# FlexBrain: A Unified Study of Routing in Conditional Computation

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

## Abstract

Conditional computation aims to improve neural network efficiency by routing each input through only the computation it needs. However, comparing routing strategies remains difficult because existing methods are typically evaluated in isolated frameworks, preventing standardized and fair comparisons. To address this, we introduce FlexBrain, a unified framework for comparing conditional computation methods under shared architectures, datasets and compute budgets. Using this standardized setup, we present an empirical analysis, comparing dense baselines and six conditional methods across language and vision tasks. In language modeling, we find that conditional computation is most beneficial when dense models are undertrained, while dense computation becomes increasingly competitive as training compute grows. In the vision domain, conditional computation reduces inference FLOPs while maintaining accuracy close to dense baselines. Overall, our results show that routing choices interact strongly with model scale and compute budget, and that unified evaluation is essential for making these tradeoffs explicit.

## 1 Introduction

The growing scale of modern neural networks has led to increasingly high computational demands during inference. Moreover, all inputs are processed by the same sequence of layers regardless of their complexity or importance Shazeer et al. (2017); Dehghani et al. (2018). This lack of input adaptation is an important limitation of traditional methods, as it is intuitive that not all inputs require the same level of processing: some inputs can be processed with relatively little computation, while others require the full model capacity. For example, in a language model, common, predictable words such as "the" or "and" may require only limited processing. At the same time, ambiguous tokens that depend on long-range context or reasoning may benefit from deeper, more extensive computation Raposo et al. (2024).

To address this limitation, conditional computation methods have emerged as a promising approach for dynamically allocating computation based on input complexity Bengio et al. (2013; 2015). Examples include Mixture-of-Experts, which route tokens to specialized modules (Shazeer et al., 2017; Fedus et al., 2022); early-exit methods, which stop computation once a prediction is sufficiently confident (Liu et al., 2020; Xin et al., 2020; Elhoushi et al., 2024; Scardapane et al., 2020); layer-skipping methods, which dynamically bypass layers (Raposo et al., 2024; Luo et al., 2025); and recursive models, which adaptively reuse the same layers multiple times (Dehghani et al., 2018; Banino et al., 2021; Bae et al., 2025). However, each proposed method typically adopts a different backbone architecture, parameter scale, training recipe and is rarely compared against existing methods under shared experimental settings Dhar & Søgaard (2026); Raposo et al. (2024); Chen et al. (2025); Bae et al. (2025). These choices are all important, but they make it difficult to attribute experimental outcomes to the routing strategy itself rather than to the surrounding architecture, optimization procedure, or evaluation protocol.

To help bridge this gap, we propose FlexBrain, a unified framework for studying conditional computation under shared experimental setup. We represent conditional computation as a dynamic neural graph (DNG), in which each input is routed through a subset of nodes. Rather than defining each routing method as a

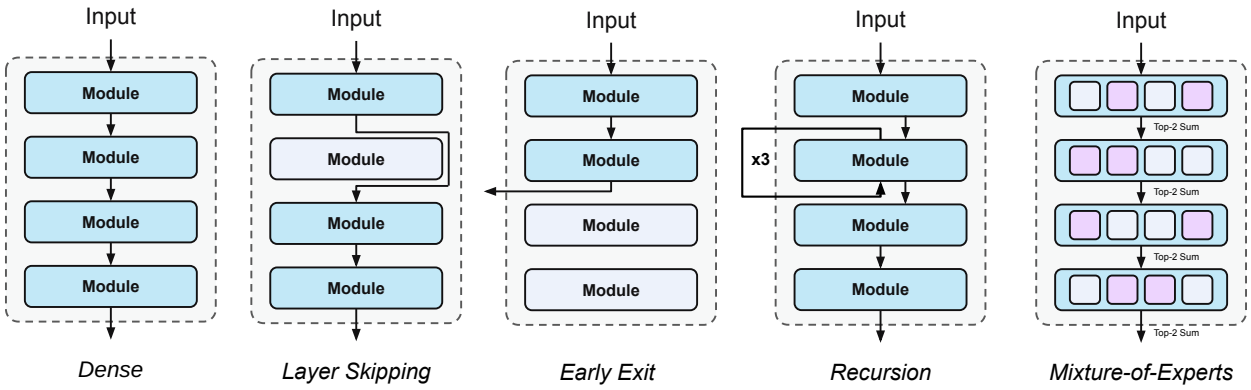


Figure 1: Overview of representative conditional computation methods. Dense models execute every module, while layer skipping, early exit, recursion, and mixture-of-experts adapt the execution path for each input.

separate architecture, FlexBrain factors routing into modular choices, including the available actions, the scoring function, the sampling rule, and the representation being routed.

Using FlexBrain, we study conditional computation in both language modeling and vision. For language modeling, we reimplement dense SmolLM2 backbones (Allal et al., 2025), Mixture-of-Depths (MoD) (Raposo et al., 2024), Mixture-of-Recursions (MoR) (Bae et al., 2025), and Inner Thinking Transformer (ITT) (Chen et al., 2025) under matched training-FLOP budgets on FineWeb-Edu Penedo et al. (2024). The results show that conditional computation performs better than dense models when trained under a compute-optimal budget, defined as 20 training tokens per parameter Hoffmann et al. (2022), while dense computation becomes more competitive as training compute increases. For vision, we compare dense DeiT backbones (Touvron et al., 2021), A-ViT (Yin et al., 2022), DynamicViT (Rao et al., 2021), and A-MoD (Gadhikar et al., 2025) on ImageNet-1K Deng et al. (2009). Conditional routing reduces inference FLOPs while keeping accuracy close to dense DeiT on Small and Base backbones, but the accuracy drop is larger on Tiny models. Overall, these results highlight the importance of evaluating different approaches under a shared experimental setting, as only controlled comparisons allow us to isolate the effect of the routing strategy itself. We provide the code for reproducibility at anonymous.4open.science

## 2 Related Work

Dense neural networks spend the same amount of computation on every input. Each example follows the full stack of layers, even when some examples are easy to classify, some tokens are redundant, or some patches carry little useful information. Conditional computation relaxes this fixed execution pattern. A model can route examples, tokens, or patches through only the parts of the network they need, reducing inference costs while allowing the full model to retain more capacity than is used for any single input. The main challenge is that routing decisions are often discrete, so training must handle non-differentiable choices (Bengio et al., 2013; 2015). Conditional computation has therefore developed into several related lines of work, including mixture-of-experts, early exit and token pruning, layer skipping, and layer recursion. Figure 1 illustrates several ways these methods change the computation executed for an input.

### 2.1 Mixture-of-Experts

Mixture-of-experts (MoE) is one of the most widely used forms of conditional computation. The early mixture-of-experts idea uses a gating network to combine specialized expert modules (Jacobs et al., 1991). Modern Transformer MoEs often place parallel experts inside feed-forward layers and route each token to a small subset of them (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). This increases total parameter capacity while keeping the active computation per token sparse. Routing can follow two complementary paradigms: token-choice routing, where each token selects its experts, or expert-choice routing,

where each expert selects which tokens from the batch to process (Zhou et al., 2022). These routing choices affect load balance, capacity use, and hardware efficiency. Recent large language models use MoE as a scaling tool Jiang et al. (2024); DeepSeek-AI et al. (2024; 2025); Team et al. (2025; 2026); Moonshot AI (2026); Team (2026).

### 2.2 Early Exit

This strategy reduces computation by terminating the processing of an input at an intermediate layer. Many approaches use a two-stage training process: pretraining a backbone, then fine-tuning auxiliary classification heads at various depths. During inference, efficiency is achieved through entropy-based heuristics: if the prediction confidence at an intermediate head exceeds a defined threshold, the remaining layers are skipped. This paradigm has been explored extensively in natural language processing Liu et al. (2020); Xin et al. (2020); Balagansky & Gavrilov (2022); Zhu (2021) and computer vision Teerapittayanon et al. (2016); Yin et al. (2022); Xu et al. (2023). Other modern variants leverage the STE or specialized gating modules to learn discrete exit policies natively during joint training Herrmann et al. (2020); Rao et al. (2021; 2023); Elhoushi et al. (2024); Bhuvaneswaran & Liu (2026); Shen et al. (2026).

### 2.3 Layer Skipping

Layer-skipping methods bypass intermediate blocks while keeping a final prediction head. SkipNet and BlockDrop learn input-dependent residual paths or block-skipping policies in convolutional networks (Wang et al., 2018; Wu et al., 2018). Vision Transformer variants also learn to adaptively use blocks, heads, or tokens to reduce computation (Meng et al., 2022; Liu et al., 2024; Kang & Lee, 2024). In language models, Mixture-of-Depths (MoD) routes only a fixed-capacity subset of tokens through selected Transformer blocks, while the other tokens bypass those blocks (Raposo et al., 2024). Adaptive layer-skipping methods for pretrained LLMs study related choices at inference or fine-tuning time (Luo et al., 2025; Li et al., 2025). A-MoD applies the MoD idea to Vision Transformers using attention scores from the previous layer instead of a learned router (Gadhikar et al., 2025).

### 2.4 Layer Recursion

Another way to change effective depth is to reuse computation. Parameter sharing across layers appears in ALBERT (Lan et al., 2020), while Universal Transformers apply a recurrent block over depth and use adaptive computation time for halting (Dehghani et al., 2018). PonderNet also learns input-dependent halting for iterative computation (Banino et al., 2021). Deep equilibrium models take a different route by solving for a fixed point of repeated computation (Bai et al., 2019). Recent work revisits recurrent depth for latent reasoning, test-time computation, and repeated hidden-state refinement (Geiping et al., 2025; Fan et al., 2025; Chen et al., 2025). Mixture-of-Recursions routes tokens through shared recursive blocks, allowing different tokens to receive different amounts of reused computation (Bae et al., 2025).

## 3 FlexBrain: A Unified Framework for Conditional Computation

We present FlexBrain, a unified framework for conditional computation. The goal is to separate the routing problem from the surrounding architecture. Instead of treating mixture-of-experts, early exit, layers skipping, layer recursion, and other routing methods as unrelated mechanisms, we describe each method using the same few choices: the available actions, the scoring function, the sampling rule, and the representation being routed. We call this shared abstraction a dynamic neural graph (DNG), in which a model contains a set of computational nodes and each input induces a route through those nodes.

### 3.1 A Dynamic Neural Graph

Consider a model with routable computation nodes $\mathcal{V} = \{v_1, \ldots, v_N\}$ and an exit action $\emptyset$. A hidden state $h_t$ is routed over a sequence of execution steps. Figure 2 illustrates one such routing step. The router first

produces scores

$$z_t = g_\theta(h_t, s_t) \in \mathbb{R}^{N+1},$$

where the first $N$ entries score computation nodes and the final entry scores exit. The routing state $s_t$ may include the current step, the active mask, and method-specific state accumulated along the route. A sampler converts these scores into an action,

$$a_t = \mathcal{S}(z_t), \qquad a_t \in \mathcal{A},$$

where $\mathcal{A}$ may be a one-hot next-node choice, a multi-hot top-$k$ selection, or a sparse assignment. A dispatcher executes the action,

$$h_{t+1} = \mathcal{D}(h_t, a_t; \mathcal{V}),$$

by applying the selected computation and updating the active state. The process stops when all active samples, patches, or tokens exit, or when a maximum route length $T_{\max}$ is reached. The route is the sequence

$$r = (a_1, \ldots, a_T), \qquad T \leq T_{\max}.$$

This object is more general than a feed-forward stack. Nodes need not correspond one-to-one with physical layers, need not be visited in depth order, and may be revisited. The same notation, therefore, covers fixed dense computation, layerwise conditional computation, and fully flexible DNG routes with jumps, loops, and logical nodes representing shared or composite blocks.

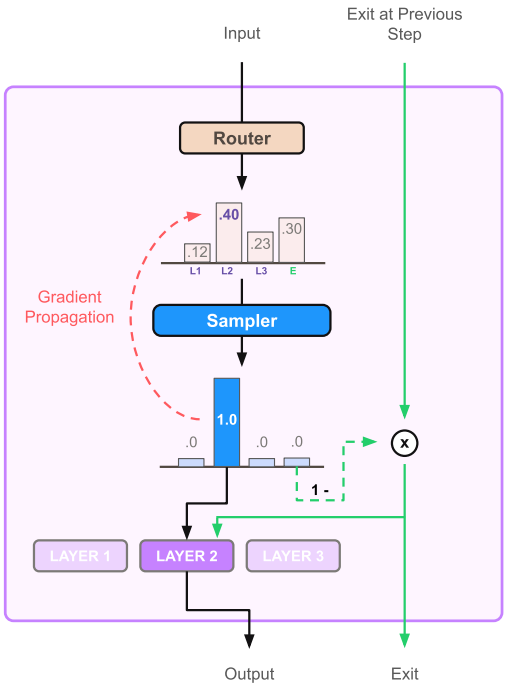


Figure 2: One routing step in the proposed Dynamic Neural Graph (DNG). Given the current hidden representation $h_t$ and routing state $s_t$ (*Input*), the router computes scores over all computation nodes and the exit action. A sampling rule selects the next action, and the dispatcher executes the corresponding computation to produce the updated representation $h_{t+1}$ (*Output*). This process repeats until an exit action is selected or the maximum route length is reached.

### 3.2 Routing Choices and Reproduced Methods

FlexBrain turns the DNG object into a framework by varying a small set of routing choices. The *action set* $\mathcal{A}_t(h_t) \subseteq \mathcal{V} \cup \{\emptyset\}$ determines which computations can be selected at step $t$. The *predictor* $g_\theta$ scores those actions, the *sampler* $\mathcal{S}$ turns scores into a discrete or sparse decision, and the *dispatcher* $\mathcal{D}$ sends the routed representation to the selected computation node. A chosen node may therefore correspond to the next layer in a stack, a parallel expert, a recursive use of a shared block, or a more general logical computation node. The route object also applies at different granularities. For instance, sample-wise routing assigns one route to each example, while token- or patch-wise routing assigns routes to individual tokens or patches. Routing can also be token-choice, where tokens choose nodes, or expert-choice, where nodes choose which tokens to process.

These choices recover the routing families we reproduce in this work. Dense computation fixes the route in advance as $r_{\text{dense}} = (v_1, \ldots, v_N, \emptyset)$. Early-exit and halting methods keep the dispatcher depth-wise but restrict the action set to $\mathcal{A}_t = \{v_{t+1}, \emptyset\}$, so the router only decides between the next computation node and exit. Layer-skipping and mixture-of-depths methods preserve depth order but choose an active token subset $M_t \subseteq \{1, \ldots, S\}$ for each conditional node $v_t$. Mixture-of-experts changes the available nodes at a fixed depth to parallel experts, $\mathcal{V}_t = \{e_{t,1}, \ldots, e_{t,E}\}$. Recursive methods use nodes such as $\mathcal{V} = \{r_1, \ldots, r_R\}$ for repeated applications of the same shared computation. The full-action-space DNG setting removes the layerwise restriction entirely by using $\mathcal{A}_t = \mathcal{V} \cup \{\emptyset\}$ at every step, allowing jumps, early termination, and node revisits. We leave a comprehensive study of MoE and full-action-space DNGs for future work.

# 4 Experimental Setup

We evaluate FlexBrain in two domains: autoregressive language modeling and image classification. Rather than treating the original implementations as fixed recipes, we reimplement representative approaches within common backbones, datasets, and compute budgets for each modality. These implementations are not intended to be exact replicas of every detail in the original papers. Instead, they provide a consistent setting for studying how different conditional computation strategies behave when many external design choices are held fixed.

## 4.1 Language Modeling

For language modeling, we train dense models, MoD, MoR, and ITT on FineWeb-Edu (Penedo et al., 2024) using 135M, 360M, and 730M SmolLM2-based backbones (Allal et al., 2025), matching the model scales used in Mixture-of-Recursions (Bae et al., 2025). We evaluate three training-FLOP budgets of $0.7\times10^{18}$, $2.0\times10^{18}$, and $4.0 \times 10^{18}$ FLOPs. Following the Chinchilla compute-optimal token allocation (Hoffmann et al., 2022), a dense model is allocated $20 \times N$ training tokens for $N$ parameters. These budgets therefore correspond to the compute-optimal token counts for the three dense baselines, with 2.7B tokens for 135M, 7.2B tokens for 360M, and 14.6B tokens for 730M. We budget by FLOPs rather than by token count because conditional models have a lower per-token cost than dense models. Under the same FLOP budget, a conditional model can process more tokens before using the full compute budget, which lets us compare loss-compute tradeoffs under the same resource constraint. We report validation loss as the main quality measure, path length as a measure of effective depth, and unique-path count as a measure of route diversity. We briefly describe the three conditional language-model variants reimplemented in FlexBrain below; full method details are available in the original papers.

**Mixture-of-Depths (MoD)** Raposo et al. (2024) adds layer skipping to the Transformer stack. In the configurations evaluated here, a routing module is placed at every other layer, so the model alternates between conditional MoD blocks and standard dense blocks. At each conditional block, an expert-choice top-$k$ router selects a fixed fraction of tokens to pass through the Transformer block, while the remaining tokens bypass it. The token capacity $\rho$ determines the active subset size $|M_t| = \rho S$ at each conditional block; we evaluate $\rho = 0.5$ and $\rho = 0.125$, corresponding to processing one half or one eighth of the tokens at each MoD block while the remaining tokens bypass it.

**Mixture-of-Recursions (MoR)** Bae et al. (2025) changes effective depth through parameter reuse rather than by skipping distinct layers. We use the middle-cycle strategy, where the first and last Transformer layers remain fixed, while tokens are routed through a shared recursive block for three passes. The capacity decreases across recursion steps: all tokens are processed in the first pass, two-thirds of the tokens are processed in the second pass, and one-third are processed in the third pass. During early training, these capacities are annealed from a less sparse setting toward the final targets. In our implementation, the 135M model uses 1 first + 9 shared + 1 last = 11 unique physical layers to obtain an unrolled depth of 29. The 360M model uses 12 unique layers for an unrolled depth of 32, and the 730M model uses 10 unique layers for an unrolled depth of 26.

**Inner Thinking Transformer (ITT)** Chen et al. (2025) routes selected tokens through an additional shared computation step, while unselected tokens bypass this extra processing. We use the same alternating layout as MoD, with conditional blocks at every other layer. At each conditional block, a top-$k$ router selects the tokens that enter one extra "thinking" step. We allow at most one additional step per conditional block.

## 4.2 Image Classification

For image classification, we evaluate A-ViT, DynamicViT, and A-MoD. All vision models use DeiT backbones (Touvron et al., 2021) at Tiny, Small, and Base scales and are trained on ImageNet-1K (Deng et al., 2009). We use top-1 and top-5 accuracy and inference G-FLOPs to measure the accuracy-compute tradeoff.

**A-ViT** Yin et al. (2022) is a token-halting method. Each token estimates whether it should continue to later layers or stop early. This can reduce inference cost, but halted tokens are removed from later computation

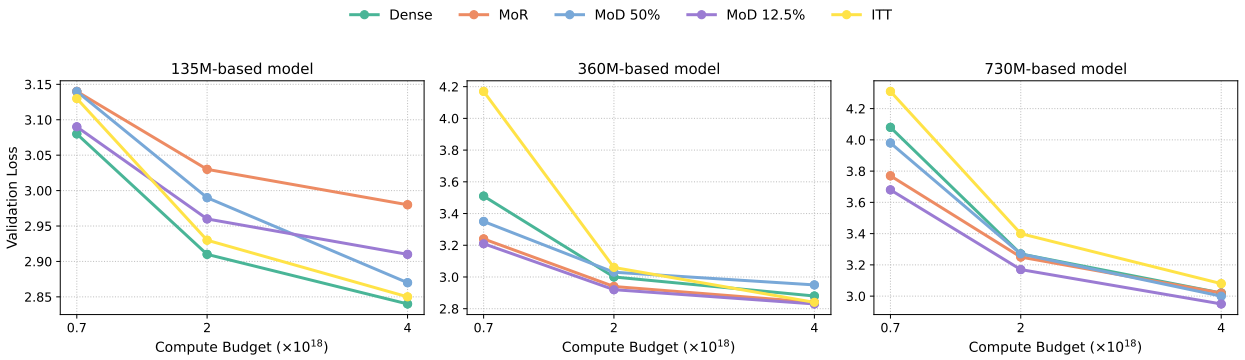


Figure 3: **Pareto scaling curves for conditional computation architectures across model scales.** The plots track performance across training compute budgets for three model capacity configurations. (*Left*) Begins at the 135M compute-optimal point and progresses into over-trained token regime. (*Center*) Initiates in a compute-suboptimal (under-trained) regime, crosses the 360M compute-optimal frontier, and ends in an overtrained regime. (*Right*) Traverses two compute-suboptimal regimes before reaching the 730M compute-optimal training limit.

and therefore do not receive downstream task gradients from subsequent layers. As a result, A-ViT relies on additional regularization, including a ponder loss and a distributional prior, to avoid halting too many tokens too early and to stabilize training.

**DynamicViT** Rao et al. (2021; 2023) is a token-pruning method that uses lightweight routing modules at several layers. Each router scores the current visual tokens and decides which tokens should be kept for later blocks. During training, tokens selected for pruning are blocked from interacting with other tokens, while the router can still be optimized end to end using the Gumbel-Softmax trick, a sparsity loss, and knowledge distillation from a dense teacher (i.e., the corresponding model without token pruning).

**A-MoD** Gadhikar et al. (2025) uses expert-choice routing without a learned router, relying instead on attention scores from the previous layer to select the top-$k$ tokens under a fixed capacity budget. Selected tokens pass through the next block, while unselected tokens bypass it through an identity connection. This makes A-MoD simpler to implement and train, because it does not require router losses, Gumbel relaxations, or distillation from the corresponding dense model.

## 5 Results on Language Modeling

Figure 3 compares dense models, MoD, MoR, and ITT across model sizes and training-compute budgets. The main pattern is that conditional computation is most helpful in the undertrained regime, where the available training compute is insufficient to reach full convergence in large dense models. For example, at the lowest compute budget of $0.7 \times 10^{18}$ FLOPs, larger dense baselines underperform sparse variants, reaching a validation loss of 3.51 for the 360M model. Under the same budget, MoD 12.5% ($\rho = 0.125$) reduces this to 3.21. Similar results appear also for the 730M model at both $0.7 \times 10^{18}$ and $2.0 \times 10^{18}$ FLOPs. This pattern is consistent with the evaluation benchmarks in Table 1, where conditional computation methods achieve higher accuracy than dense baselines. Overall, these results suggest that in low-compute regimes, sparse models can make better use of larger backbones than their dense counterparts.

This trend changes once the model receives more data than the compute-optimal allocation for its size. The performance advantage shifts back toward baselines that keep a larger computational active. In the 135M setting at $4.0 \times 10^{18}$ FLOPs, for example, the dense model reaches a validation loss of 2.84, while MoD 12.5% ($\rho = 0.125$) reaches 2.91. This behavior is also consistent with the evaluation benchmarks reported in Appendix 7, which provide the full set of evaluations corresponding to the experiments shown in Figure 3.

Figure 4 shows the same pattern through the number of active layers used by each method. The top row shows undertrained regimes, where the dense models receive less training compute than the compute-optimal

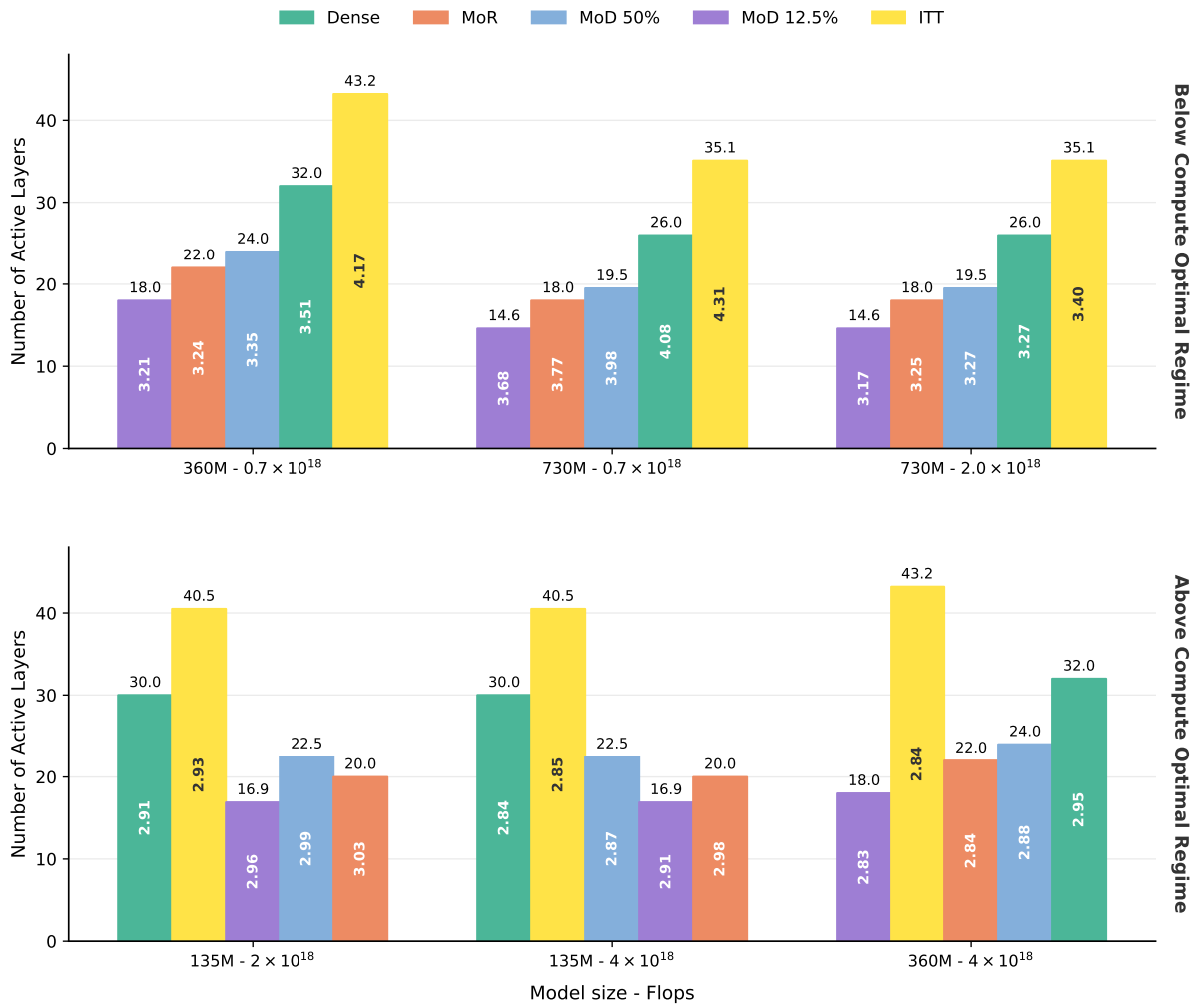


Figure 4: **Active-layer allocation across training regimes.** Each bar reports the average number of active layers used by a method. The top row shows *undertrained* regimes, where models receive less training compute than the compute-optimal allocation for their size, and the bottom row shows *overtrained* regimes. Within each model group, **the bars are ordered by validation loss, so the architecture on the far left achieves the lowest loss**. In undertrained regimes, the lowest loss is consistently coupled with fewer active layers (shorter effective paths), whereas in overtrained regimes this trend partially reverses and deeper active paths become competitive.

allocation for their size. In these cases, the lowest validation losses usually come from shorter effective paths, especially MoD 12.5%, which skips most conditional-block computation while still using the larger backbone. The bottom row shows overtrained regimes, in which dense, higher-compute routes become more competitive. For example, at 135M and $4.0 \times 10^{18}$ FLOPs, dense computation and ITT both reach better validation loss despite using many more active layers than MoD. Effective path length is therefore useful for interpreting the loss-compute tradeoff. In undertrained settings, a lower number of active layers often leads to lower loss, while in overtrained settings, methods that keep more computation active can become competitive.

### 5.1 The Effect of the Learned Router

To verify whether learned routers matter, we replace our learned gating mechanism with a random router (Table 2). Under random assignment, performance degrades, yielding validation losses worse than those of dense baselines at identical compute budgets. This confirms that conditional computation requires learned,

Table 1: Downstream few-shot benchmark evaluation and validation loss in the undertrained regime across varying parameter scales and FLOP budgets. In this regime, conditional computation achieves stronger downstream performance than comparable dense baselines.

| | | | | Few Shot Accuracy | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | **# Params** | **FLOPs** | **Val Loss** | **LD** | **HS** | **PQ** | **WG** | **ARC** | **MMLU** | **Avg** |
| Dense | 360M | $0.7 \times 10^{18}$ | 3.51 | 10.56 | 26.52 | **58.92** | **51.93** | 37.12 | 24.41 | 34.91 |
| MoR | 360M | $0.7 \times 10^{18}$ | 3.24 | **18.88** | **28.91** | 57.89 | 50.67 | **43.18** | **25.10** | **37.44** |
| MoD 12.5% | 360M | $0.7 \times 10^{18}$ | **3.21** | 14.17 | 26.98 | 57.29 | 50.67 | 39.06 | 24.53 | 35.45 |
| MoD 50.0% | 360M | $0.7 \times 10^{18}$ | 3.35 | 11.20 | 26.67 | 57.40 | 49.80 | 40.87 | 24.75 | 35.11 |
| ITT | 360M | $0.7 \times 10^{18}$ | 3.68 | 1.11 | 25.28 | 54.90 | 50.20 | 34.13 | 23.99 | 31.60 |
| Dense | 730M | $0.7 \times 10^{18}$ | 4.08 | 0.93 | 25.60 | 54.13 | 49.49 | 32.58 | 23.31 | 31.01 |
| MoR | 730M | $0.7 \times 10^{18}$ | 3.77 | **7.16** | 25.99 | 55.17 | **52.88** | **35.31** | 24.04 | **33.42** |
| MoD 12.5% | 730M | $0.7 \times 10^{18}$ | **3.68** | 3.36 | **26.36** | **55.88** | 50.91 | 34.97 | **24.38** | 32.64 |
| MoD 50.0% | 730M | $0.7 \times 10^{18}$ | 3.98 | 1.57 | 25.72 | 54.41 | 49.64 | 34.13 | 23.89 | 31.56 |
| ITT | 730M | $0.7 \times 10^{18}$ | 4.31 | 0.25 | 25.17 | 54.08 | 50.12 | 30.39 | 23.69 | 30.62 |
| Dense | 730M | $2.0 \times 10^{18}$ | 3.27 | 14.77 | 28.59 | 58.87 | 50.04 | **42.72** | 25.12 | 36.68 |
| MoR | 730M | $2.0 \times 10^{18}$ | 3.25 | **18.28** | 28.43 | 58.00 | **53.35** | 40.19 | **25.16** | **37.24** |
| MoD 12.5% | 730M | $2.0 \times 10^{18}$ | **3.17** | 12.38 | **28.93** | **59.30** | 49.96 | 41.62 | 25.08 | 36.21 |
| MoD 50.0% | 730M | $2.0 \times 10^{18}$ | 3.27 | 7.34 | 26.31 | 56.80 | 50.67 | 37.63 | 24.33 | 33.85 |
| ITT | 730M | $2.0 \times 10^{18}$ | 3.40 | 11.24 | 27.48 | 57.02 | 50.83 | 39.18 | 24.80 | 35.09 |

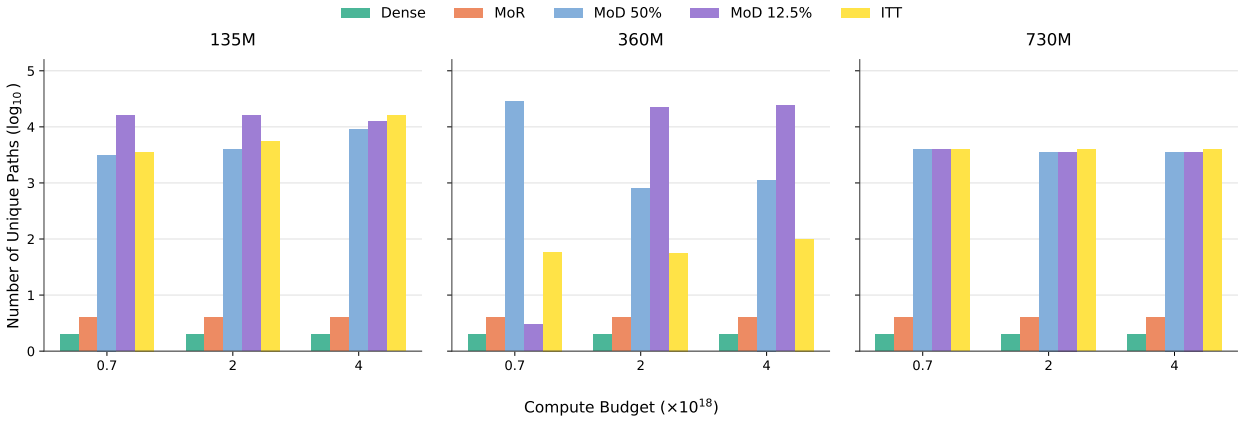


Figure 5: Unique path counts across model scales and training-FLOP budgets. The y-axis reports $\log_{10}(\text{unique paths} + 1)$, so larger values indicate greater route diversity. Dense and MoR have fixed or low route diversity, while MoD and ITT expose larger token-level routing spaces.

dynamic token routing to achieve compute-optimal scaling efficiency. To further assess whether the routers learn meaningful behavior, we examine qualitative examples (Figure 7 and Appendix 15). We observe that routing decisions tend to prioritize informative tokens, such as nouns and verbs, while assigning less weight to punctuation, such as ":", ".", and "?".

The diversity of the router is measured using the *unique path count* metric, that counts the number of different paths chosen by the tokens of the validation set (Table 2). A collapsed router leads to very few unique paths, whereas a random router explores nearly the entire combinatorial path space. In contrast, our models consistently operate in a healthy intermediate regime, balancing routing diversity and specialization. Relative to the maximum unique paths of MoD ($2^L$), the 360M and 730M models explore 2.38% and 90% of possible paths, corresponding to approximately 1,000 and 3,700 unique execution paths, respectively. These results indicate that both models learn expressive and well-distributed routing policies without collapsing or resorting to random exploration. Figure 5 compares the number of distinct routes used by different approaches. Dense models always follow a single execution path, MoR supports three possible paths, and

Table 2: Learned and random routing for MoD with token capacity $\rho = 0.5$. The learned routing never saturates the number of possible paths, whereas random routing covers nearly all of this empirical reference but consistently yields higher validation loss than learned routing.

| Model size | FLOPs | Unique paths (%) | | Val loss | |
|---|---|---|---|---|---|
| | | Learned | Random | Learned | Random |
| 135M | $0.7 \times 10^{18}$ | 19.83 | 99.99 | **3.29** | 3.31 |
| 135M | $2.0 \times 10^{18}$ | 25.16 | 99.99 | **2.99** | 3.05 |
| 135M | $4.0 \times 10^{18}$ | 56.46 | 100.00 | **2.87** | 2.96 |
| 360M | $0.7 \times 10^{18}$ | 88.81 | 99.94 | **3.35** | 4.40 |
| 360M | $2.0 \times 10^{18}$ | 2.38 | 99.94 | **3.03** | 4.17 |
| 360M | $4.0 \times 10^{18}$ | 3.46 | 99.94 | **2.95** | 3.15 |
| 730M | $0.7 \times 10^{18}$ | 97.39 | 100.00 | **3.98** | 4.35 |
| 730M | $2.0 \times 10^{18}$ | 90.50 | 100.00 | **3.27** | 3.40 |
| 730M | $4.0 \times 10^{18}$ | 88.79 | 100.00 | **3.00** | 3.09 |

MoD and ITT produce thousands to tens of thousands of distinct paths depending on model scale and training budget. Overall, the unique path count provides a simple quantitative measure of routing diversity and router health.

Finally, we investigate the training mechanisms responsible for preventing routing collapse. In Table 3, we assess the joint impact of auxiliary load-balancing losses $\mathcal{L}_{\text{aux}}$ and stability regularizers $\mathcal{L}_z$ on MoD. The auxiliary loss encourages the router to produce probabilities close to either 0 or 1, while the $z$-loss stabilizes the magnitudes of the router logits, preventing gradient instabilities during mixed-precision training. We observe a stronger stability effect for MoR. Without these regularizations, its routing becomes unstable and performance degrades. Overall, our ablation study demonstrates that both regularizers improve model performance across the configurations evaluated.

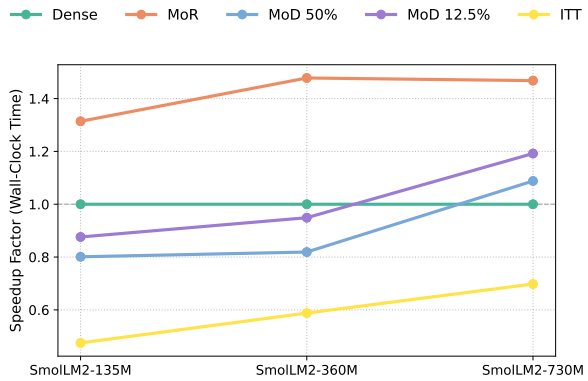


Figure 6: Validation wall-clock speedup relative to the dense baseline for the 135M, 360M, and 730M SmolLM2 backbones. Dense is normalized to 1.0, and values above 1.0 are faster.

| Aux-loss | z-loss | Path length | Val loss |
|---|---|---|---|
| x | x | 19.50 | 4.11 |
| ✓ | x | 19.50 | **3.98** |
| x | ✓ | 19.50 | **3.98** |
| ✓ | ✓ | 19.50 | 4.00 |
| x | x | 14.62 | 3.71 |
| ✓ | x | 14.62 | 3.70 |
| x | ✓ | 14.62 | **3.68** |
| ✓ | ✓ | 14.62 | **3.68** |

Table 3: Ablation study on the effect of auxiliary load-balancing loss and router $z$-loss across MoD 12.5% (below) and 50.0% (above) capacity.

### 5.2 Validation Wall-Clock Time

Figure 6 reports validation wall-clock speedup relative to the dense baseline for all the model sizes. We measure the elapsed time of a full validation pass over the same validation set, including model forward passes, validation metric computation, and distributed gathering. The speedup is the dense validation time

divided by the conditional-model validation time for the same model size, so values above 1.0 mean the conditional model is faster than dense.

MoR is the fastest method across model sizes because it groups multiple physical layers into a single block and applies this block only $N_r = 3$ times. This substantially reduces the number of router and dispatch calls. In contrast, MoD and ITT place routing modules at every other layer, causing routing to be invoked repeatedly throughout the network depth. For example, the 360M MoD backbone has 32 layers and therefore triggers 16 conditional routing operations, whereas MoR performs only three recursions. As a result, MoR incurs much lower wall-clock overhead than layer-by-layer conditional approaches.

The other conditional methods show a clear scale trend. MoD ($\rho = 0.5$) and MoD ($\rho = 0.125$) are slower than dense at 135M and 360M, but become faster than dense at 730M. This demonstrates that routing overhead becomes a minor factor at larger scales, because skipping expensive layer computations provides substantial time savings. ITT also improves with scale, but remains slower than dense, running up to 2× slower in our results, because it adds extra recurrent computation for selected tokens.

Figure 7: Token-wise routing visualization for the four different conditional computation approaches on one sample from the PIQA evaluation set. Colors indicate the effective path length traversed by each token.

## 6 Results on Image Classification

As shown in Table 4, our framework reproduces the three conditional methods in the vision setting and, in several configurations, achieves stronger performance. The first key finding is that conditional computation does not improve over the dense DeiT baseline in vision models, although it reduces inference FLOPs while maintaining accuracy close to the dense baseline. This is in contrast with language models, where sparse computation can be beneficial for larger undertrained dense models.

On DeiT-Small, our implementations closely reproduce the original results for DynamicViT and A-MoD Rao et al. (2021); Gadhikar et al. (2025). For A-ViT, our implementation exceeds the baseline accuracy reported in the original paper Yin et al. (2022), improving accuracy by +0.92% while requiring only 0.2 additional FLOPs. On the larger DeiT-Base backbone, we observe similar or stronger trends. Our A-MoD implementation improves over the results reported in the original paper by +0.6% for both capacity variants, and our A-ViT implementation achieves competitive performance, despite this setting not being explicitly evaluated in the original A-ViT paper. Our DynamicViT reproduction on DeiT-Base shows only a minor average accuracy deviation of −0.6%, consistent with the evaluation by Haurum et al. (2023).

In terms of computational efficiency, both DeiT-Small and DeiT-Base achieve a substantial reduction in FLOPs while maintaining high accuracy. For instance, the DeiT-Small achieves up to a 20% reduction in

Table 4: Comparison of conditional computation methods on ImageNet-1k. Our unified FlexBrain framework reproduces DynamicViT, A-ViT, and A-MoD across three DeiT backbones. Conditional computation consistently reduces inference FLOPs while maintaining accuracy close to the dense baseline, with the best trade-offs observed for DeiT-Small and DeiT-Base.

| Method | Approach | #Params | G-FLOPs | Top-1 Accuracy (%) | Top-5 Accuracy (%) |
|---|---|---|---|---|---|
| | | | **DeiT-Tiny** | | |
| Dense | - | 5M | 1.2 | 72.20 | 91.50 |
| A-ViT | Halting | 5M | 0.9 | 67.93 | 88.24 |
| DynamicViT | Dropping | 6M | 0.9 | 70.31 | 89.88 |
| A-MoD (50.0%) | Skipping | 5M | 0.9 | 70.43 | 89.92 |
| A-MoD (12.5%) | Skipping | 5M | 0.7 | 66.44 | 87.59 |
| | | | **DeiT-Small** | | |
| Dense | - | 22M | 4.6 | 79.80 | 95.00 |
| A-ViT | Halting | 22M | 3.8 | 79.52 | 94.93 |
| DynamicViT | Dropping | 24M | 3.7 | 79.60 | 94.79 |
| A-MoD (50.0%) | Skipping | 22M | 3.5 | 78.68 | 94.33 |
| A-MoD (12.5%) | Skipping | 22M | 2.6 | 76.94 | 93.32 |
| | | | **DeiT-Base** | | |
| Dense | - | 86M | 17.6 | 81.80 | 95.60 |
| A-ViT | Halting | 86M | 14.3 | 81.40 | 95.58 |
| DynamicViT | Dropping | 90M | 15.1 | 81.43 | 95.38 |
| DynamicViT | Dropping | 90M | 13.0 | 81.14 | 95.28 |
| DynamicViT | Dropping | 90M | 11.2 | 80.50 | 94.83 |
| A-MoD (50.0%) | Skipping | 86M | 13.2 | 81.00 | 95.32 |
| A-MoD (12.5%) | Skipping | 86M | 9.9 | 78.97 | 94.24 |

FLOPs with only a 0.3% drop in accuracy, while DeiT-Base benefits from a 24% reduction in FLOPs with a 0.4% accuracy decrease. To better understand how these savings arise, we visualize the learned routing policies on DeiT-Small in Figure 8, with additional examples provided in Appendix C. The visualizations show that conditional methods progressively reduce the number of active tokens as depth increases. Notably, the routers tend to preserve visually informative regions, such as the main subject, while removing less relevant background tokens. Overall, these results show that conditional computation can reduce resource consumption while preserving performance, and that the learned routing policies become increasingly selective in deeper layers.

DeiT-Tiny exposes a more challenging regime. First, the original DynamicViT paper does not provide a recipe for the smallest model, making direct comparison difficult. We also find that both A-ViT and A-MoD struggle to match their reported results on the DeiT-Tiny backbone, despite our efforts to explore different configurations and hyperparameter settings, as detailed in Appendix 8. In fact, with a 25% reduction in computation, the model loses 1.8% accuracy in the best case. These results, in contrast to what is observed in the Small and Base models, suggest that smaller architectures such as DeiT-Tiny have less representational capacity, making them more sensitive to token-dropping and layer-skipping decisions.

### 6.1 The Impact of Pruned Tokens on Training

DynamicViT uses a Gumbel-Softmax relaxation to allow gradients from pruned tokens to flow back to the router. We find that hard-masking inactive tokens achieves 79.25% top-1 accuracy, whereas preserving gradient flow through the DynamicViT relaxation improves accuracy to 80.50%, as shown in Table 5. This suggests that, in DynamicViT, maintaining a differentiable path for pruned tokens is important for effectively training the router. Interestingly, this behavior differs from what we observe in our language experiments in Section 5, where tokens pruned from the router are removed from the computational graph and do not provide gradients to improve routing decisions. This might be due to the routing strategy. Expert-choice routing selects tokens from the current active sequence at each routing step, whereas DynamicViT uses token-choice routing, where each token makes an independent decision. As a result, expert-choice routing may introduce implicit interactions among active tokens, which act as a form of regularization.

Table 5: DynamicViT ablation on attention masking

| Method | Dead Tokens Gradient | G-FLOPs | Top-1 Accuracy (%) | Top-5 Accuracy (%) |
|---|---|---|---|---|
| DynamicViT-Base | No | 11.2 | 79.25 | 94.05 |
| DynamicViT-Base | Yes | 11.2 | 80.50 | 94.83 |

Figure 8: Qualitative analysis of conditional computation approaches on ImageNet-1k.

## 7 Conclusion

In this work, we address the lack of standardized evaluation practices in conditional computation by introducing FlexBrain, a framework for studying conditional computation through a shared routing formulation, making routing methods easier to compare. Our experiments show that routing choices interact strongly with model scale and compute budget. In language modeling, conditional computation is most useful when models are trained below the compute-optimal budget, revealing that the number of active layers is correlated with better performance. On the other hand, dense models become more competitive as the training compute increases. Therefore, MoD, MoR, and ITT represent different tradeoffs rather than a single best recipe. In vision, A-ViT, DynamicViT, and A-MoD reduce inference FLOPs on ImageNet-1K DeiT backbones, but none of the conditional models improves over the dense baseline. The accuracy drop is smaller on DeiT-Small and DeiT-Base than on DeiT-Tiny, suggesting that conditional computation helps more with bigger backbone capacity. Overall, our findings indicate that the effectiveness of routing strategies depends on both model scale and compute budget, making unified evaluation crucial for exposing these tradeoffs.

## References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martin Blazquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Agustín Piqueres Lajarín, Hynek Kydlíček, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan Son NGUYEN, Ben Burtenshaw, Clémentine Fourrier, Haojun Zhao, Hugo Larcher, Mathieu Morlon, Cyril Zakka, Colin Raffel, Leandro Von Werra, and Thomas Wolf. SmolLM2: When smol goes big — data-centric training of a fully open small language model. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=3JiCl2A14H.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025. URL https://arxiv.org/abs/2507.10524.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/01386bd6d8e091c2ab4c7c7de644d37b-Paper.pdf.

Nikita Balagansky and Daniil Gavrilov. PALBERT: Teaching ALBERT to ponder. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_WQ6XkVP23f.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL https://openreview.net/forum?id=1EuxRTe0WN.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *CoRR*, abs/1511.06297, 2015. URL http://arxiv.org/abs/1511.06297.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Ramshankar Bhuvaneswaran and Handan Liu. Bitskip: An empirical analysis of quantization and early exit composition in transformers, 2026. URL https://arxiv.org/abs/2510.23766.

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 Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1369. URL https://aclanthology.org/2025.acl-long.1369/.

DeepSeek-AI et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI et al. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. *ArXiv*, abs/1807.03819, 2018. URL https://api.semanticscholar.org/CorpusID:49667762.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Ruchira Dhar and Anders Søgaard. Evaluation revisited: A taxonomy of evaluation concerns in natural language processing, 2026. URL https://arxiv.org/abs/2604.25923.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.681. URL https://aclanthology.org/2024.acl-long.681/.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=2edigk8yoU.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL http://jmlr.org/papers/v23/21-0998.html.

Advait Gadhikar, Souptik Kumar Majumdar, Niclas Popp, Piyapat Saranrittichai, Martin Rapp, and Lukas Schott. Attention is all you need for mixture-of-depths routing. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*, 2025. URL https://openreview.net/forum?id=1uDP4ld3eZ.

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

Joakim Bruslund Haurum, Sergio Escalera, Graham W. Taylor, and Thomas B. Moeslund. Which tokens to use? investigating token reduction in vision transformers. In *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 773–783, 2023. doi: 10.1109/ICCVW60793.2023.00085.

Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. Channel selection using gumbel softmax. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*, pp. 241–257, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58582-2. doi: 10.1007/978-3-030-58583-9_15. URL https://doi.org/10.1007/978-3-030-58583-9_15.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Robert Jacobs, Michael Jordan, Steven Nowlan, and Geoffrey Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 03 1991. doi: 10.1162/neco.1991.3.1.79.

Albert Q. Jiang et al. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Woochul Kang and Hyungseop Lee. Adaptive depth networks with skippable sub-paths. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=NPu7Cdk2f9.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. {GS}hard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? test-time depth adaptation of pretrained llms, 2025. URL https://arxiv.org/abs/2507.07996.

Junzhang Liu, Tingkai Liu, Yueyuan Sui, and Stephen Xia. Elastiformer: Learned redundancy reduction in transformer via self-distillation, 11 2024.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. FastBERT: a self-distilling BERT with adaptive inference time. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6035–6044, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.537. URL https://aclanthology.org/2020.acl-main.537/.

Xuan Luo, Weizhi Wang, and Xifeng Yan. Adaptive layer-skipping in pre-trained LLMs. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Gu0XSax2YS.

Lingchen Meng, Hengduo Li, Bor-Chun Chen, Shiyi Lan, Zuxuan Wu, Yu-Gang Jiang, and Ser-Nam Lim. Adavit: Adaptive vision transformers for efficient image recognition. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12299–12308, 2022. doi: 10.1109/CVPR52688.2022.01199.

Moonshot AI. Kimi k2.6: Scaling agentic intelligence with trillion-parameter mixture-of-experts. Technical report, Moonshot AI, April 2026. URL https://www.kimi.com/blog/kimi-k2-6. Open-weight model with 32B active parameters and 256K context window.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 13937–13949. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/747d3443e319a22747fbb873e8b2f9f2-Paper.pdf.

Yongming Rao, Zuyan Liu, Wenliang Zhao, Jie Zhou, and Jiwen Lu. Dynamic spatial sparsification for efficient vision transformers and convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10883–10897, 2023. doi: 10.1109/TPAMI.2023.3263826.

David Raposo, Sam Ritter, Blake A. Richards, Timothy P. Lillicrap, Peter C. Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *ArXiv*, abs/2404.02258, 2024. URL https://api.semanticscholar.org/CorpusID:268876220.

Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12, 09 2020. doi: 10.1007/s12559-020-09734-4.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR (Poster)*. OpenReview.net, 2017. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2017.html#ShazeerMMDLHD17.

Yixian Shen, Qi Bi, Zihan Wang, Zhiheng Yang, Changshuo Wang, Zhi Zhang, Prayag Tiwari, Andy D. Pimentel, and Anuj Pathania. Efficient multimodal spatial reasoning via dynamic and asymmetric routing. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=BQASoLmREU.

Core Team et al. Mimo-v2-flash technical report, 2026. URL https://arxiv.org/abs/2601.02780.

Gemma Team et al. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Qwen Team. Qwen3.5-omni technical report, 2026. URL https://arxiv.org/abs/2604.15804.

Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016. doi: 10.1109/ICPR.2016.7900006.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10347–10357. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/touvron21a.html.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 01 2018. doi: 10.48550/arXiv.1711.08393.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic early exiting for accelerating BERT inference. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.204. URL https://aclanthology.org/2020.acl-main.204/.

Guanyu Xu, Jiawei Hao, Li Shen, Han Hu, Yong Luo, Hui Lin, and Jialie Shen. Lgvit: Dynamic early exiting for accelerating vision transformer. *arXiv preprint arXiv:2308.00255*, 2023.

Hongxu Yin, Arash Vahdat, Jose M. Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10809–10818, June 2022.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Y Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V Le, and James Laudon. Mixture-of-experts with expert choice routing. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=jdJo1HIVinI.

Wei Zhu. LeeBERT: Learned early exit for BERT with cross-level optimization. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 2968–2980, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.231. URL https://aclanthology.org/2021.acl-long.231/.

# A Extended Experimental and Implementation Details

## A.1 General Setup

All language models (135M, 360M, and 730M scales) are trained from scratch on an infrastructure consisting of 4× NVIDIA H100 (80GB) GPUs. Optimization is conducted using AdamW with a base learning rate $\eta_0$ configured per architecture scale, utilizing bfloat16 mixed precision and a gradient clipping threshold of 1.0. The details of the SmolLM2 architecture are reported in Table 6.

Table 6: Key architectural hyperparameters of the foundational dense model size variants utilized as backbones in our unified evaluation framework. All configurations reflect the base hyperparameter choices of the open-source SmolLM2 family (Allal et al., 2025), leveraging Grouped-Query Attention (GQA) to minimize key-value cache overhead.

| | **Base Configuration** | | | | **Attention & Feed-Forward** | | | | **Input** | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Models** | N-emb | Emb | $N_L$ | $d_{model}$ | $N_{head}$ | $N_{KV}$ | $d_{head}$ | $d_{inter}$ | Vocab | $L_{ctx}$ |
| Vanilla 135M | 106M | 28M | 30 | 576 | 9 | 3 | 64 | 1536 | 49K | 2K |
| Vanilla 360M | 315M | 47M | 32 | 960 | 15 | 5 | 64 | 2560 | 49K | 2K |
| Vanilla 730M | 654M | 75M | 26 | 1536 | 24 | 8 | 64 | 4096 | 49K | 2K |

## A.2 FLOP-Conditioned Learning Rate Schedule

We use the same forward-FLOP accounting rule for dense and conditional language models. Before training, the code profiles each model with `FlopCounterMode` at batch size one and the training context length. This gives $F_{\text{pre}}$ for computation before routing, $F_{\text{block}}$ for one Transformer block, $F_{\text{post}}$ for computation after routing, and, for conditional models, $F_{\text{router}}$ for one router call. During training and evaluation, the route object $r = (a_1, \ldots, a_T)$ gives the batch mean path length $\bar{L}$. For routers that score an action at each routed step, the per-sample forward FLOPs are

$$F_{\text{sample}} = F_{\text{pre}} + \bar{L}\,(F_{\text{block}} + F_{\text{router}}) + F_{\text{post}}.$$

For routers whose decision is made outside the repeated block computation, we use

$$F_{\text{sample}} = F_{\text{pre}} + \bar{L}F_{\text{block}} + F_{\text{router}} + F_{\text{post}}.$$

The batch contribution is $BF_{\text{sample}}$. We use these forward-pass FLOPs to decide when a run has reached its target training budget and to log the accumulated training compute. Actual training executes additional backward-pass work, but forward FLOPs provide a single architecture-level unit that is comparable across dense models and models whose active path length changes with the route.

The language recipes stop training when the accumulated forward training FLOPs reach the target budget. The same accumulated FLOP counter drives the trapezoidal learning-rate schedule. Let $T_{\text{FLOP}}$ be the target forward training-FLOP budget, $W = 0.05T_{\text{FLOP}}$ the warmup budget, $C = 0.20T_{\text{FLOP}}$ the cooldown budget, $P = T_{\text{FLOP}} - W - C$ the plateau budget, and $f$ the accumulated forward training FLOPs. The learning-rate multiplier is

$$\lambda(f) = \begin{cases} \frac{1}{W} + \frac{f}{W}\left(1 - \frac{1}{W}\right), & f < W, \\ 1, & W \le f \le W + P, \\ 1 + (\lambda_{\min} - 1)\frac{f - W - P}{C}, & W + P < f < T_{\text{FLOP}}, \\ \lambda_{\min}, & f \ge T_{\text{FLOP}}. \end{cases}$$

For the language experiments, $\lambda_{\min} = 0$ unless otherwise specified.

## A.3 Evaluation on NLP Benchmarks

To support the analysis presented in Section 5, in Table 7, we evaluate model performance using few-shot accuracy on six benchmarks from the Language Model Evaluation Harness: LAMBADA (LD), HellaSwag

Table 7: Downstream benchmark evaluation and validation loss across varying parameter scales and FLOP budgets. Architectural variants are grouped by active parameter and compute tiers, reporting performance on standard few-shot tasks along with the aggregate benchmark average.

| | | | | Few Shot Accuracy | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | **# Params** | **FLOPs** | **Val Loss** | **LD** | **HS** | **PQ** | **WG** | **ARC** | **MMLU** | **Avg** |
| Dense | 135M | $0.7 \times 10^{18}$ | **3.08** | 19.44 | 29.20 | 60.34 | 50.75 | 43.43 | 25.26 | 38.07 |
| MoR | 135M | $0.7 \times 10^{18}$ | 3.14 | 21.44 | 29.97 | 59.74 | 51.54 | 43.14 | 25.32 | 38.53 |
| MoD 12.5% | 135M | $0.7 \times 10^{18}$ | 3.09 | **22.01** | **30.19** | 61.10 | **52.01** | **44.44** | 25.70 | **39.24** |
| MoD 50.0% | 135M | $0.7 \times 10^{18}$ | 3.14 | 21.09 | 29.15 | 59.79 | 49.01 | 43.69 | 25.12 | 37.98 |
| ITT | 135M | $0.7 \times 10^{18}$ | 3.13 | 20.49 | 29.73 | **61.32** | 49.96 | 43.73 | **25.94** | 38.53 |
| Dense | 360M | $0.7 \times 10^{18}$ | 3.51 | 10.56 | 26.52 | **58.92** | **51.93** | 37.12 | 24.41 | 34.91 |
| MoR | 360M | $0.7 \times 10^{18}$ | 3.24 | **18.88** | **28.91** | 57.89 | 50.67 | **43.18** | **25.10** | **37.44** |
| MoD 12.5% | 360M | $0.7 \times 10^{18}$ | **3.21** | 14.17 | 26.98 | 57.29 | 50.67 | 39.06 | 24.53 | 35.45 |
| MoD 50.0% | 360M | $0.7 \times 10^{18}$ | 3.35 | 11.20 | 26.67 | 57.40 | 49.80 | 40.87 | 24.75 | 35.11 |
| ITT | 360M | $0.7 \times 10^{18}$ | 3.68 | 1.11 | 25.28 | 54.90 | 50.20 | 34.13 | 23.99 | 31.60 |
| Dense | 730M | $0.7 \times 10^{18}$ | 4.08 | 0.93 | 25.60 | 54.13 | 49.49 | 32.58 | 23.31 | 31.01 |
| MoR | 730M | $0.7 \times 10^{18}$ | 3.77 | **7.16** | 25.99 | 55.17 | **52.88** | **35.31** | 24.04 | **33.42** |
| MoD 12.5% | 730M | $0.7 \times 10^{18}$ | **3.68** | 3.36 | **26.36** | **55.88** | 50.91 | 34.97 | **24.38** | 32.64 |
| MoD 50.0% | 730M | $0.7 \times 10^{18}$ | 3.98 | 1.57 | 25.72 | 54.41 | 49.64 | 34.13 | 23.89 | 31.56 |
| ITT | 730M | $0.7 \times 10^{18}$ | 4.31 | 0.25 | 25.17 | 54.08 | 50.12 | 30.39 | 23.69 | 30.62 |
| Dense | 135M | $2.0 \times 10^{18}$ | **2.91** | **27.69** | **34.01** | **63.28** | 50.99 | **49.24** | **26.58** | **41.96** |
| MoR | 135M | $2.0 \times 10^{18}$ | 3.03 | 24.22 | 31.84 | 61.86 | 49.33 | 45.03 | 26.05 | 39.72 |
| MoD 12.5% | 135M | $2.0 \times 10^{18}$ | 2.96 | 25.77 | 31.95 | 62.84 | 48.62 | 46.51 | 25.89 | 40.26 |
| MoD 50.0% | 135M | $2.0 \times 10^{18}$ | 2.99 | 25.85 | 31.72 | 61.43 | **51.93** | 46.93 | 26.09 | 40.66 |
| ITT | 135M | $2.0 \times 10^{18}$ | 2.93 | 25.33 | 33.07 | 62.62 | 50.75 | 47.69 | 26.45 | 40.98 |
| Dense | 360M | $2.0 \times 10^{18}$ | 3.00 | 24.82 | 32.07 | 62.30 | **53.59** | 46.68 | 25.89 | 40.89 |
| MoR | 360M | $2.0 \times 10^{18}$ | 2.94 | **26.92** | **33.69** | 61.59 | 50.36 | **48.57** | 26.41 | **41.25** |
| MoD 12.5% | 360M | $2.0 \times 10^{18}$ | **2.92** | 25.89 | 33.10 | **62.68** | 49.88 | 47.18 | **26.47** | 40.87 |
| MoD 50.0% | 360M | $2.0 \times 10^{18}$ | 3.03 | 22.84 | 30.66 | 61.26 | 50.91 | 46.21 | 25.84 | 39.62 |
| ITT | 360M | $2.0 \times 10^{18}$ | 3.06 | 0.19 | 29.52 | 59.90 | 50.51 | 40.78 | 24.10 | 34.17 |
| Dense | 730M | $2.0 \times 10^{18}$ | 3.27 | 14.77 | 28.59 | 58.87 | 50.04 | **42.72** | 25.12 | 36.68 |
| MoR | 730M | $2.0 \times 10^{18}$ | 3.25 | **18.28** | 28.43 | 58.00 | **53.35** | 40.19 | **25.16** | **37.24** |
| MoD 12.5% | 730M | $2.0 \times 10^{18}$ | **3.17** | 12.38 | **28.93** | **59.30** | 49.96 | 41.62 | 25.08 | 36.21 |
| MoD 50.0% | 730M | $2.0 \times 10^{18}$ | 3.27 | 7.34 | 26.31 | 56.80 | 50.67 | 37.63 | 24.33 | 33.85 |
| ITT | 730M | $2.0 \times 10^{18}$ | 3.40 | 11.24 | 27.48 | 57.02 | 50.83 | 39.18 | 24.80 | 35.09 |
| Dense | 135M | $4.0 \times 10^{18}$ | **2.84** | **30.41** | **36.04** | **65.61** | 50.91 | **50.29** | **27.11** | **43.40** |
| MoR | 135M | $4.0 \times 10^{18}$ | 2.98 | 25.46 | 32.48 | 62.68 | 50.59 | 46.59 | 26.39 | 40.70 |
| MoD 12.5% | 135M | $4.0 \times 10^{18}$ | 2.91 | 28.47 | 32.74 | 64.25 | 50.43 | 47.26 | 27.10 | 41.71 |
| MoD 50.0% | 135M | $4.0 \times 10^{18}$ | 2.87 | 27.71 | 34.71 | 63.33 | 51.14 | 48.95 | 26.93 | 42.13 |
| ITT | 135M | $4.0 \times 10^{18}$ | 2.85 | 29.48 | 35.38 | 64.09 | **53.20** | 49.75 | 26.93 | 43.14 |
| Dense | 360M | $4.0 \times 10^{18}$ | 2.88 | **29.65** | **36.50** | 64.20 | 50.99 | 50.93 | 27.05 | **43.22** |
| MoR | 360M | $4.0 \times 10^{18}$ | 2.84 | 28.47 | 35.69 | 64.04 | 50.91 | 48.61 | 26.95 | 42.44 |
| MoD 12.5% | 360M | $4.0 \times 10^{18}$ | **2.83** | 28.95 | 35.54 | **65.07** | 50.67 | **51.35** | **27.28** | 43.14 |
| MoD 50.0% | 360M | $4.0 \times 10^{18}$ | 2.94 | 28.53 | 33.13 | 62.57 | 51.46 | 48.61 | 26.29 | 41.76 |
| ITT | 360M | $4.0 \times 10^{18}$ | 2.84 | 29.48 | 35.37 | 64.74 | **51.78** | 49.41 | 26.92 | 42.95 |
| Dense | 730M | $4.0 \times 10^{18}$ | 3.02 | 17.64 | 30.11 | 61.10 | 49.96 | 44.19 | 25.71 | 38.12 |
| MoR | 730M | $4.0 \times 10^{18}$ | 3.02 | **22.90** | **31.65** | 60.99 | 51.30 | 45.71 | 25.53 | 39.68 |
| MoD 12.5% | 730M | $4.0 \times 10^{18}$ | **2.95** | 20.65 | 31.63 | **62.08** | 51.14 | **46.59** | **26.22** | **39.72** |
| MoD 50.0% | 730M | $4.0 \times 10^{18}$ | 3.00 | 18.65 | 30.42 | 61.32 | **53.43** | 45.24 | 25.85 | 39.15 |
| ITT | 730M | $4.0 \times 10^{18}$ | 3.08 | 20.03 | 30.04 | 61.37 | 49.25 | 44.32 | 25.63 | 38.44 |

(HS), PIQA (PQ), WinoGrande (WG), ARC (Easy and Challenge), and MMLU. Following the standard evaluation protocol, we normalize accuracy by the byte length of the target string for all benchmarks except LAMBADA, WinoGrande, and MMLU. We also use the standard number of shots for each benchmark and evaluate MMLU using the continuation task for simplicity. Each point in Figure 3 has a corresponding row in Table 7, which provides the full set of autoregressive evaluation results across all benchmarks.

## B A-ViT Hyperparameter Ablation

We perform a hyperparameter sweep for A-ViT on DeiT-Tiny using ImageNet-1k, to better understand the gap observed on this backbone. Table 8 reports the effect of varying the learning rate, ponder loss weight, and distributional prior loss weight.

Across settings, A-ViT tends to preserve a high average path length, often using nearly the full depth of the backbone. Configurations that force shorter paths, mainly through stronger ponder regularization, consistently reduce accuracy. We do not observe a setting that achieves a clear accuracy benefit under substantially reduced computation. These results suggest that, on DeiT-Tiny, reducing the effective path length limits representation learning and leads to a direct performance drop.

Table 8: Hyperparameter sweep for **A-ViT / DeiT-Tiny** on ImageNet-1k. We vary the learning rate, ponder loss weight, and distributional prior loss weight, and report the resulting average path length and top-1 accuracy. All runs use 100 training epochs, prior mean 8.0, prior standard deviation 0.02, gate scale 5.0, and gate center 10.0.

| Learning Rate | Ponder Loss Weight | Distributional Weight | Path Length | Accuracy (%) |
|---|---|---|---|---|
| 0.0002 | 0.0005 | 0.001 | 11.50 | 71.99 |
| 0.0005 | 0.0005 | 0.001 | 8.69 | 67.41 |
| 0.0002 | 0.0010 | 0.001 | 8.12 | 66.36 |
| 0.0005 | 0.0010 | 0.001 | **7.61** | 66.63 |
| 0.0002 | 0.0001 | 0.010 | 11.83 | 72.03 |
| 0.0005 | 0.0001 | 0.010 | 11.63 | 71.25 |
| 0.0002 | 0.0005 | 0.010 | 11.63 | **72.08** |
| 0.0005 | 0.0005 | 0.010 | 10.92 | 71.02 |
| 0.0002 | 0.0010 | 0.010 | 11.07 | 71.97 |
| 0.0005 | 0.0010 | 0.010 | 10.01 | 70.19 |
| 0.0002 | 0.0001 | 0.100 | 11.81 | 72.02 |
| 0.0005 | 0.0001 | 0.100 | 11.61 | 71.22 |
| 0.0002 | 0.0005 | 0.100 | 11.59 | 71.98 |
| 0.0005 | 0.0005 | 0.100 | 11.02 | 70.99 |
| 0.0002 | 0.0010 | 0.100 | 11.17 | 71.94 |
| 0.0005 | 0.0010 | 0.100 | 10.36 | 70.67 |
| 0.0005 | 0.0010 | 0.100 | 10.95 | 71.15 |

## C Qualitative Evaluation of Learned Routing Policies

To better understand the behavior of the learned routing functions, we perform a qualitative analysis of the routing decisions produced by the different architectures. Across tasks, we observe that the routers learn to allocate computation selectively, identifying the most informative components of the input while suppressing redundant information. Despite their distinct optimization dynamics and architectural constraints, the resulting routing policies exhibit consistent patterns that align with task-relevant structures. The following sections provide qualitative analyses for both image classification and language modeling.

### C.1 Image Classification

We visualize spatial token selection across intermediate layers to compare the routing behavior of the vision models. Across all three methods, the learned policies tend to preserve tokens corresponding to the foreground object while removing or skipping less informative background regions. This suggests that, despite their different routing mechanisms, the methods learn semantically meaningful token selection policies.

The methods differ mainly in how sparsity emerges across depth. A-ViT concentrates most token exits around the middle layers, consistent with its ponder loss and distributional prior. DynamicViT follows its predefined pruning schedule, removing tokens in discrete stages at layers 3, 6, and 9. A-MoD shows a more adaptive pattern where early layers retain broader spatial context, while later layers focus more strongly on the main object. Even under a 12.5% capacity limit, A-MoD still produces coherent routing maps that preserve visually relevant regions.

### C.2 Language Modeling

In language modeling, we visualize token-wise routing behavior on PIQA examples under teacher forcing, using SmolLM2-730M trained on $0.7 \times 10^{18}$ tokens. Unlike the vision setting, where routing can be inspected spatially, language-model routing shows how computation is distributed across tokens in a sequence. We report the effective *path length*, defined as the number of blocks visited by each token during inference. Blue tokens correspond to shorter paths, while red tokens correspond to longer paths.

Figures 12–15 reveal several consistent routing patterns across all the paradigms presented. Across all models, punctuation and function words such as ":", ".", "and", and "or" generally receive shorter computational paths, whereas structurally important tokens such as nouns, entities ("Ivan Drago" and "Rocky"), and verbs tend to receive increased computation. This suggests that the learned routing policies successfully identify semantically informative portions of the sequence.

In MoR, we observe several consistent patterns. Question marks tend to receive longer computational paths, and the model increasingly emphasizes informative tokens such as nouns, verbs, and adjectives. The two MoD variants operate in different sparsity regimes. MoD 12.5% aggressively reduces computation, producing shorter path lengths for most tokens, as reflected by the predominance of blue tokens. In contrast, MoD 50% skips fewer tokens and therefore maintains longer effective paths, leading to a warmer color distribution.

ITT adopts a different strategy. Rather than skipping computation, it dynamically increases the number of executed layers by revisiting blocks multiple times. As a result, its path lengths are generally larger and more uniform across the sequence, indicating that computation is allocated through iterative refinement rather than token dropping.

Overall, despite their different mechanisms, all four approaches learn meaningful routing policies that focus computation on semantically relevant tokens while reducing computation on less informative parts of the input.

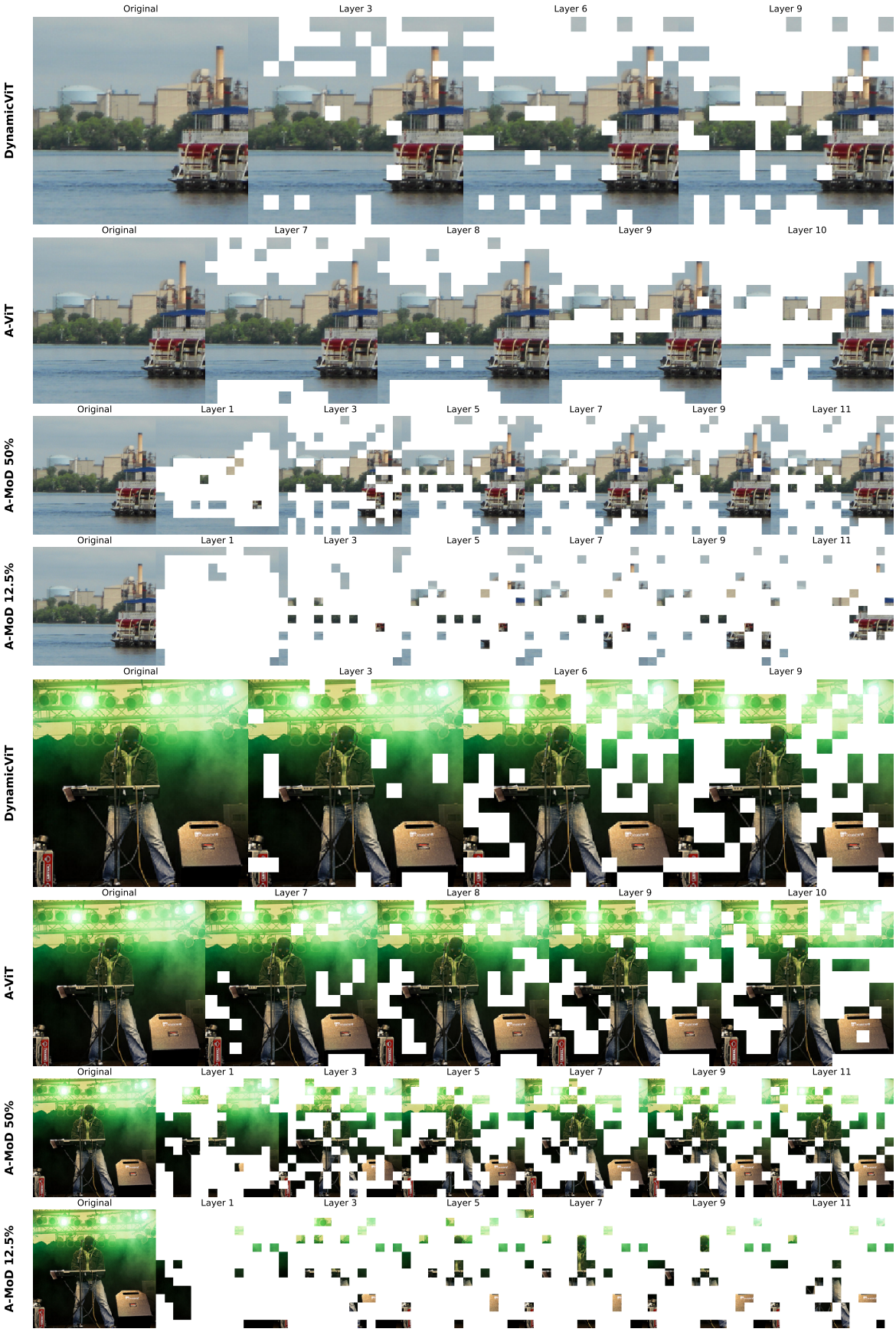


Figure 9: Qualitative analysis of conditional computation approaches on ImageNet-1k.

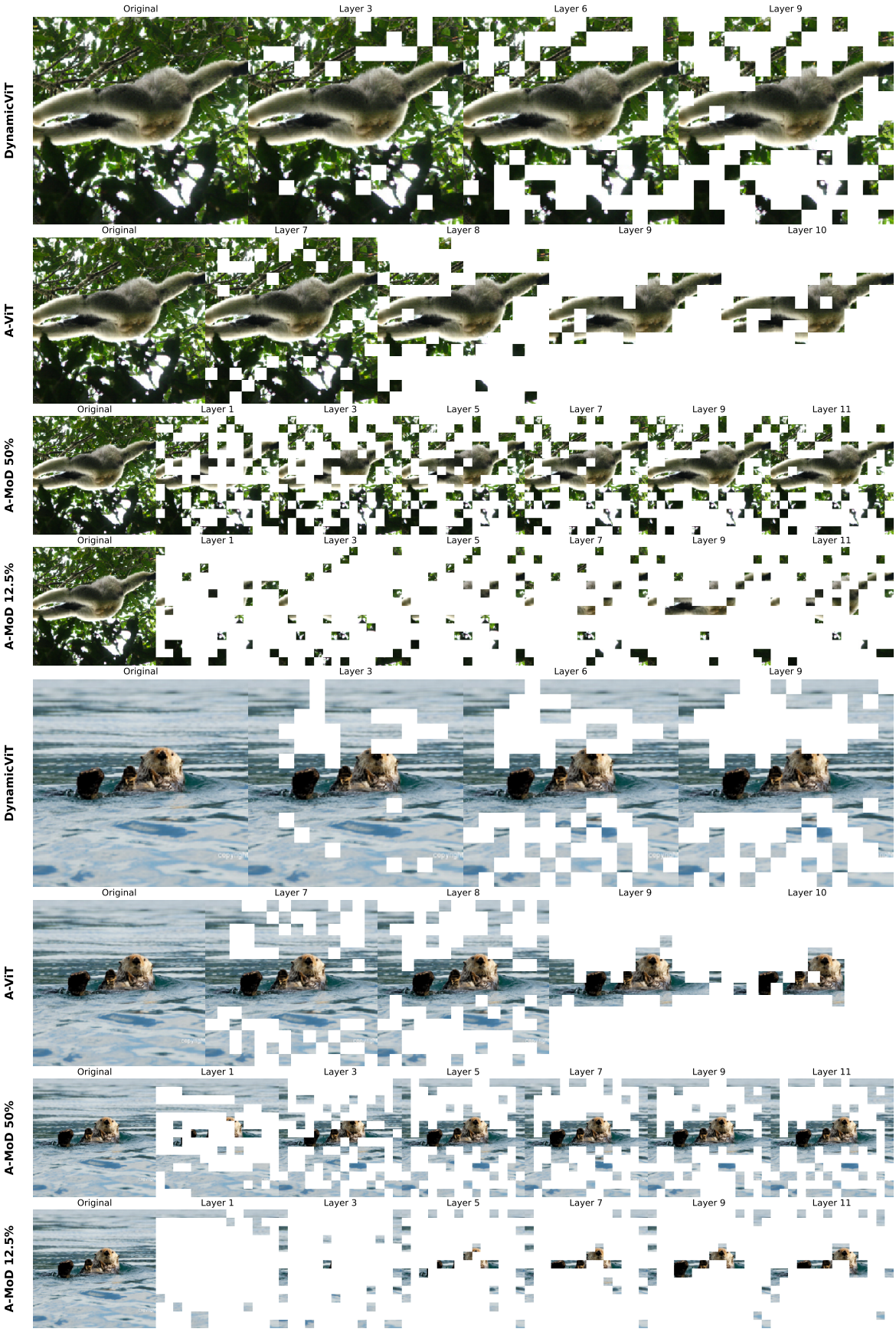


Figure 10: Qualitative analysis of conditional computation approaches on ImageNet-1k.

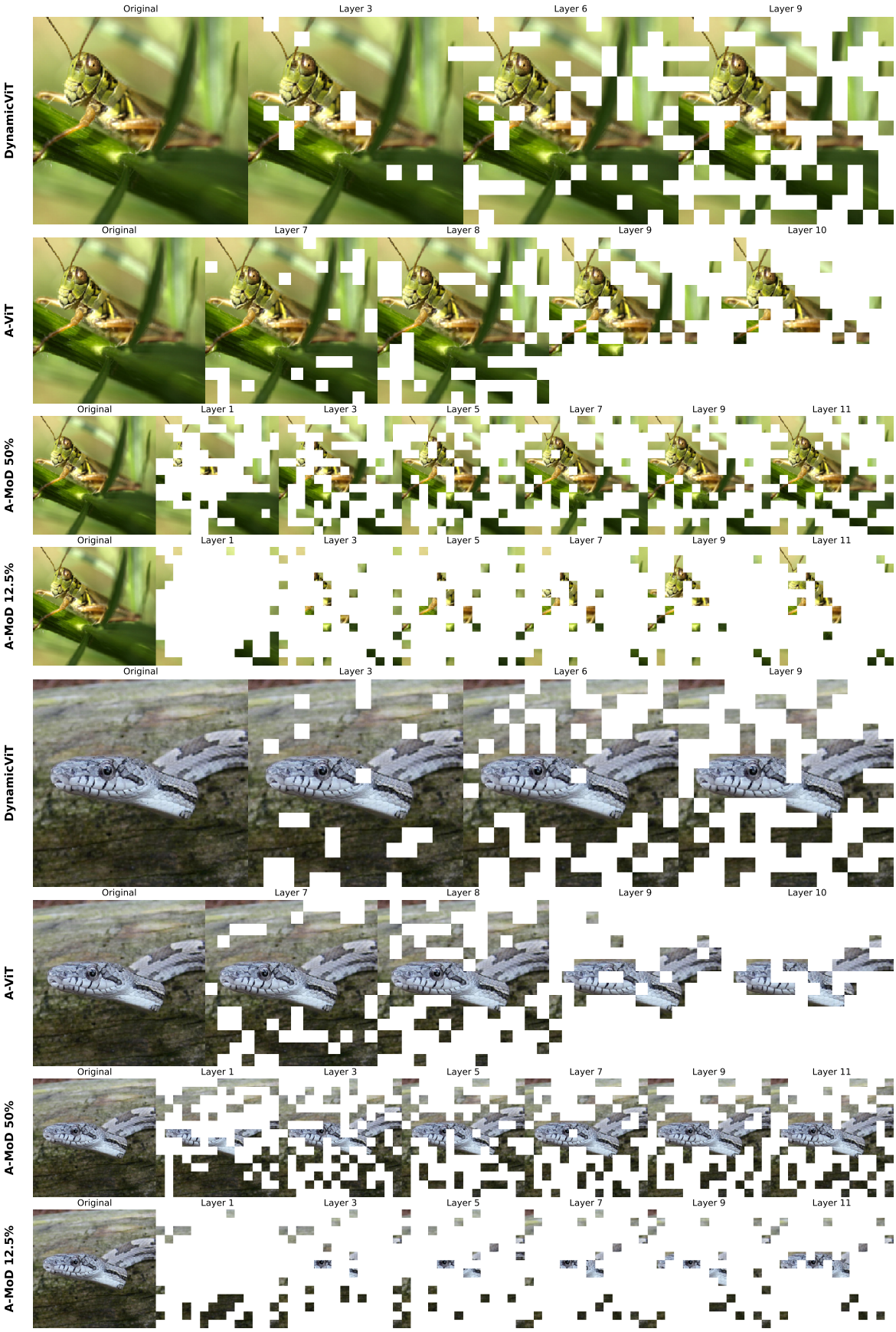


Figure 11: Qualitative analysis of conditional computation approaches on ImageNet-1k.

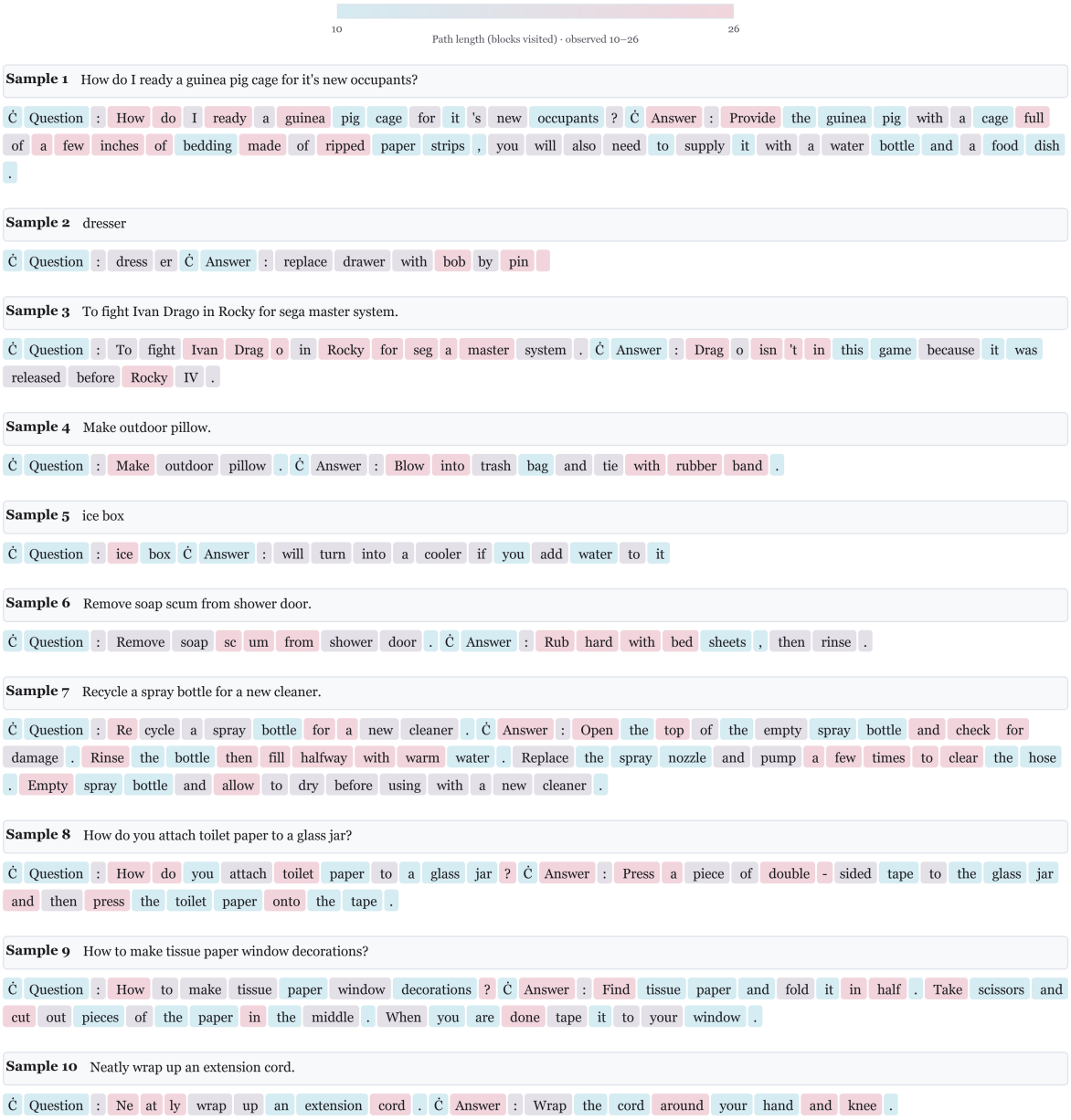


Figure 12: Token-wise routing visualization for MoR on samples from the PIQA evaluation set. Colors indicate the effective path length traversed by each token. In this example, the router can select only three distinct paths, resulting in effective path lengths of 10, 18, or 26 layers.

Figure 13: Token-wise routing visualization for MoD 12.5% on samples from the PIQA evaluation set. Colors indicate the effective path length traversed by each token.

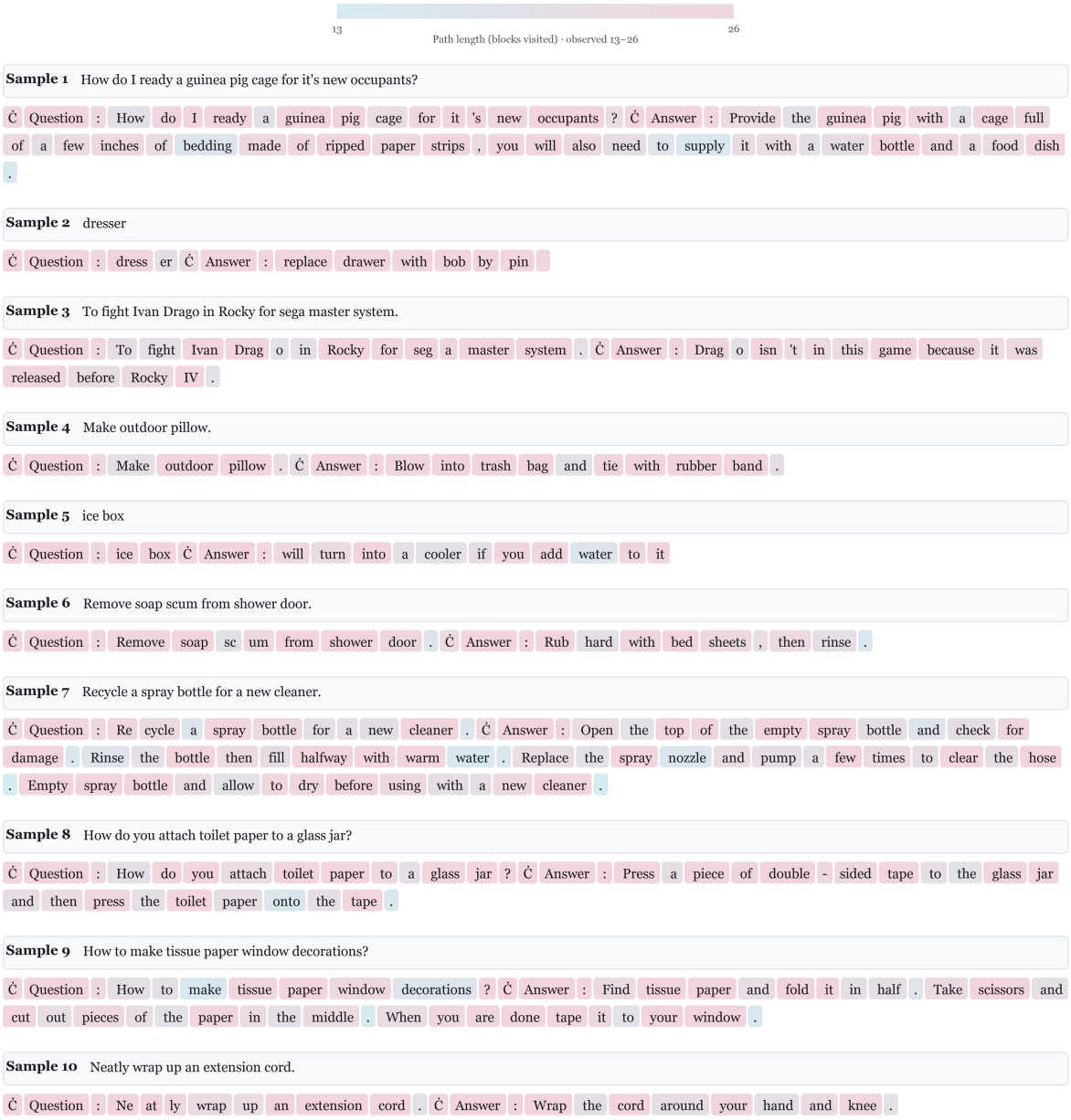


Figure 14: Token-wise routing visualization for MoD 50% on samples from the PIQA evaluation set. Colors indicate the effective path length traversed by each token.

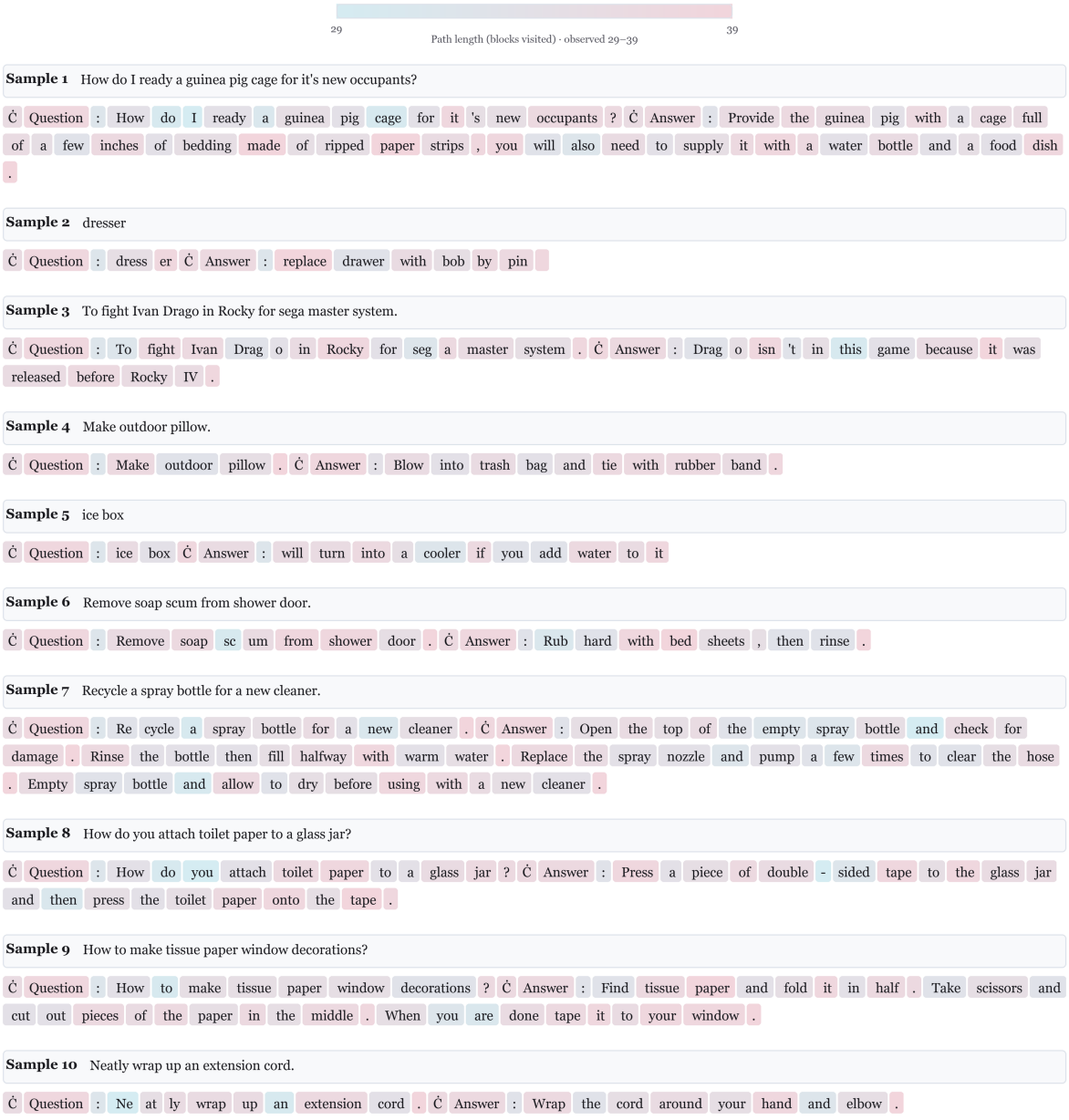


Figure 15: Token-wise routing visualization for ITT on samples from the PIQA evaluation set. Colors indicate the effective path length traversed by each token.