, these terms define the target vector $u(\boldsymbol{x}, \cdot; \boldsymbol{\lambda}) \in \mathbb{R}^{|\mathcal{R}_{\texttt{BLR}}|}$ for each input.

**Reward regression.** A single best route label discards the relative values of other 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 $r$, and the mean squared error (MSE) loss directly optimizes this supervised approximation. The global router predicts $g_{\boldsymbol{\phi}}(\boldsymbol{x}) \in \mathbb{R}^{|\mathcal{R}_{\texttt{BLR}}|}$ and is trained with the following loss, averaged over examples and routes:

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

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

## 4 Experiments

We first characterize `BLR` empirically at the model-, task-, and instance-level (Sec. 4.2) to characterize how successful routes vary across inputs. We then train a router (Sec. 4.3) and measure the accuracy achieved by a learned policy in practice, including quantitative results, efficiency, 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. A, for multiple-choice and numeric answers respectively (Apps. A.1 and A.2).

**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). Oracle BLR selects the best route per example using the gold answer, giving an upper bound over $\mathcal{R}_{\text{BLR}}$ (App. B). We compare to Dr.LLM (Heakl et al., 2026), a state-of-the-art learned routing approach for frozen LLMs, and to CoLa (MCTS) (Li et al., 2025), which provides a finite gold-guided search comparison in the broader CoLa route space. We do not include layer-skipping methods in the main comparison because they target efficiency rather than accuracy, or CoT because it extends the output sequence.

**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. Hyperparameters are selected using only ARC and DART training data, with no test or OOD data used for selection. We use the same configuration across all Qwen2.5 base models and determine the training duration by early stopping with patience 2 on a held-out validation split before retraining on the full training set (see App. C.2.3). The final configuration uses attention dimension 256, pooling count 32, and a confidence-heavy reward profile $\boldsymbol{\lambda} = (0.1, 0.5, 0.2)$ (see App. C.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 oracle upper bound 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

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 | 43.8 (+0.0) | **10.8 (+1.8)** | 28.7 (+0.8) | 32.8 (+0.2) | 11.6 (+0.5) | 27.4 (+0.0) | 27.4 (+0.0) | 27.8 (+0.3) |
| | sBLR | 55.6 (+11.8) | 7.5 (-1.5) | 33.6 (+5.7) | **50.2 (+17.6)** | **20.2 (+9.1)** | 37.9 (+10.5) | 38.1 (+10.7) | 36.7 (+9.2) |
| | aBLR | **56.0 (+12.2)** | 8.1 (-0.9) | **34.1 (+6.2)** | 49.2 (+16.6) | **20.2 (+9.1)** | **38.0 (+10.6)** | **38.2 (+10.8)** | **36.9 (+9.4)** |
| | 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 | 36.6 (+0.0) | **18.9 (+2.5)** | 28.5 (+1.2) | 33.8 (+0.0) | 13.1 (+0.0) | 24.6 (+0.0) | 24.8 (+0.0) | 25.9 (+0.3) |
| | sBLR | 70.2 (+33.6) | 13.0 (-3.4) | 44.0 (+16.7) | 65.0 (+31.2) | **17.2 (+4.1)** | 42.6 (+18.0) | 43.0 (+18.2) | 43.3 (+17.7) |
| | aBLR | **72.3 (+35.7)** | 15.3 (-1.1) | **46.2 (+18.9)** | **65.2 (+31.4)** | 16.2 (+3.1) | **43.4 (+18.8)** | **43.8 (+19.0)** | **44.5 (+18.9)** |
| | 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 | 55.7 (+2.7) | **32.2 (+2.6)** | 44.9 (+2.6) | 48.4 (+1.8) | 15.7 (+0.0) | 34.6 (+2.2) | 34.8 (+2.2) | 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) |

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 is 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. The resulting accuracy substantially exceeds both the unrouted baseline and sBLR (see Fig. 2 and Tab. 1). For Qwen2.5-0.5B, Oracle BLR reaches 86.6% overall compared with 27.5% for the unrouted baseline, with the largest single-task gain reaching +75.8 points on GPQA Diamond. It is important to note that these values indicate to some degree an optimistic upper bound. Because the oracle selects among many routes using the gold answer, some routes will produce correct answers by chance rather than through better computation. We examine this effect further in App. D.3 using matched stochastic-generation and majority-voting controls. Even so, the results show that most BLR routes provide complementary correct outputs across inputs containing useable learning signal, motivating aBLR to learn to select routes without access to the reference answer.

**Generality across Model Families.** The three findings above use Qwen2.5 as the main model family. To test whether the same structure appears beyond this family, we repeat the exhaustive BLR analysis on additional base and instruction-tuned checkpoints, 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). Figure 2 shows the same pattern across all models. sBLR improves over the baseline, while the oracle remains substantially higher. The best BLR block varies across checkpoints, and the gap between sBLR and the oracle indicates strong input dependence across architectures. These results show that BLR is not specific to Qwen2.5. We report further dataset-level results in App. D, with per-dataset figures in App. D.1 and per-model gain heatmaps in App. D.2.

### 4.3 Adaptive BLR and Its Generalization

Findings 1–3 establish that successful routes vary across inputs, while the oracle upper bound remains well above sBLR accuracy. We now train a router $g_\phi$ to implement aBLR, selecting routes from $\mathcal{R}_{\text{BLR}}$ per input without oracle access. Appendix E reports additional results on DART levels, instruction-tuned checkpoints, cross-format transfer, and repeated blocks.

Across all ten datasets, aBLR achieves the best non-oracle overall average at every scale (see Tab. 1): 36.9% at 0.5B, 44.5% at 1.5B, and 61.9% at 7B. It substantially outperforms the unrouted baseline and Dr.LLM, while providing additional gains over sBLR of 0.2, 1.2, and 3.6 points, respectively. Results on instruction-tuned checkpoints, where these gains are less consistent, are reported in App. E.2.

**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 at all three model sizes on the in-domain average. While the selected Qwen static blocks reduce DART accuracy, useful BLR routes remain. For Qwen2.5-1.5B, the post hoc best block improves DART-1 from 42.5% to 45.5%, while sBLR improves all five DART levels on OLMo-3-7B-Instruct by 1.9–9.3 points. Appendix E.1 and Tab. 14 provides the full per-level analysis.

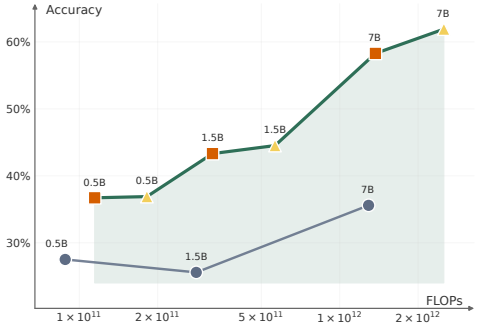



Figure 3: **Accuracy–FLOPs tradeoff on all selected datasets.** ● baseline (unrouted), ■ sBLR, and ▲ aBLR. Routed models extend the Pareto frontier by reaching higher accuracy at competitive compute. FLOPs are shown on a logarithmic scale.

**OOD Generalization.** We next test whether these gains transfer to datasets unseen during router training. Table 1 shows that BLR improves the OOD average more consistently than Dr.LLM. aBLR improves over sBLR at all three scales (38.1% → 38.2%, 43.0% → 43.8%, and 57.0% → 61.9%). Cross-format transfer is more limited and depends on direction and model scale (App. E.3).

**Efficiency Analysis.** sBLR increases computation by re-executing a block of layers, while aBLR adds further cost through the router and a preliminary pass used for route selection. To evaluate these effects, Fig. 3 shows the accuracy–FLOPs tradeoff across all datasets. Notably, aBLR with Qwen2.5-0.5B exceeds the accuracy of the unrouted Qwen2.5-7B model while using substantially fewer FLOPs. More broadly, BLR extends the accuracy–FLOPs frontier across the Qwen2.5 family. Full efficiency results and comparisons to baselines are reported in App. F.

**Comparing BLR to Larger Search Spaces.** We compare the compact BLR family with the broader search space of CoLa (MCTS) (Li et al., 2025). We first evaluate a restricted CoLa variant with $k = \{1\}$ using CoLa's official budget of 200 simulations. This variant allows single-layer skip, execute, or repeat actions. Oracle BLR substantially outperforms this setting at all three scales, showing that the much smaller contiguous-block family already exposes strong alternative routes. For a fair comparison with full CoLa, we then use $k \in \{1, 2, 3, 4\}$ and match its simulation budget to the number of BLR routes, with 277 simulations for Qwen2.5-0.5B and 379 for 1.5B and 7B. Full CoLa improves accuracy further, but Oracle BLR still retains 89–93% of its gain over the unrouted baseline. Thus, BLR captures most of the gain found by the broader search while remaining small enough to enumerate exhaustively and provide complete supervision over every route rather than a search-dependent subset, making policy learning substantially more tractable.

Table 2: **CoLa vs. Oracle BLR.** CoLa ($k = \{1\}$) uses single-layer repetition with 200 MCTS simulations. Full CoLa uses $k \in \{1, 2, 3, 4\}$ with budgets matched to the BLR route count: 277 for 0.5B and 379 for 1.5B and 7B.

| Size | Method | ARC | DART | Avg. |
|---|---|---|---|---|
| 0.5B | Unrouted ($r_0$) | 43.8 | 9.0 | 27.9 |
| | CoLa ($k = \{1\}$) | 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) |
| | CoLa | **100.0** (+56.2) | **41.8** (+32.8) | **73.3** (+45.5) |
| 1.5B | Unrouted ($r_0$) | 36.6 | 16.4 | 27.3 |
| | CoLa ($k = \{1\}$) | 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) |
| | CoLa | **99.8** (+63.2) | **55.1** (+38.7) | **79.3** (+52.0) |
| 7B | Unrouted ($r_0$) | 53.0 | 29.6 | 42.3 |
| | CoLa ($k = \{1\}$) | 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) |
| | CoLa | **99.9** (+46.9) | **69.8** (+40.2) | **86.1** (+43.8) |

### 4.4 Qualitative Analysis

We next analyze how aBLR allocates routes across inputs and how these choices change model outputs.

**aBLR routes.** Figure 4 and App. G.1 suggest that aBLR does not collapse to one route. The router repeatedly selects several structured BLR blocks, rather than sampling the route space uniformly. This shows that the learned policy uses multiple structured, input-dependent routes rather than collapsing to a single fixed path.

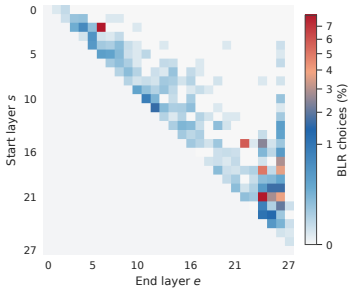



Figure 4: **aBLR block frequency for Qwen2.5-1.5B.** Fraction of inputs assigned to each BLR block.

**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 3 shows that augmenting the baseline parser with a numeric fallback ($1 \mapsto A, \ldots, 4 \mapsto D$) recovers 27–45% of the aBLR gain across scales. Still, 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 3 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. Additional extraction controls and representative generations are provided in Apps. A.3 and G.2.

## 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 captures much of the practical accuracy gain, while a learned global router (aBLR) improves further by selecting routes per input, improving both in-domain and out-of-distribution accuracy and shifting the accuracy–compute frontier of frozen Qwen2.5 backbones.

Table 3: **aBLR gain decomposition on Qwen2.5 base models.** Rem. is the gain after removing FR. SF is the percentage of Rem. explained by semantic fixes.

| Size | Gain | FR | Rem. | SF |
|---|---|---|---|---|
| 0.5B | +9.4 | 45% | +5.1 | 94% |
| 1.5B | +18.9 | 41% | +11.1 | 94% |
| 7B | +26.3 | 27% | +19.1 | 92% |

These results position BLR as a complementary axis of test-time scaling. CoT methods extend computation through additional tokens, whereas BLR increases internal computation within the frozen backbone by deviating from the default linear execution path. The large instance-level oracle gap shows that different inputs are solved by different routes, while learning to select them without labels remains a separate challenge. This aligns with recent findings on recurrent and latent reasoning, and motivates learning which computation to reuse for each input.

**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 (Hu et al., 2022). In addition, BLR is restricted to a single repeated block, which enables exhaustive evaluation but limits the expressivity of admissible routes.

**Future work.** Future work could explore more expressive route families that retain BLR's tractability, including repeating multiple distinct blocks or other structures that remain small enough for exhaustive evaluation. Our results also raise the question of *routability*: how models can be trained so that useful alternative computation paths emerge more reliably. More broadly, routing could be integrated directly into model training so that representations become better suited to input-dependent execution.

## Reproducibility Statement

All datasets and backbone checkpoints used in this work are publicly available. The paper and appendix specify the BLR route construction, router architecture, reward and training objectives, data splits, hyperparameter selection, evaluation protocol, and baseline configurations. All backbones are frozen and evaluated with the decoding setup described in Sec. 4.1 and App. A, while full implementation and training details are provided in Apps. C.1 and C.2.3. We will release the fixed data splits, per-instance route rewards, training and evaluation code.

## The Use of Large Language Models

Large language models were used only to support language editing, including grammar correction and local sentence rephrasing. The experimental design, analysis, interpretation of results, and scientific conclusions were produced by the authors.

## References

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Longze Chen, Renke Shan, Huiming Wang, Lu Wang, Ziqiang Liu, Run Luo, Jiawei Wang, Hamid Alinejad-Rokny, and Min Yang. 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.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 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.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. 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.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

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

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

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *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.

Jonas Geiping, Sean Michael 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. 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.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, 2024.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason E Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025.

Ahmed Heakl, Martin Gubri, Salman Khan, Sangdoo Yun, and Seong Joon Oh. Dr.LLM: Dynamic layer routing in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. DynaBERT: Dynamic BERT with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. doi: 10.48550/arXiv.2001.08361.

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

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? test-time depth adaptation of pretrained LLMs, 2025.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 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.

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

Xuan Luo, Weizhi Wang, and Xifeng Yan. 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.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025. doi: 10.48550/arXiv.2511.07384.

Xin Men, Mingyu Xu, Bingning Wang, Qingyu Zhang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *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.

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, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *The Thirteenth International Conference on Learning Representations*, 2025.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Weiqiao Shan, Long Meng, Tong Zheng, Yingfeng Luo, Bei Li, Junxin Wang, Tong Xiao, and Jingbo Zhu. Early exit is a natural capability in transformer-based models: An empirical study on early exit without joint optimization, 2024.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 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.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 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.

Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-star: Language models can teach themselves to think before speaking. In *First Conference on Language Modeling*, 2024.

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. doi: 10.48550/arXiv.2510.25741.

# A Evaluation Protocol

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.

## A.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 try four patterns in order: an answer label such as `Answer: B`, a phrase such as `Correct answer is B`, a letter at the start of the output, and a phrase such as `Option B`. We return the first match of the first successful pattern, converted to uppercase.

**Step 2: Fallback matching.** If Step 1 fails, we search for isolated option letters and return the last match. If that also fails, we return the first standalone letter anywhere in the output. These letter patterns are not restricted to the choices in the prompt. A letter outside the valid choices cannot match the reference and is counted as incorrect. Any nonempty letter prediction prevents the choice text fallback.

**Step 3: Choice text overlap.** If no letter is found, we normalize the output and choice texts by lowercasing, removing punctuation, and collapsing whitespace. We first accept an exact normalized choice match. Otherwise, we compute the fraction of each choice's distinct tokens present in the output. We accept the unique highest scoring choice if this fraction is at least $0.8$.

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

## A.2 Numeric Extraction

For DART, the extractor accepts boxed answers, answer phrases, and unboxed numbers in that order.

**Boxed content.** We first use the grader's boxed answer extractor, with a regular expression fallback that takes the last boxed span.

**Answer phrases.** If no box is found, we try phrases for the correct answer, final answer or answer, therefore, and so, in that order. Within the first matching phrase, we take boxed content if available, then the last signed integer or decimal token, and otherwise the remaining phrase text.

**Numeric fallback.** If no phrase matches, we take the last signed integer or decimal token in the output. When extraction fails or reaches this fallback, we also check the prompt followed by the generated suffix. This handles answers that complete a box opened in the prompt, using the same extraction order.

**Grading.** We compare the extracted answer with the reference using symbolic grading with normalization. Equivalent extracted values such as $2.0$ and $2$ are accepted. If the grader raises an exception, we compare lowercased strings after removing whitespace. Failed extraction or an incorrect value receives zero credit. DART evaluation does not require a boxed answer.

## A.3 Controls for Answer Formatting

The same extraction and grading functions are used for default execution, `sBLR`, `aBLR`, and the routing baselines. We additionally evaluate three controls using the model outputs.

**Numeric option fallback.** Base models sometimes answer an MCQ with an option number rather than the requested letter. For example, a model may output `2` or `answer is 2` when the expected answer is $B$. For default outputs with no valid extracted option letter, we therefore map $1, 2, 3, 4$ to $A, B, C, D$ and rescore them. The permissive fallback used in Tab. 3 accepts a bare leading option number, an explicit answer phrase such as `answer is 2`, or an explicit option phrase such as `option 2`. This control applies only to MCQs and deliberately gives the default route the benefit of answers that may be correct but expressed in the wrong format.

**A stricter fallback.** The permissive rule can also match numbers that are not intended as answers. For example, an output beginning with `1. The facial nerve, also known as ...` would be interpreted as answer $A$. Across the three scales, there are $54{,}864$ default MCQ outputs in total. Of these, $14{,}267$ ($26.0\%$) contain no valid option letter and are therefore candidates for repair. The permissive fallback rewrites $11{,}772$ outputs, corresponding to $82.5\%$ of these candidates and $21.5\%$ of all MCQ outputs. Of the rewritten outputs, $7{,}512$ are mapped to $A$, $2{,}998$ to $B$, $962$ to $C$, and $300$ to $D$, and $34.7\%$ become correct compared with $25\%$ under random guessing. We therefore also evaluate a stricter fallback that accepts only explicitly stated answers or option phrases and ignores bare leading numbers. It rewrites only $316$ outputs and changes default accuracy by at most $0.37$ points, while the `aBLR` gains remain essentially unchanged (Tab. 4). The format-repair shares of $45\%$, $40\%$, and $27\%$ in Tab. 3 should therefore be viewed as conservative upper bounds. The stricter rule is conservative in the opposite direction, since it does not recover outputs consisting only of a bare option number.

**Default parsable subset.** As a separate control, we restrict evaluation to examples whose default output already yields a nonempty extraction. This removes cases where the default route fails only because its answer cannot be parsed, allowing us to test whether `aBLR` still improves predictions when formatting is not the main issue. The resulting subsets contain $1{,}197$, $1{,}120$, and $1{,}201$ of $1{,}280$ examples at 0.5B, 1.5B, and 7B, respectively. On these examples, `aBLR` still improves over the default by $3.18$, $5.98$, and $10.91$ points (Tab. 5). Since the subset is selected using the default output, this is a conditional comparison rather than full test accuracy.

On instruction-tuned checkpoints, all $1{,}280$ default outputs at each scale are already parsable and neither fallback changes accuracy. This suggests that the formatting failures above are specific to the base checkpoints rather than the evaluation procedure.

Table 4: **Answer extraction controls on Qwen2.5 base models.** Accuracy under the primary parser and permissive and strict numeric fallbacks. Fallbacks are applied only to default outputs.

| Scale | Parser | Default | sBLR | aBLR |
|---|---|---|---|---|
| 0.5B | Primary | 27.5 | 36.7 | 36.9 |
| | + permissive fallback | 31.8 | 36.7 | 36.9 |
| | + strict fallback | 27.5 | 36.7 | 36.9 |
| 1.5B | Primary | 25.6 | 43.3 | 44.5 |
| | + permissive fallback | 33.4 | 43.3 | 44.5 |
| | + strict fallback | 26.0 | 43.3 | 44.5 |
| 7B | Primary | 35.6 | 58.3 | 61.9 |
| | + permissive fallback | 42.8 | 58.3 | 61.9 |
| | + strict fallback | 35.6 | 58.3 | 61.9 |

Table 5: **Nonempty default extraction control.** `aBLR` gain over the default on examples with a nonempty default extraction.

| | 0.5B | 1.5B | 7B |
|---|---|---|---|
| `aBLR` gain over default | +3.18 | +5.98 | +10.91 |
| Retained examples (of 1,280) | 1,197 | 1,120 | 1,201 |

# B Route Properties and Reward Details

## B.1 Default Route Inclusion

The BLR route set includes the default route $r_0$. As a result, the best admissible route cannot have lower reference likelihood than the default route.

**Proposition 1** (Best admissible route dominates the default route)**.** *Fix an input–output pair $(\boldsymbol{x}, \boldsymbol{y})$ and a finite 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 $r_0$. □

This follows directly from set inclusion. It does not imply that a learned policy improves accuracy, or that higher reference likelihood changes the decoded answer correctly.

## B.2 Oracle Route-Set Bound

For a dataset $\mathcal{D}$, fix the decoder and grading rule and let $\mathbf{1}[\mathrm{correct}(\boldsymbol{x}, r)]$ denote generated answer correctness against $\boldsymbol{y}$ as in Sec. 3.6. We define the oracle accuracy of a route set $\mathcal{R}$ as

$$C(\mathcal{R}) = \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}} \max_{r \in \mathcal{R}} \mathbf{1}[\mathrm{correct}(\boldsymbol{x}, r)].$$

This is the quantity reported as Oracle BLR. Because $r_0 \in \mathcal{R}$, it is at least as high as default accuracy and gives an upper bound for policies selecting among the same route outputs. It measures route-set coverage rather than whether the successful route can be identified from the input. Appendix D.3 provides matched candidate and voting controls that further examine this distinction. In contrast, a finite CoLa (MCTS) run evaluates only part of its larger route space, so its observed gold guided result is not the oracle accuracy of the full space.

## B.3 Reward Normalization

For each route, we compute the mean reference token log probability as

$$\ell(\boldsymbol{x}, r) = \frac{1}{T_y} \sum_{t=1}^{T_y} \log p_{\boldsymbol{\theta}}(y_t \mid \boldsymbol{x}, y_{<t}, r).$$

For each example, let $\ell_{\min}$ and $\ell_{\max}$ be its minimum and maximum over $\mathcal{R}_{\mathrm{BLR}}$. We normalize across routes for the same example by

$$\ell_{\mathrm{norm}}(\boldsymbol{x}, r) = \frac{\ell(\boldsymbol{x}, r) - \ell_{\min}}{\max(\ell_{\max} - \ell_{\min}, 10^{-8})}.$$

When all routes have the same likelihood, the normalized term is zero.

Let $p_{r,t}$ be the next token distribution over $\mathcal{V}$ conditioned on $(\boldsymbol{x}, y_{<t})$ under route $r$, and let $H(p) = -\sum_{v \in \mathcal{V}} p(v) \log p(v)$ be its entropy. The confidence term is

$$c_{\mathrm{correct}}(\boldsymbol{x}, r) = \mathbf{1}[\mathrm{correct}(\boldsymbol{x}, r)] \left(1 - \frac{1}{T_y \log |\mathcal{V}|} \sum_{t=1}^{T_y} H(p_{r,t})\right).$$

Likelihood and entropy are computed using the reference prefix, while correctness is determined from the generated answer. These quantities provide offline supervision for router training and are not required at deployment. The cost term $|r|/(2L)$ measures route executions during training, while the runtime analysis separately accounts for scouting, router evaluation, and generation.

# C Router Design and Ablations

## C.1 Router Architecture and Implementation

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 global attention dimension or local MLP 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 $\ell$, it predicts action values for (i) continuing on the default route and (ii) jumping back to start a recursive block at some earlier layer $s < \ell$. 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 learned attention query. The second stage pools across the $L$ layers using the same mechanism. Both stages project keys to dimension $D$ and values to the backbone dimension $d$. The resulting vector in $\mathbb{R}^d$ is mapped by a linear projection to scores over $\mathcal{R}_{\texttt{BLR}}$, 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 $\ell$ are continuing on the default route or jumping back to start a recursive block at some earlier layer $s < \ell$. Each action $a \in \mathcal{A}_\ell$ induces a route $r(a) \in \mathcal{R}_{\texttt{BLR}}$ and defines the target $q^*(\boldsymbol{x}, \ell, a) := u(\boldsymbol{x}, r(a); \boldsymbol{\lambda})$.

Let $\boldsymbol{x}_\ell \in \mathbb{R}^{W \times d}$ denote the pooled hidden states at layer $\ell$. The local router outputs $g_{\boldsymbol{\phi}}(\boldsymbol{x}_\ell) \in \mathbb{R}^{|\mathcal{A}_\ell|}$ and is trained with

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

At inference, we perform a full preliminary forward pass to compute $g_{\boldsymbol{\phi}}(\boldsymbol{x}_\ell)$ for all layers. We then select the jump with the largest positive margin over the continue value at the same layer, comparing all $(\ell, a)$ pairs. This introduces one recursive block, or recovers the default route if no recursive action exceeds the continue value at any layer.

**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.

### C.1.1 A Worked Inference Example

We illustrate `aBLR` inference using the global router presented in Sec. 3.5. Consider a six-layer backbone and a prompt $\boldsymbol{x}$ and suppose the router selects the block from layer 2 through layer 3. Inference proceeds as follows.

1. **Prompt scouting pass.** Run the full prompt once through the default sequence $(1, 2, 3, 4, 5, 6)$ and pool its hidden states. This pass supplies router features and generates no answer tokens.
2. **Router forward pass.** Apply $g_{\boldsymbol{\phi}}$ once to the pooled prompt features and select the highest scoring route. In this example, the selected layer sequence is $(1, 2, 3, 2, 3, 4, 5, 6)$.
3. **Routed prefill.** Set the backbone's forward execution order to this sequence and run the full prompt again, starting at layer 1. This pass builds the route's KV cache and predicts the first answer token. The scouting pass does not replace the routed prefill.

4. **Generation.** Decode the remaining tokens with the same route and KV cache. No further scouting pass or router call is made for generated tokens.

The repeated calls to physical layers 2 and 3 occupy virtual steps 4 and 5, with separate KV entries from steps 2 and 3. The eight virtual steps reuse six sets of layer weights. Appendix F.1 includes the scouting pass and router call in the adaptive cost.

### C.2 Design Ablations

Each ablation varies one choice from the default configuration, with pooling window $W = 32$ (Fig. 5b) and global attention dimension $D = 256$ (Fig. 5a).

#### C.2.1 Local Router Ablation

Table 6: **Router-family ablation on Qwen2.5 base models.** Entries are weighted average accuracy over all ten 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) | **36.9** (+9.4) | **44.5** (+18.9) | **61.9** (+26.3) |

Both routers improve over the unrouted baseline (Tab. 6), but the global router is stronger at every scale on the all-dataset weighted average, and the gap widens with scale: $+2.3$ points at 0.5B, $+3.3$ points at 1.5B, and $+6.6$ points at 7B. The global router's two-stage attention pooling outperforms the local architecture at all three evaluated scales.

#### C.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 7 lists the profiles used in the reward ablation of Fig. 5c. 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 7: **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.

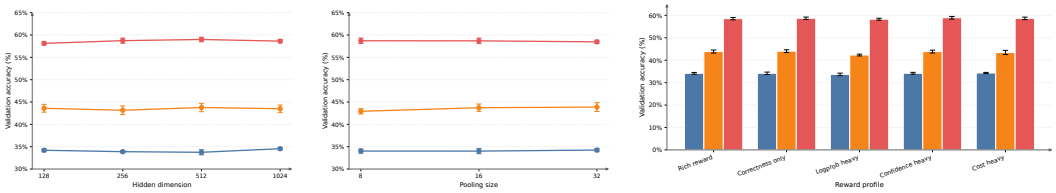
| Reward profile | $\boldsymbol{\lambda}_{\text{lp}}$ | $\boldsymbol{\lambda}_{\text{conf}}$ | $\boldsymbol{\lambda}_{\text{cost}}$ |
|---|---|---|---|
| Correctness only | 0.0 | 0.0 | 0.0 |
| Rich-reward | 0.2 | 0.2 | 0.2 |
| Logprob-heavy | 0.5 | 0.1 | 0.2 |
| Confidence-heavy | 0.1 | 0.5 | 0.2 |
| Cost-heavy | 0.2 | 0.2 | 0.5 |

#### C.2.3 Hyperparameter Ablation

**Training data and validation protocol.** Table 8 reports the fitting, validation, full training, and test counts. We use the official ARC training and test splits, while the DART splits are constructed for this study using the counts shown in the table. OpenBookQA, MMLU, and GPQA Diamond provide test examples only.

For the validation study, we divide each training dataset into 80% fitting examples and 20% validation examples. This gives 5,096 fitting examples and 1,274 validation examples. We use seed 42 and keep the assignments fixed across all configurations and the three Qwen2.5 base backbones. We concatenate the fitting examples without dataset balancing or upsampling.

**Hyperparameter search.** We search a grid of router dimensions $\{128, 256, 512, 1024\}$, pooling sizes $\{8, 16, 32\}$, and the five reward profiles in Tab. 7, giving 60 configurations for each Qwen2.5




(a) Hidden dimension. (b) Pooling size. (c) Reward profile.

Figure 5: **Global router hyperparameter ablations on the validation split.** Validation accuracy across hidden dimension (a), pooling size (b), and reward profile (c), averaged over the remaining hyperparameters. Error bars show one sample standard deviation across configurations. Colors denote model scale: ■ Qwen2.5-0.5B, ■ Qwen2.5-1.5B, and ■ Qwen2.5-7B.

base backbone and 180 runs in total. Each configuration trains for 100 epochs with AdamW, learning rate $2 \times 10^{-4}$, weight decay 0.01, exponential moving average decay 0.999, a cosine learning rate schedule, and training seed 42. Checkpoints are selected by example-weighted validation accuracy.

Table 8: **Training, validation, and test counts for each Qwen2.5 base backbone.** Full training combines fitting and validation and is the dataset used for the main training runs.

| Dataset | Fitting | Validation | Full training | Test |
|---|---|---|---|---|
| ARC Easy | 1,801 | 450 | 2,251 | 2,376 |
| ARC Challenge | 895 | 224 | 1,119 | 1,172 |
| DART-1 | 160 | 40 | 200 | 200 |
| DART-2 | 320 | 80 | 400 | 400 |
| DART-3 | 480 | 120 | 600 | 600 |
| DART-4 | 640 | 160 | 800 | 800 |
| DART-5 | 800 | 200 | 1,000 | 1,000 |
| OpenBookQA | 0 | 0 | 0 | 500 |
| MMLU | 0 | 0 | 0 | 14,042 |
| GPQA Diamond | 0 | 0 | 0 | 198 |
| Total | 5,096 | 1,274 | 6,370 | 21,288 |

We evaluate 180 global-router configurations over hidden dimension, pooling size, and reward profile using the fixed fitting and validation splits in Tab. 8. Figure 5 shows that validation accuracy varies little across these settings. We use hidden dimension 256, pooling size 32, and the confidence-heavy reward, which reaches 45.42% mean validation accuracy across scales, within 0.89 points of the best shared setting at 46.31%. This setting is retrained on the full 6,370 training examples and used for all main experiments.

### C.2.4 LoRA Fine-Tuning Comparison

LoRA (Hu et al., 2022) 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 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. 9 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 compares an oracle with a deployable accuracy and does not establish that a BLR policy outperforms LoRA. The results therefore highlight that frozen routes produce correct answers, while reliable selection remains a separate challenge.

Table 9: **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) |

# D Extended Empirical Analysis

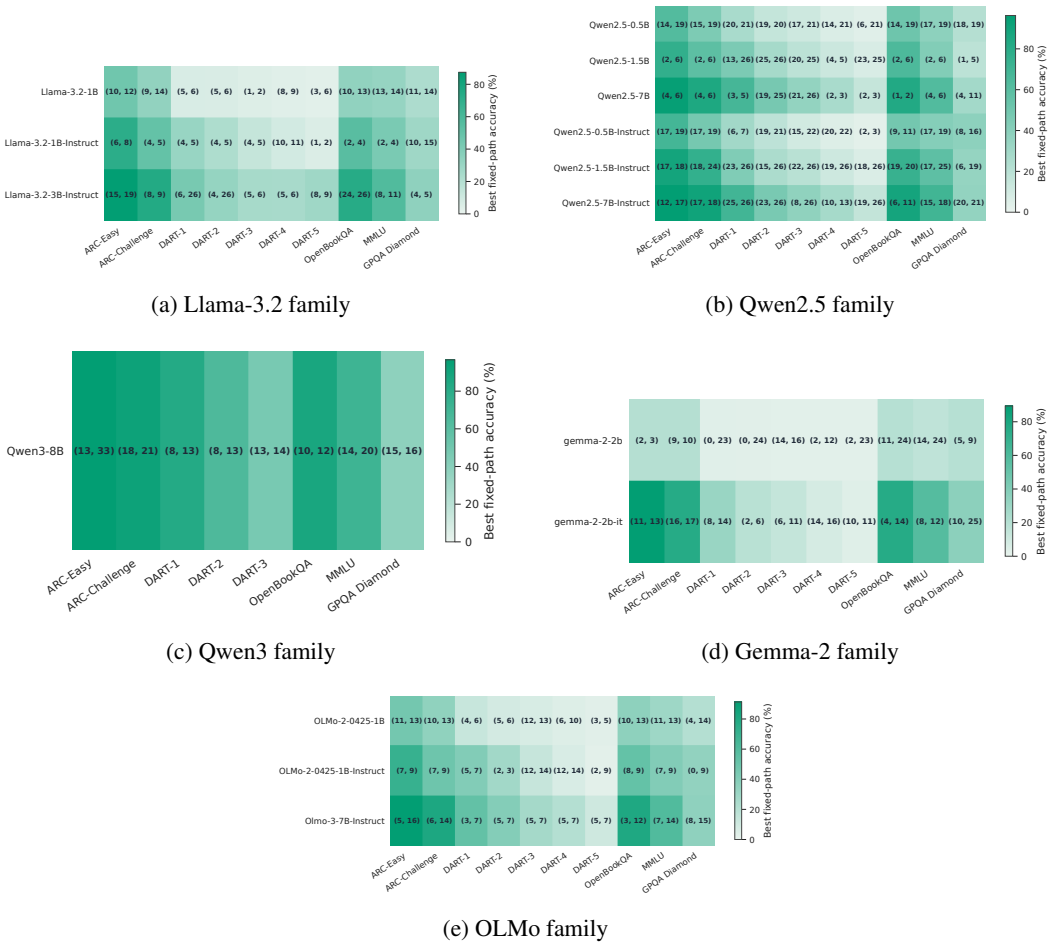
## D.1 Generality Across Model Families

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 6 to 8 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

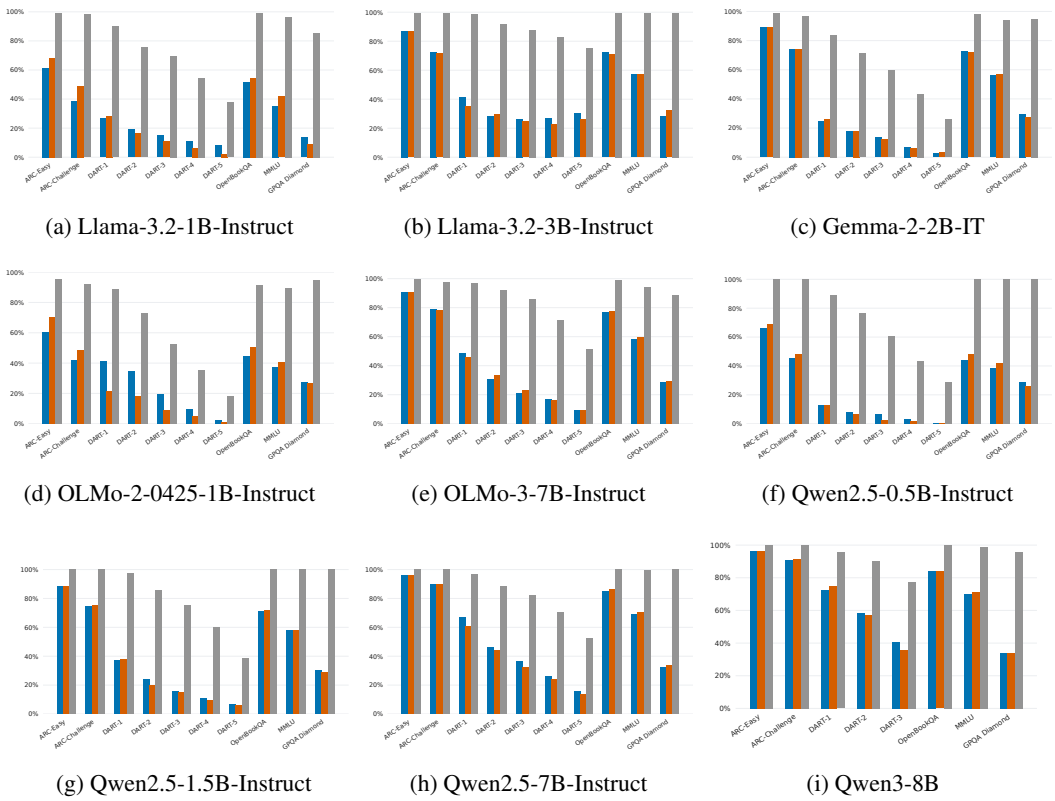
(b) Qwen2.5 family

(c) Qwen3 family

(d) Gemma-2 family

(e) OLMo family

Figure 6: **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. 3.4 into per-checkpoint, per-benchmark heatmaps.

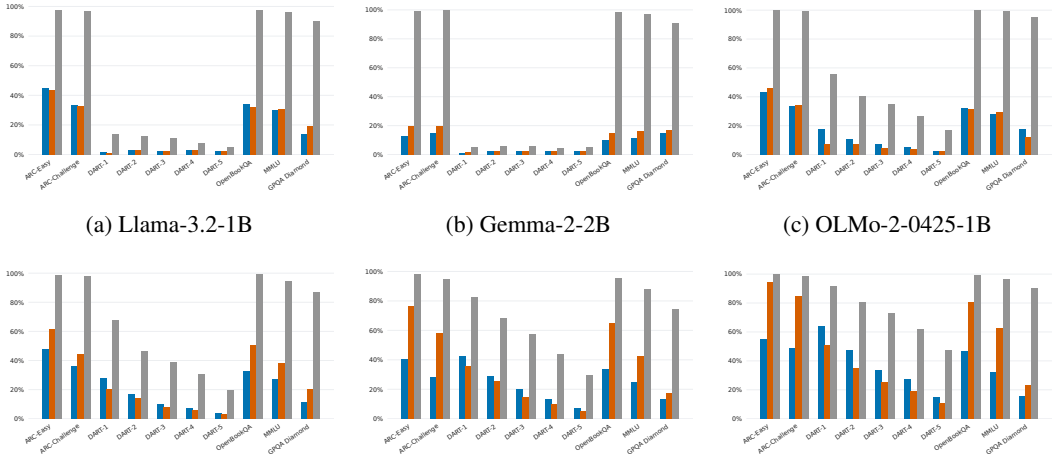


(a) Llama-3.2-1B-Instruct (b) Llama-3.2-3B-Instruct (c) Gemma-2-2B-IT

(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 7: **Instance-specific BLR gaps persist across instruction-tuned checkpoints.** ■ baseline (unrouted), ■ best sBLR block, and ■ oracle. The instance-level oracle exceeds both the baseline and the best sBLR block, showing that BLR routes still produce correct answers 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 8: **Instance-specific BLR gaps across base checkpoints.** ■ baseline (unrouted), ■ best sBLR block, and ■ oracle. The instance-level oracle exceeds both the baseline and the best sBLR block across benchmarks, showing that BLR routes produce correct answers at the per-dataset level across architectures.

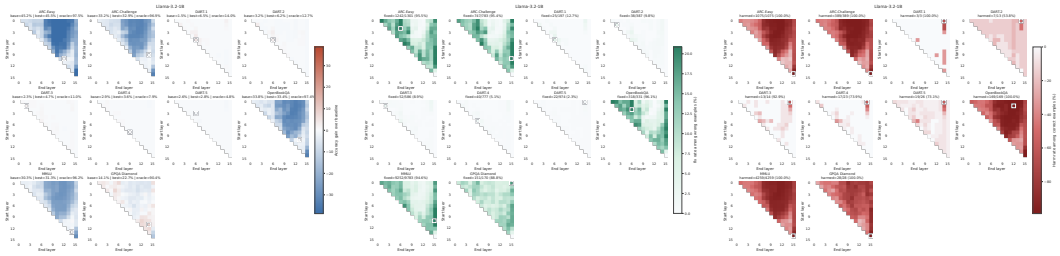
### D.2 HEATMAPS

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.

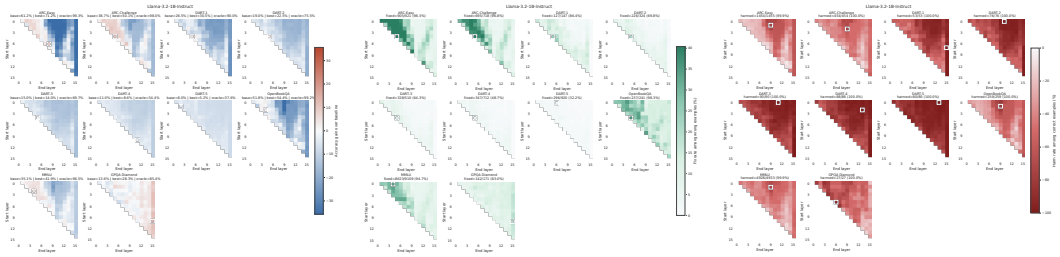
The heatmaps make the empirical findings from Sec. 4.2 explicit. Useful recursive blocks form structured regions rather than isolated cells, but these regions shift across tasks and models. Many blocks are harmful, especially when recursion starts early and spans a large portion of the network. The oracle 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.



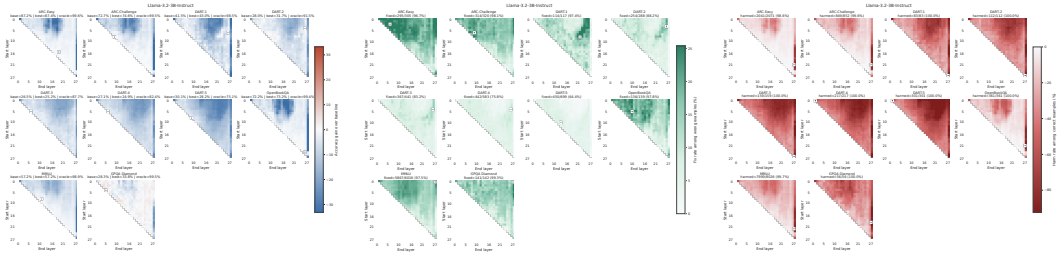
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 9: **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.



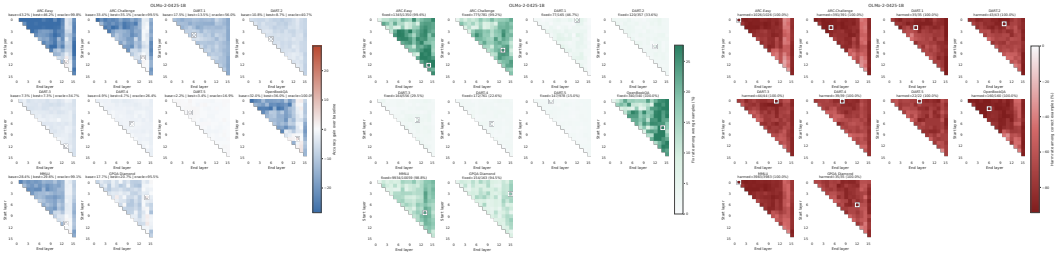
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 10: **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.



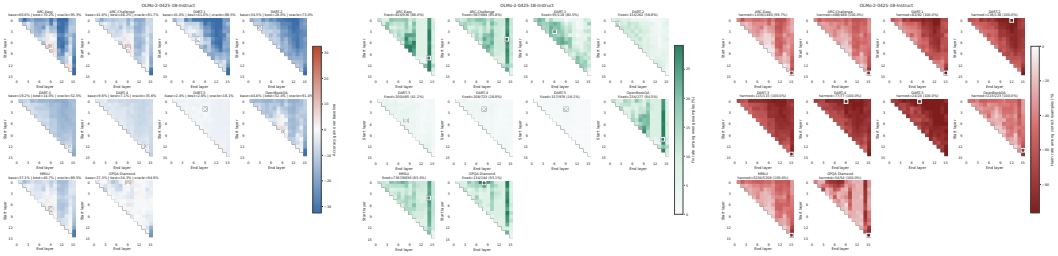
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 11: **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.



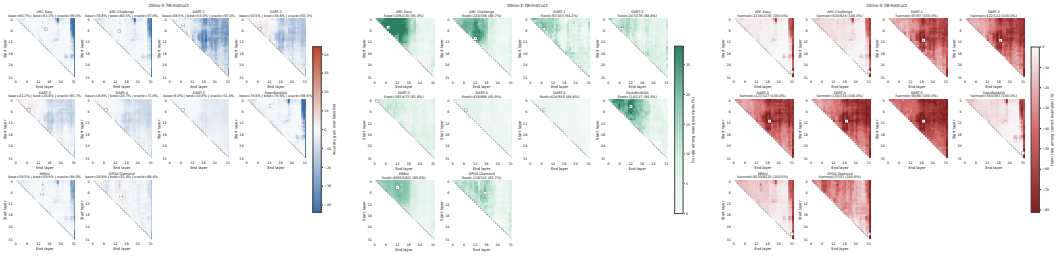
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 12: **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.



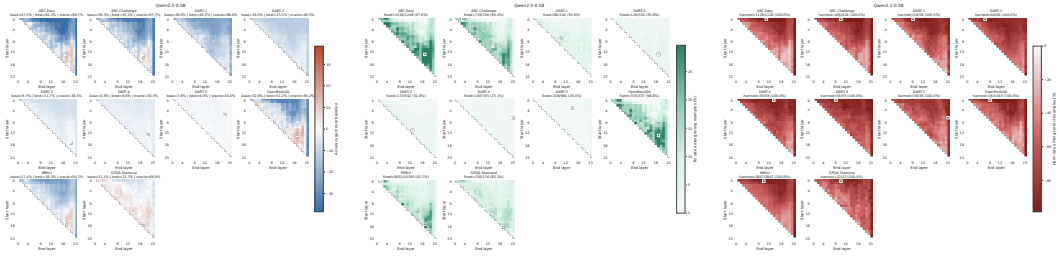
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 13: **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.



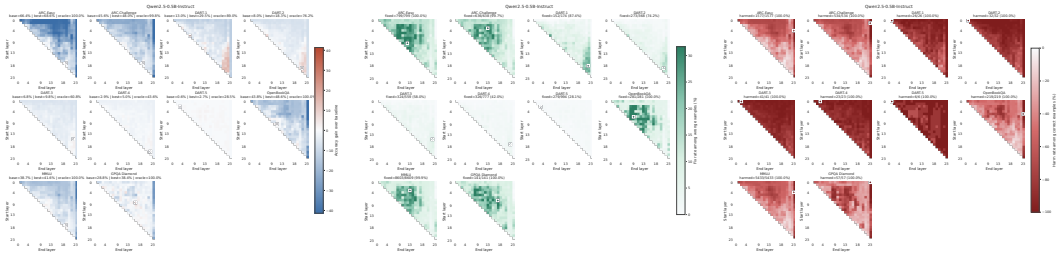
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 14: **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.



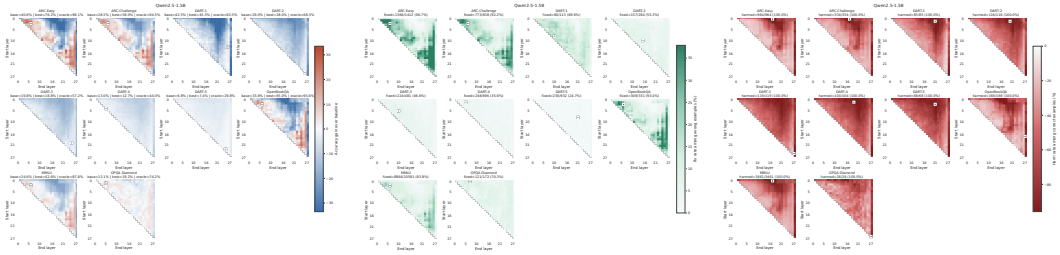
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 15: **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.



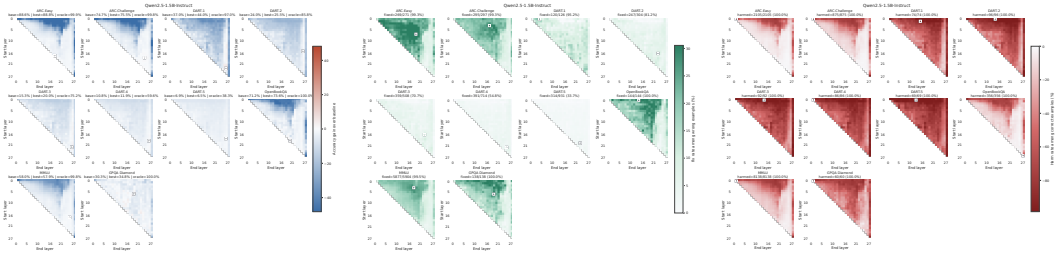
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 16: **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.



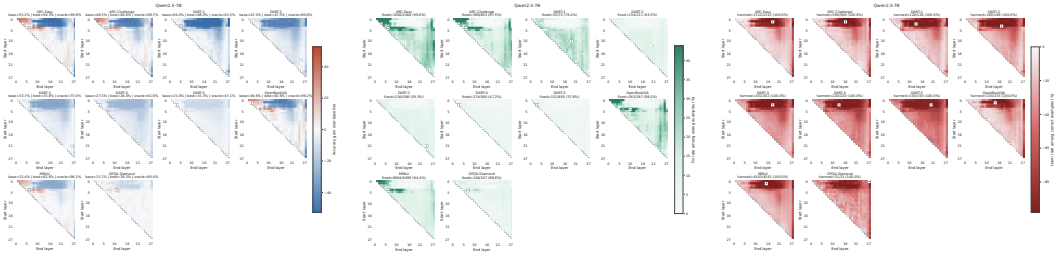
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 17: **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.



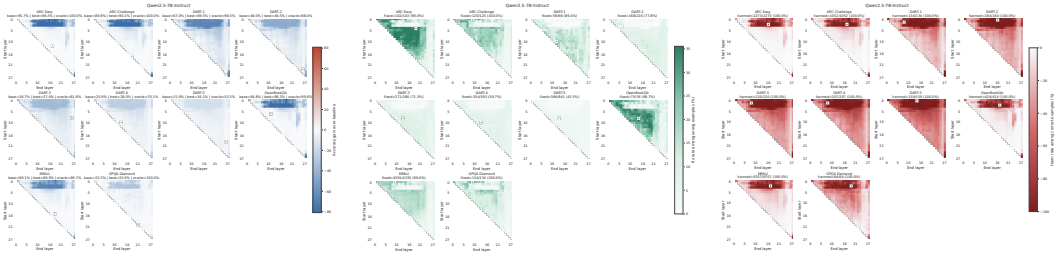
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 18: **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.



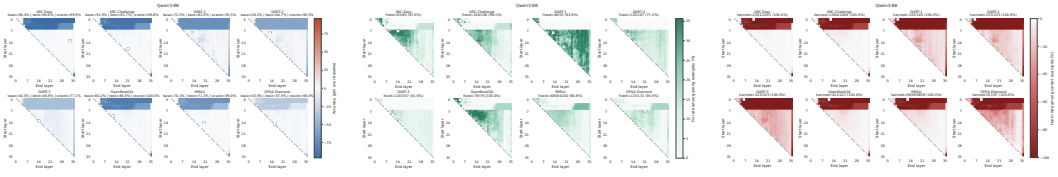
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 19: **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.



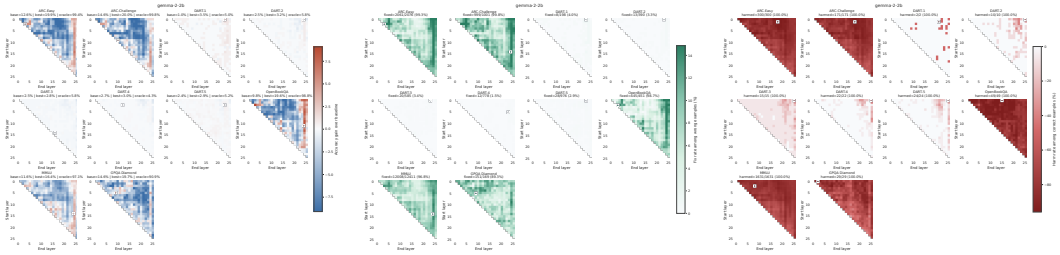
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 20: **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.



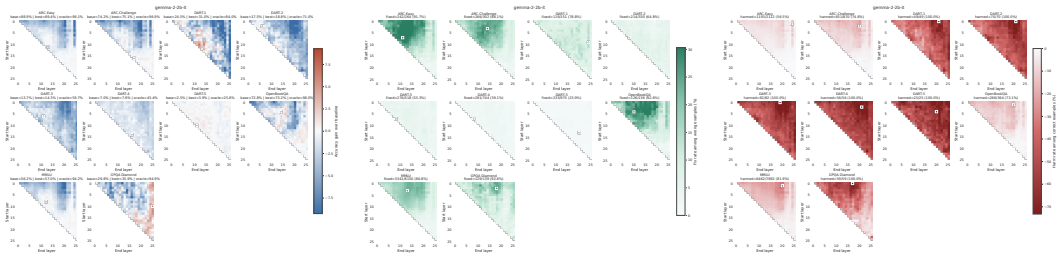
(a) Accuracy gain (b) Fix count (c) Harm count

Figure 21: **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 22: **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 23: **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.

### D.3 Oracle Accuracy and Majority Voting

**Matched candidate setup.** We test whether high Oracle BLR accuracy reflects route-specific output diversity or a generic best-of-many effect. Using Qwen2.5-0.5B, we sample 128 test examples from each of the ten datasets. The MCQ suite combines ARC Easy, ARC Challenge, OpenBookQA, MMLU, and GPQA Diamond, while DART combines its five free-form numeric levels; GPQA Diamond is also reported separately. For each example, BLR provides all 277 greedy route outputs, including the default route. We compare these with 277 stochastic generations from the default route using temperature 0.7 and top-$p = 0.95$. BLR routes are randomly ordered over 50 orderings and stochastic generation is repeated over three seeds. We evaluate matched candidate budgets from $K = 1$ to $K = 277$.

**Oracle coverage.** For each budget $K$, oracle accuracy counts an example as correct when at least one of the first $K$ candidates is correct. This uses the gold answer for selection and therefore measures candidate coverage rather than deployable accuracy. Figure 24 and Tab. 10 shows that stochastic sampling is stronger at intermediate budgets and reaches similar or higher coverage at $K = 277$. Thus, part of the large Oracle BLR result is a general best-of-many effect rather than a property unique to BLR.

**Free-form answers.** The MCQ results alone cannot distinguish useful route diversity from chance collisions in a small answer space. DART provides a complementary control because answers are free-form numeric rather than one of four choices. On DART, BLR oracle coverage rises from $6.26\%$ at $K = 1$ to $40.63\%$ at $K = 277$. This increase cannot be explained solely by repeatedly selecting among four answer choices and shows that different BLR routes also expose complementary correct outputs in an open-ended setting. Stochastic sampling reaches even higher coverage at $54.79\%$, confirming that this phenomenon is not unique to BLR.

**Selection without gold answers.** We finally replace oracle selection with majority voting over the extracted candidate answers. At $K = 277$, voting reaches only $9.06\%$ for BLR and $13.13\%$ for stochastic sampling on DART, far below their oracle coverage of $40.63\%$ and $54.79\%$. The gap illustrates the distinction between having a correct answer somewhere in the candidate set and identifying it without access to the gold label. Moreover, majority voting requires generating $K$ candidates for each input. In contrast, sBLR and aBLR select a single route and generate a single answer, avoiding this additional generation cost and addressing route selection directly. Full results across candidate budgets are reported in Tabs. 11 and 12.

Table 10: **Oracle accuracy versus voting at $K = 277$.** Entries are percentages with standard deviations across orderings or generation seeds. BLR uses population standard deviations across its 50 orderings, and sampling uses sample standard deviations across three seeds. Zero variation across BLR orderings does not imply zero uncertainty over examples.

| Suite | Candidates | Oracle accuracy | Voting accuracy |
|---|---|---|---|
| MCQ | BLR | $94.69 \pm 0.00$ | $32.34 \pm 0.00$ |
| | Sampling | $95.36 \pm 0.09$ | $38.59 \pm 0.16$ |
| GPQA | BLR | $85.94 \pm 0.00$ | $17.19 \pm 0.00$ |
| | Sampling | $85.42 \pm 1.19$ | $20.83 \pm 0.45$ |
| DART | BLR | $40.63 \pm 0.00$ | $9.06 \pm 0.00$ |
| | Sampling | $54.79 \pm 0.74$ | $13.13 \pm 0.56$ |

Table 11: **Gold label coverage at every evaluated budget.** Entries are percentages, reported as mean $\pm$ standard deviation across 50 BLR orderings or three sampling seeds. BLR uses population standard deviation over the orderings and sampling uses sample standard deviation over seeds. These quantify candidate variation rather than test population uncertainty.

| Suite | Family | 1 | 5 | 10 | 25 | 50 | 100 | 150 | 277 |
|---|---|---|---|---|---|---|---|---|---|
| MCQ suite | BLR | $19.79 \pm 1.24$ | $52.13 \pm 1.45$ | $65.48 \pm 1.25$ | $78.49 \pm 0.99$ | $84.83 \pm 0.84$ | $89.68 \pm 0.66$ | $91.84 \pm 0.61$ | $94.69 \pm 0.00$ |
| MCQ suite | Sampling | $30.05 \pm 0.99$ | $64.90 \pm 1.04$ | $77.60 \pm 1.06$ | $87.40 \pm 0.70$ | $91.56 \pm 0.16$ | $93.54 \pm 0.39$ | $94.27 \pm 0.59$ | $95.36 \pm 0.09$ |
| GPQA | BLR | $11.02 \pm 2.68$ | $31.81 \pm 3.22$ | $43.20 \pm 2.93$ | $57.94 \pm 2.57$ | $67.17 \pm 2.20$ | $75.16 \pm 1.86$ | $79.19 \pm 1.58$ | $85.94 \pm 0.00$ |
| GPQA | Sampling | $17.19 \pm 2.82$ | $41.93 \pm 2.26$ | $55.73 \pm 3.16$ | $69.79 \pm 1.19$ | $77.08 \pm 0.45$ | $80.73 \pm 1.19$ | $82.55 \pm 1.97$ | $85.42 \pm 1.19$ |
| DART | BLR | $6.26 \pm 0.88$ | $16.86 \pm 0.89$ | $21.89 \pm 0.93$ | $28.13 \pm 0.66$ | $32.46 \pm 0.60$ | $36.35 \pm 0.54$ | $38.26 \pm 0.55$ | $40.63 \pm 0.00$ |
| DART | Sampling | $10.21 \pm 1.11$ | $22.24 \pm 1.06$ | $28.96 \pm 0.86$ | $38.65 \pm 1.02$ | $44.64 \pm 1.04$ | $49.58 \pm 0.74$ | $52.14 \pm 0.63$ | $54.79 \pm 0.74$ |

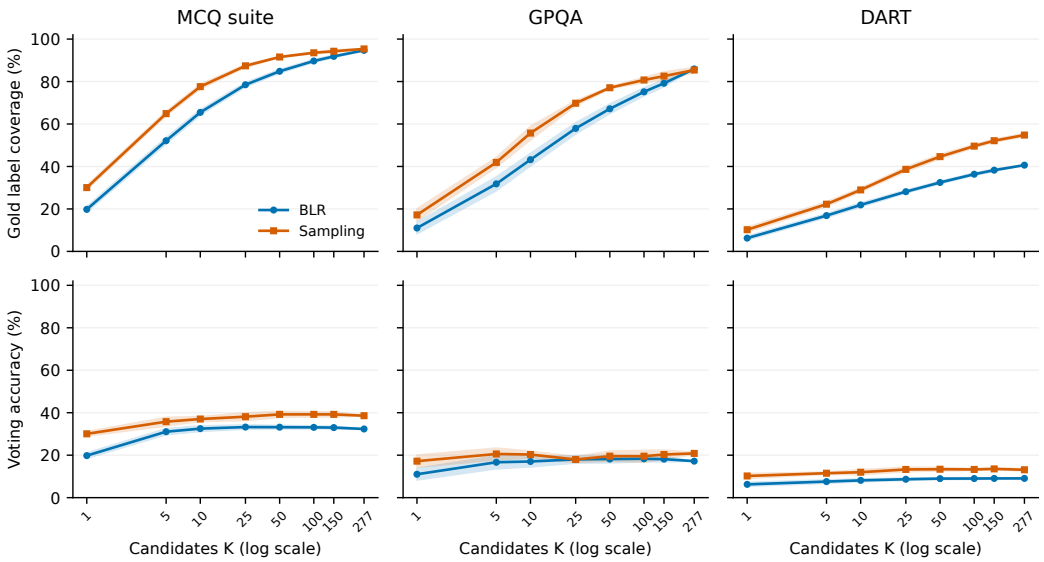



Figure 24: **Oracle accuracy and majority voting at every evaluated budget.** Top panels report oracle coverage, which uses the gold answer for selection. Bottom panels report voting accuracy, which does not. Lines connect measured candidate counts on a logarithmic horizontal axis. Shading shows variation across 50 BLR orderings or three sampling seeds, not uncertainty over test examples. GPQA is a subset of the MCQ suite.

Table 12: **Majority voting accuracy at every evaluated budget.** Entries are percentages, reported as mean ± standard deviation across 50 BLR orderings or three sampling seeds. BLR uses population standard deviation over the orderings and sampling uses sample standard deviation over seeds. These quantify candidate variation rather than test population uncertainty.

| Suite | Family | 1 | 5 | 10 | 25 | 50 | 100 | 150 | 277 |
|---|---|---|---|---|---|---|---|---|---|
| MCQ suite | BLR | 19.79 ± 1.24 | 31.06 ± 1.41 | 32.50 ± 1.26 | 33.24 ± 0.98 | 33.19 ± 0.80 | 33.11 ± 0.66 | 33.00 ± 0.61 | 32.34 ± 0.00 |
| MCQ suite | Sampling | 30.05 ± 0.99 | 35.78 ± 2.04 | 37.03 ± 1.09 | 38.13 ± 1.80 | 39.22 ± 1.39 | 39.22 ± 1.18 | 39.22 ± 1.13 | 38.59 ± 0.16 |
| GPQA | BLR | 11.02 ± 2.68 | 16.67 ± 3.04 | 17.03 ± 2.53 | 18.02 ± 1.69 | 18.17 ± 1.75 | 18.31 ± 1.35 | 18.14 ± 1.25 | 17.19 ± 0.00 |
| GPQA | Sampling | 17.19 ± 2.82 | 20.57 ± 2.74 | 20.31 ± 1.56 | 17.97 ± 1.35 | 19.53 ± 2.34 | 19.53 ± 2.71 | 20.31 ± 2.07 | 20.83 ± 0.45 |
| DART | BLR | 6.26 ± 0.88 | 7.57 ± 0.76 | 8.14 ± 0.68 | 8.66 ± 0.51 | 8.97 ± 0.42 | 9.00 ± 0.37 | 9.05 ± 0.29 | 9.06 ± 0.00 |
| DART | Sampling | 10.21 ± 1.11 | 11.51 ± 0.50 | 11.98 ± 1.15 | 13.28 ± 1.02 | 13.39 ± 0.80 | 13.28 ± 0.41 | 13.54 ± 0.36 | 13.13 ± 0.56 |

# E Additional Results for the Routing Policies

## E.1 DART Results by Level

Table 13 reports per-level DART results for the Qwen2.5 base models. The joint sBLR block selected on ARC and DART decreases accuracy across DART, but DART-specific and per-level selection reveal useful blocks on several levels. For example, DART-specific sBLR improves level 1 from 42.5% to 45.0% at 1.5B, while per-level selection improves level 3 from 33.7% to 34.3% at 7B. Thus, beneficial BLR blocks exist on DART even when the joint static policy does not select them reliably.

Oracle coverage remains substantially higher, showing that additional correct outputs are available across routes, although this does not imply that they can be identified from training data. The effect is also backbone-dependent: on OLMo-3-7B-Instruct, a single sBLR block selected on joint ARC and DART training data improves all five DART levels (App. E.2.1).

Table 13: **DART accuracy by level for Qwen2.5 base models.** *Joint* selects one block on ARC and DART training data, *DART* on DART only, and *per level* separately for each level. Green entries improve over the default. Oracle selects per instance using the reference answer.

| Backbone | Policy | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 |
|---|---|---|---|---|---|---|
| Qwen2.5-0.5B | Default | 28.0 | 16.5 | 9.7 | 6.9 | 3.6 |
| | sBLR (joint) | 20.5 | 13.8 | 8.2 | 5.9 | 3.2 |
| | sBLR (DART) | 26.5 | 14.5 | 10.2 | 5.6 | 3.2 |
| | sBLR (per level) | 26.5 | 15.3 | 10.0 | 5.1 | 3.1 |
| | Oracle | 68.0 | 46.5 | 38.5 | 30.3 | 19.4 |
| Qwen2.5-1.5B | Default | 42.5 | 29.0 | 19.8 | 13.0 | 6.8 |
| | sBLR (joint) | 35.5 | 25.2 | 14.7 | 10.0 | 5.0 |
| | sBLR (DART) | 45.0 | 27.0 | 15.2 | 10.9 | 6.8 |
| | sBLR (per level) | 41.0 | 26.0 | 16.0 | 11.5 | 5.7 |
| | Oracle | 82.5 | 68.3 | 57.2 | 44.0 | 29.8 |
| Qwen2.5-7B | Default | 64.0 | 47.2 | 33.7 | 27.5 | 15.0 |
| | sBLR (joint) | 62.5 | 45.2 | 32.5 | 24.9 | 13.7 |
| | sBLR (DART) | 64.0 | 46.8 | 31.3 | 25.5 | 13.8 |
| | sBLR (per level) | 61.5 | 46.8 | 34.3 | 25.5 | 13.2 |
| | Oracle | 91.5 | 80.8 | 73.0 | 61.8 | 47.1 |

**Corrected and harmed answers.** Table 14 compares how often the selected static blocks correct an incorrect default answer or change a correct answer to an incorrect one. For the Qwen2.5 base models, the joint blocks produce both effects but harm more answers than they fix, explaining the overall DART decrease. For OLMo-3-7B-Instruct, the opposite holds at every level, consistent with its positive DART gains.

Table 14: **DART fixes and harms for the selected static blocks.** Each cell reports fixes / harms as counts of examples. Levels 1 through 5 contain 200, 400, 600, 800, and 1,000 test examples. These are the joint policy outputs, corresponding to the sBLR (joint) rows of Tab. 13 for the Qwen base backbones and to the OLMo-3-7B-Instruct route of App. E.2.1.

| Backbone | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 |
|---|---|---|---|---|---|
| Qwen2.5-0.5B | 7 / 22 | 19 / 30 | 15 / 24 | 19 / 27 | 8 / 12 |
| Qwen2.5-1.5B | 15 / 29 | 23 / 38 | 24 / 55 | 30 / 54 | 16 / 34 |
| Qwen2.5-7B | 9 / 12 | 18 / 26 | 39 / 46 | 29 / 50 | 34 / 47 |
| OLMo-3-7B-Instruct | 25 / 18 | 65 / 28 | 65 / 31 | 74 / 42 | 55 / 36 |

## E.2 Routing Instruction-Tuned Checkpoints

Table 15 evaluates both BLR policies on instruction-tuned Qwen2.5 checkpoints. At 0.5B, sBLR improves overall accuracy from 37.3% to 39.5%, while aBLR reaches 38.4%. At 1.5B and 7B, both policies remain close to the stronger unrouted baselines, with sBLR reaching 55.6% and 67.6%,

and aBLR 54.9% and 67.2%. Thus, unlike on the base checkpoints, practical BLR gains are not consistent after instruction tuning.

Oracle BLR remains high at all three scales (92.8%, 94.4%, and 95.7%), showing that different routes within the BLR family continue to produce complementary correct answers across inputs. At the same time, instruction tuning substantially strengthens the unrouted checkpoints: overall accuracy increases from 27.5% to 37.3% at 0.5B, from 25.6% to 56.1% at 1.5B, and from 35.6% to 67.8% at 7B. This changes the starting point for routing considerably. At 0.5B, the instruction-tuned default already exceeds aBLR on the base checkpoint (37.3% vs. 36.9%), while at 1.5B and 7B it is substantially higher than the corresponding base-checkpoint aBLR results of 44.5% and 61.9%. Applying BLR on top of these stronger checkpoints provides only a small gain at 0.5B and no overall improvement at the two larger scales. These results highlight a limitation of the current route-selection policies: high route-set coverage remains after instruction tuning, but converting it into consistent gains is difficult. We next test whether using a more task-specific static policy improves route selection, while more general approaches remain for future work.

Table 15: **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 | 59.9 (+0.4) | 2.8 (-1.5) | 33.7 (-0.5) | 46.4 (+2.6) | 24.8 (-4.0) | 40.4 (+1.7) | 40.4 (+1.7) | 38.4 (+1.1) |
| | 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 (-0.3) | 71.0 (-0.2) | **28.3** (-2.0) | **57.3** (-0.7) | **57.3** (-0.7) | 55.6 (-0.5) |
| | aBLR | 83.2 (-0.8) | 14.5 (+0.6) | 51.7 (-0.2) | 71.2 (+0.0) | 25.8 (-4.5) | 56.3 (-1.7) | 56.4 (-1.6) | 54.9 (-1.2) |
| | 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 (-0.3) | 64.4 (-0.2) | **85.0** (+0.2) | **32.3** (+0.0) | 69.0 (-0.1) | 69.0 (-0.2) | 67.6 (-0.2) |
| | aBLR | 93.7 (+0.0) | 27.7 (-2.4) | 63.4 (-1.2) | 83.6 (-1.2) | 30.3 (-2.0) | 68.9 (-0.2) | 68.8 (-0.4) | 67.2 (-0.6) |
| | 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) |

### E.2.1 Static Route Selection on Instruction-Tuned Checkpoints

**Setup.** To test whether the weaker instruction-tuned results are partly due to selecting one static block across heterogeneous training datasets, we compare two selection granularities. The pooled sBLR policy selects one block jointly on all seven in-domain training splits. As a diagnostic, we also select a separate block on each dataset's training split and evaluate it on the corresponding test split. This per-dataset variant requires task-specific training data and knowledge of the target dataset, so we do not treat it as a general deployment policy. We restrict the comparison to ARC and DART, where both training and test splits are available.

**Results.** Per-dataset selection improves over pooled sBLR on six of the eight instruction-tuned checkpoints, but exceeds the unrouted default on only five. The largest gains over the default are +4.3 points for Llama-3.2-1B-Instruct, +3.0 for OLMo-2-0425-1B-Instruct, and +2.4 for Qwen2.5-0.5B-Instruct. In contrast, Qwen2.5-7B-Instruct and Llama-3.2-3B-Instruct remain below the default by 1.2 and 1.6 points. Thus, finer static selection can recover additional gains on some instruction-tuned checkpoints, but the effect is not consistent across models.

**Interpretation.** Oracle BLR remains substantially higher on all eight checkpoints, showing that the BLR family still contains routes that produce complementary correct answers. Together with the positive gains from per-dataset selection on several checkpoints, this suggests that instruction tuning does not eliminate useful recursive routes. Rather, our current selection strategies do not identify them reliably across models and tasks. Developing more reliable route-selection methods for instruction-tuned checkpoints is therefore an important direction for future work.

Table 16: **Static selection granularity on instruction-tuned checkpoints.** For each dataset, *Block* gives the block selected on that dataset's training split, *Def.* the unrouted accuracy, *Stat.* the accuracy with the selected block, Δ its change from the default, and *Orac.* the Oracle BLR upper bound. *Pooled* summarizes the single block selected jointly across all in-domain training splits, while *Per-ds* summarizes dataset-specific selection. Both summaries are weighted by test-set size.

| Data | Block | Def. | Stat. | Δ | Orac. |
|---|---|---|---|---|---|
| **Qwen2.5-0.5B-Instruct** | | | | | |
| ARC-E | 17–19 | 66.4 | 68.6 | (+2.3) | 100.0 |
| ARC-C | 9–10 | 45.6 | 46.8 | (+1.2) | 99.8 |
| DART-1 | 19–22 | 13.0 | 26.5 | (+13.5) | 89.0 |
| DART-2 | 17–22 | 8.0 | 16.5 | (+8.5) | 76.2 |
| DART-3 | 17–22 | 6.8 | 9.2 | (+2.3) | 60.8 |
| DART-4 | 19–21 | 2.9 | 3.4 | (+0.5) | 43.6 |
| DART-5 | 6–13 | 0.6 | 1.9 | (+1.3) | 28.5 |
| *Pooled* | 17–19 | 34.2 | 34.8 | (+0.6) | 76.8 |
| *Per-ds* | – | 34.2 | 36.6 | (+2.4) | 76.8 |
| **Qwen2.5-1.5B-Instruct** | | | | | |
| ARC-E | 19–21 | 88.6 | 88.3 | (-0.3) | 99.9 |
| ARC-C | 19–23 | 74.7 | 75.1 | (+0.4) | 99.8 |
| DART-1 | 9–26 | 37.0 | 36.5 | (-0.5) | 97.0 |
| DART-2 | 23–26 | 24.0 | 24.8 | (+0.8) | 85.8 |
| DART-3 | 22–26 | 15.3 | 20.0 | (+4.7) | 75.2 |
| DART-4 | 22–26 | 10.8 | 11.4 | (+0.6) | 59.6 |
| DART-5 | 23–26 | 6.9 | 5.9 | (-1.0) | 38.3 |
| *Pooled* | 22–25 | 51.9 | 51.6 | (-0.2) | 82.3 |
| *Per-ds* | – | 51.9 | 52.2 | (+0.3) | 82.3 |
| **Qwen2.5-7B-Instruct** | | | | | |
| ARC-E | 22–24 | 95.7 | 95.8 | (+0.2) | 100.0 |
| ARC-C | 13–18 | 89.8 | 89.1 | (-0.7) | 100.0 |
| DART-1 | 13–15 | 67.0 | 64.0 | (-3.0) | 96.5 |
| DART-2 | 10–12 | 46.0 | 44.0 | (-2.0) | 88.0 |
| DART-3 | 9–13 | 36.7 | 32.0 | (-4.7) | 81.8 |
| DART-4 | 23–25 | 25.9 | 24.3 | (-1.6) | 70.1 |
| DART-5 | 13–15 | 15.9 | 13.9 | (-2.0) | 52.5 |
| *Pooled* | 19–26 | 64.6 | 64.4 | (-0.2) | 86.6 |
| *Per-ds* | – | 64.6 | 63.4 | (-1.2) | 86.6 |
| **OLMo-2-0425-1B-Inst.** | | | | | |
| ARC-E | 7–9 | 60.6 | 70.2 | (+9.6) | 95.3 |
| ARC-C | 7–9 | 41.6 | 48.2 | (+6.6) | 91.7 |
| DART-1 | 3–4 | 41.0 | 37.5 | (-3.5) | 88.5 |
| DART-2 | 1–2 | 34.5 | 25.2 | (-9.3) | 73.0 |
| DART-3 | 1–2 | 19.2 | 13.0 | (-6.2) | 52.5 |
| DART-4 | 13–14 | 9.6 | 6.5 | (-3.1) | 35.6 |
| DART-5 | 5–8 | 2.4 | 1.8 | (-0.6) | 18.1 |
| *Pooled* | 7–8 | 36.1 | 38.0 | (+1.9) | 70.1 |
| *Per-ds* | – | 36.1 | 39.1 | (+3.0) | 70.1 |

| Data | Block | Def. | Stat. | Δ | Orac. |
|---|---|---|---|---|---|
| **OLMo-3-7B-Instruct** | | | | | |
| ARC-E | 3–16 | 90.7 | 90.9 | (+0.1) | 99.5 |
| ARC-C | 2–10 | 78.8 | 77.7 | (-1.1) | 97.6 |
| DART-1 | 25–27 | 48.5 | 46.5 | (-2.0) | 97.0 |
| DART-2 | 4–7 | 30.5 | 36.0 | (+5.5) | 92.3 |
| DART-3 | 5–7 | 21.2 | 26.8 | (+5.7) | 85.7 |
| DART-4 | 5–7 | 16.8 | 20.7 | (+4.0) | 71.6 |
| DART-5 | 5–7 | 9.0 | 10.9 | (+1.9) | 51.4 |
| *Pooled* | 5–7 | 55.7 | 57.1 | (+1.4) | 86.6 |
| *Per-ds* | – | 55.7 | 57.2 | (+1.4) | 86.6 |
| **Gemma-2-2b-it** | | | | | |
| ARC-E | 8–13 | 88.9 | 89.0 | (+0.1) | 99.1 |
| ARC-C | 6–10 | 74.2 | 73.7 | (-0.5) | 96.9 |
| DART-1 | 6–10 | 24.5 | 23.0 | (-1.5) | 84.0 |
| DART-2 | 5–14 | 17.5 | 14.0 | (-3.5) | 71.0 |
| DART-3 | 7–8 | 13.7 | 12.8 | (-0.8) | 59.7 |
| DART-4 | 3–7 | 7.0 | 5.1 | (-1.9) | 43.4 |
| DART-5 | 10–15 | 2.5 | 2.9 | (+0.4) | 25.8 |
| *Pooled* | 10–13 | 49.8 | 49.8 | (-0.1) | 74.9 |
| *Per-ds* | – | 49.8 | 49.3 | (-0.6) | 74.9 |
| **Llama-3.2-1B-Instruct** | | | | | |
| ARC-E | 4–5 | 61.2 | 68.8 | (+7.5) | 99.3 |
| ARC-C | 4–5 | 38.7 | 50.1 | (+11.3) | 98.0 |
| DART-1 | 4–5 | 26.5 | 30.5 | (+4.0) | 90.0 |
| DART-2 | 4–5 | 19.0 | 22.5 | (+3.5) | 75.5 |
| DART-3 | 4–5 | 15.0 | 14.0 | (-1.0) | 69.7 |
| DART-4 | 10–11 | 11.0 | 8.6 | (-2.4) | 54.4 |
| DART-5 | 1–2 | 8.0 | 5.2 | (-2.8) | 37.6 |
| *Pooled* | 4–5 | 35.1 | 39.0 | (+3.9) | 79.7 |
| *Per-ds* | – | 35.1 | 39.4 | (+4.3) | 79.7 |
| **Llama-3.2-3B-Instruct** | | | | | |
| ARC-E | 21–22 | 87.2 | 87.2 | (+0.0) | 99.6 |
| ARC-C | 21–22 | 72.7 | 73.5 | (+0.8) | 99.5 |
| DART-1 | 6–10 | 41.5 | 38.5 | (-3.0) | 98.5 |
| DART-2 | 7–8 | 28.0 | 26.2 | (-1.8) | 91.5 |
| DART-3 | 7–11 | 26.5 | 22.8 | (-3.7) | 87.7 |
| DART-4 | 7–9 | 27.1 | 24.3 | (-2.9) | 82.4 |
| DART-5 | 1–3 | 30.1 | 24.3 | (-5.8) | 75.1 |
| *Pooled* | 8–11 | 58.0 | 56.1 | (-1.8) | 92.1 |
| *Per-ds* | – | 58.0 | 56.3 | (-1.6) | 92.1 |

### E.3 Cross-Format OOD Generalization

The main OOD evaluation tests transfer to unseen datasets, but the training mixture contains both multiple-choice ARC and free-form numeric DART examples. We therefore perform an additional experiment that isolates transfer across answer formats. In one direction, we select and train the policies using only ARC and evaluate on DART. In the reverse direction, we use only DART for training and evaluate on the MCQ suite consisting of ARC, OpenBookQA, MMLU, and GPQA Diamond. The target evaluations contain 3,000 DART examples and 18,288 MCQ examples, with dataset accuracies weighted by example count.

Table 17 shows that cross-format transfer is asymmetric. ARC supervision does not improve DART at any scale. In the reverse direction, DART supervision improves MCQ accuracy for sBLR at 1.5B by 12.4 points and for aBLR at 7B by 4.2 points, while the 0.5 point aBLR change at 1.5B is small. Other settings decline relative to the default. Thus, positive cross-format transfer is possible in selected settings, but is not consistent across directions, model scales, or selection policies. This experiment imposes a stricter shift than the main OOD evaluation because both the task distribution and the required answer format change simultaneously.

Table 17: **Cross-format OOD generalization.** Each policy is fitted using only the source-format training data and evaluated on the target-format test data. Entries report test accuracy in percent, with deltas relative to the unrouted default.

| Training → Testing | Method | 0.5B | 1.5B | 7B |
|---|---|---|---|---|
| ARC → DART | Default | 9.0 | 16.4 | 29.6 |
| | sBLR | 6.6 (-2.4) | 13.0 (-3.4) | 21.7 (-7.9) |
| | aBLR | 5.9 (-3.1) | 11.9 (-4.5) | 21.4 (-8.2) |
| DART → MCQ | Default | 30.5 | 27.1 | 36.6 |
| | sBLR | 25.7 (-4.8) | **39.5 (+12.4)** | 35.2 (-1.4) |
| | aBLR | 20.8 (-9.7) | 27.6 (+0.5) | **40.8 (+4.2)** |

### E.4 Ablating the Number of Block Repetitions

BLR repeats the selected block once by design. We ablate this choice on Qwen2.5-0.5B by allowing up to four extra traversals of a single contiguous block. For each repeat count, we evaluate every block and select the static route using the same joint ARC and DART training procedure as sBLR. We then jointly select both the block and the number of extra traversals from this expanded route family.

Table 18 shows that the joint procedure still selects a single extra traversal of layers 13 through 21, exactly matching the original sBLR policy. The train-selected routes with two to four extra traversals also achieve lower test accuracy on both ARC and DART. Thus, enlarging the static search space in this way does not yield a better policy and supports the single-repeat design used in the main experiments.

Combining routes across all repeat counts increases Oracle BLR coverage to 100.0% on ARC and 42.2% on DART. This shows that the larger route family contains additional complementary correct outputs, but does not imply that these outputs can be reliably selected without reference answers. Since the additional repetitions do not improve the train-selected static policy, we retain the simpler single-repeat formulation and its smaller route family.

Table 18: **Selection over additional block repetitions on Qwen2.5-0.5B.** For each repetition count, *Block* is the block selected on the joint ARC and DART training set, while *ARC* and *DART* report its test accuracy. The oracle columns report route-set coverage for that repetition count. *Joint* selects both the block and repetition count from training data and still chooses the original one-repeat `sBLR` route.

| Extra repeats | Block | Steps | ARC | DART | ARC oracle | DART oracle |
|---|---|---|---|---|---|---|
| 1 | $(13, 21)$ | 33 | 55.6 | 7.5 | 98.4 | 33.0 |
| 2 | $(18, 21)$ | 32 | 54.2 | 7.0 | 99.6 | 32.7 |
| 3 | $(18, 21)$ | 36 | 52.5 | 6.0 | 99.6 | 32.1 |
| 4 | $(17, 19)$ | 36 | 54.3 | 3.2 | 99.7 | 30.8 |
| **Joint** | **(13,21)** | **33** | **55.6** | **7.5** | **100.0** | **42.2** |

# F EFFICIENCY

## F.1 RUNTIME AND OFFLINE SUPERVISION COSTS

**Hardware and measurement setup.** All experiments were run on the same cluster using NVIDIA GH200 GPUs. Each experimental run used a single GPU, with one experiment running per GPU. Inference measurements use batch size one and include prompt processing and greedy decoding with KV caching. For aBLR, latency also includes the preliminary pass and router computation. All methods use the same generation limits of 16 tokens for MCQ and 15 tokens for DART.

**Offline supervision cost.** Training the BLR policies first requires evaluating the complete route family on the training data. We evaluate 3,370 ARC and 3,000 DART examples, giving 6,370 training examples in total. Table 19 reports the resulting number of route evaluations and the wall time on one node with eight NVIDIA GH200 GPUs. Generating the complete supervision takes approximately 4.2 hours for Qwen2.5-0.5B, 6.7 hours for 1.5B, and 8.4 hours for 7B. This cost is incurred once offline and route enumeration is not required during inference.

Table 19: **Offline BLR supervision cost.** The route dataset contains 6,370 ARC and DART training examples. Route counts include the default route, and time reports node wall time on eight NVIDIA GH200 GPUs.

| Scale | Layers | Routes | Route evaluations | Hours |
|---|---|---|---|---|
| 0.5B | 24 | 277 | 1,764,490 | 4.16 |
| 1.5B | 28 | 379 | 2,414,230 | 6.67 |
| 7B | 28 | 379 | 2,414,230 | 8.36 |

**Dr.LLM baseline.** We follow the released skip, execute, and repeat configuration of Dr.LLM (Heakl et al., 2026). Its router uses hidden dimension 128 and eight pooling windows, and is trained with effective-number weighting ($\beta = 0.999$), focal loss ($\gamma = 2$), and AdamW for 25 epochs at learning rate $10^{-3}$. We change only the interface to Qwen2.5 and use the same datasets and evaluation protocol as for BLR. Dr.LLM reports 50 MCTS simulations per training example. We increase this budget to 200 simulations to provide stronger route supervision, matching the search budget used in our CoLa comparison (Li et al., 2025).

**Inference cost.** We compare the default model, sBLR, aBLR, and Dr.LLM on the same balanced set of 1,280 examples at each model scale. Table 20 reports FLOPs, latency, generation throughput, and peak GPU memory. sBLR has the lowest FLOPs among the routed methods because it only re-executes the selected block, although this does not always translate to the lowest measured latency. aBLR additionally performs a preliminary pass and router evaluation, which increases FLOPs and reduces throughput relative to the default at all three scales. Dr.LLM has slightly lower FLOPs and latency than aBLR in these profiles. Overall, the results show an accuracy–computation tradeoff rather than a runtime advantage from adaptive routing.

Repeated blocks reuse the same model parameters, so the additional cost mainly comes from executing more layer steps and storing the corresponding KV-cache entries. For the selected sBLR routes, the three model scales execute 33, 33, and 30 layer steps instead of 24, 28, and 28 for the default model. The additional memory is small in these profiles because model weights dominate the total footprint.

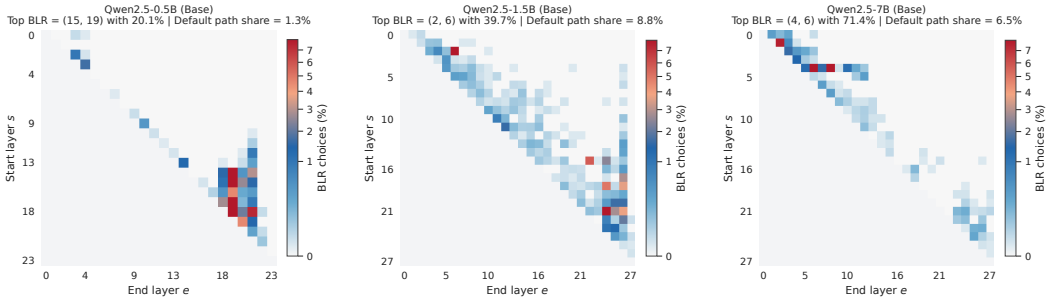
BLR FLOPs include the routed execution and, for aBLR, the preliminary pass and router computation.

Table 20: **Inference cost at batch size one.** FLOPs are reported in trillions per example, latency in seconds per example, throughput in generated tokens per second, and memory as peak GPU memory in MiB.

| Scale | Method | FLOPs | Latency | Throughput | Memory |
|---|---|---|---|---|---|
| 0.5B | Default | 0.116 | 0.244 | 61.02 | 1022.19 |
| | sBLR | 0.159 | 0.332 | 45.16 | 1022.60 |
| | aBLR | 0.242 | 0.319 | 46.18 | 1026.61 |
| | Dr.LLM | 0.216 | 0.280 | 52.69 | 1027.47 |
| 1.5B | Default | 0.363 | 0.314 | 48.82 | 3024.49 |
| | sBLR | 0.428 | 0.367 | 41.96 | 3026.08 |
| | aBLR | 0.742 | 0.386 | 38.48 | 3036.47 |
| | Dr.LLM | 0.682 | 0.361 | 42.24 | 3035.02 |
| 7B | Default | 1.793 | 0.303 | 50.71 | 14615.06 |
| | sBLR | 1.915 | 0.314 | 47.92 | 14615.06 |
| | aBLR | 3.526 | 0.358 | 41.92 | 14668.67 |
| | Dr.LLM | 3.391 | 0.348 | 43.94 | 14639.59 |

# G Qualitative Analysis

## G.1 Router Outputs




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

Figure 25: **aBLR choices across model scales.** The router assigns probability mass to multiple BLR blocks rather than collapsing to a single route.

Figs. 4 and 25 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.

## G.2 LLM Generation

This section provides representative generations for the formatting and answer changes discussed in Sec. 4.4. These examples illustrate output changes rather than decomposing the aggregate gain into causal components.

**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
```