DHKR: DATA-DEMANDED CAPACITY CURRICULA FOR HIERARCHICAL REPRESENTATION ROUTING IN FROZEN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

While adapting language models often relies on rigid capacities and static routing structures, we introduce Dynamic Hierarchical Knowledge Routing (DHKR), a general framework for representation adaptation on frozen backbones. DHKR treats structural intervention availability as a control variable distinct from input-conditioned allocation and parameter optimization. Using a Knowledge Steering Layer (KSL) as an external portfolio of nonlinear representation interventions, DHKR uses a rule-based controller to retain or release paths within a configured intervention portfolio. Counterfactual analyses show that replacing learned weights with uniform or averaged variants increases validation loss, while restricted availability yields lower validation loss than full-from-start baselines under a matched regularized protocol. These findings show that allocation, structural availability, and Controller execution produce distinct outcomes in the tested setting. Separate trajectories verify that training continues after portfolio release.

1 INTRODUCTION

The Lottery Hedge Fund Hypothesis (LHFH) Kawamae (2025) models adaptation through a portfolio of latent subnetworks that contribute with input-dependent weights. In this relational framework, knowledge represents the relative difference between a source model and target data rather than isolated semantic expertise. The functional role of each intervention path depends on the frozen backbone, target distribution, current input, and co-active pathways, treating each path as a latent-representation pivot rather than an independent expert.

While large language models achieve strong general performance, adapting them to specialized domains typically requires full-model fine-tuning, which can be computationally costly and limit throughput efficiency. Parameter-Efficient Fine-Tuning (PEFT) methods—including adapters Houlsby et al. (2019), LoRA Hu et al. (2022), BitFit Zaken et al. (2022), and QLoRA Dettmers et al. (2023)—significantly reduce the number of trainable parameters used to adapt a pretrained model, whereas representation-intervention methods instead learn transformations of hidden states while retaining a frozen backbone Wu et al. (2024). Knowledge-Aligned Domain Shift Tuning (KADA) Kawamae (2025) addresses domain adaptation by reusing latent knowledge fragments under LHFH, instantiating a portfolio view through a fixed set of sub-networks with input-conditioned weighting, where each token is modeled as a weighted sum of sub-networks, suggesting that dynamically varying this count can enhance LLM capabilities. While existing Mixture-of-Experts (MoE) Jordan & Jacobs (1994); Shazeer et al. (2017); Fedus et al. (2022); Guo et al. (2025); Muennighoff et al. (2025) and dynamic-architecture methods primarily focus on internal execution efficiency and routing, they lack a unified framework for data-driven capacity growth governed by explicit stability controls.

To bridge this gap, **Dynamic Hierarchical Knowledge Routing (DHKR)** studies a complementary control paradigm: an external portfolio of representation interventions attached to a frozen backbone. Rather than altering internal computations, DHKR separates input-conditioned allocation, structural intervention availability, and parameter optimization in an external intervention portfolio. DHKR builds upon the KSL intervention architecture while addressing a key question left open by

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

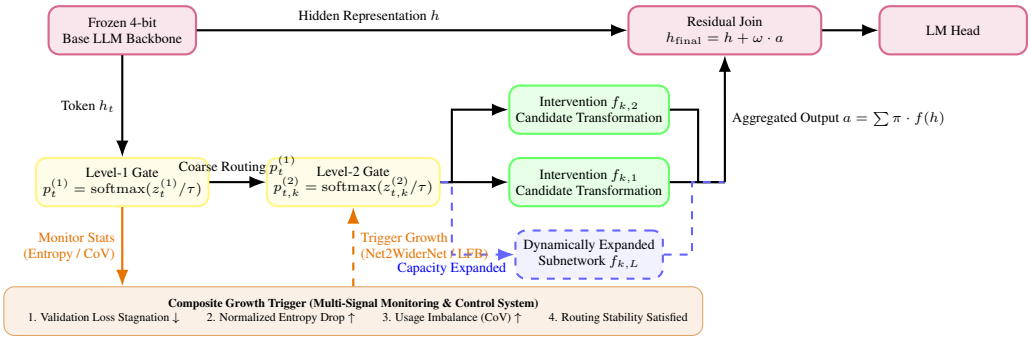

Figure 1: Overall architecture of DHKR. A frozen 4-bit backbone produces hidden states h, and an external Knowledge Steering Layer (KSL) applies two-level routing over subnetworks. The first-level router allocates weight across available intervention families, while the second-level router optionally refines a selected parent when a finer-grained bottleneck is detected. A composite growth trigger monitors validation stagnation, routing entropy, usage imbalance, and routing stability, enabling demand-driven capacity expansion without modifying the frozen backbone.

static formulations: which intervention paths should be structurally available to routing and optimization during training? Our contributions are summarized as follows:

- We formulate structural intervention availability as a control variable distinct from input-conditioned allocation and parameter optimization in frozen-backbone representation adaptation.
- We instantiate this formulation with an external intervention portfolio whose active set can be retained or released during training.
- Same-model counterfactuals show that preserving the input-to-weight correspondence improves validation loss in one trained realization.
- In a matched regularized comparison, restricted availability yields lower validation loss than a full-from-start portfolio across three seeds.
- Separate transition trajectories show that newly released paths receive routing mass and that optimization continues after structural transitions.

2 Previous Work

Weight-space and representation-space adaptation. Weight-space PEFT methods constrain trainable parameter updates, whereas representation-intervention methods modify the hidden states of a frozen backbone. DHKR operates in the latter setting, focusing on portfolio allocation and structural availability rather than parameter minimality.

Conditional computation and dynamic model capacity. Conditional-computation approaches route inputs through subsets of parameters. Drawing on BASE Layers Lewis et al. (2021), DHKR employs a two-level router with bias-based load smoothing and multi-level balancing across K and L subnetworks. Following KADA, DHKR treats Knowledge Steering Layer (KSL) components as intervention paths rather than semantic networks, avoiding assumptions of isolated knowledge specialties. Furthermore, dynamic capacity methods govern network expansion and routing stability. Dynamic MoE methods can adapt expert activation and expert count during training Guo et al. (2025). DHKR applies related routing and transition concepts to an external representation-intervention portfolio, and utilizes routing statistics for path release and LFB-style routing-bias updates Wang et al. (2024) to ensure smooth structural transitions. While traditional conditional computation and dynamic MoE methods adapt internal experts, DHKR adapts these routing and capacity mechanisms to serve a different control object: external residual interventions attached to a frozen representation.

Portfolio transitions. Network-morphism and routing-balancing methods motivate warm-started path initialization and transition control. DHKR uses these as implementation components rather than as independent novelty claims, leveraging them to ensure stable and efficient structural adaptation during portfolio allocation.

3 FEEDBACK-CONTROLLED AVAILABILITY OF REPRESENTATION INTERVENTIONS

3.1 OVERVIEW

Our primary contribution lies in the separation of intervention parameters, input-conditioned weighting, and active-set availability. DHKR operationalizes this separation through a configurable intervention portfolio. At each training stage, only the active portfolio participates in composition and optimization. The current implementation organizes the portfolio along two axes, where first-level release expands breadth and second-level release adds parent-local transformation freedom. We do not assume that hierarchy itself improves prediction quality, nor that paths correspond to predefined topics, domains, or semantic networks.

DHKR combines an input-conditioned router with a rule-based active-set controller. The controller can retain the current portfolio or release an additional path when its configured conditions are met. Unlike prior work Guo et al. (2025); Jordan & Jacobs (1994); Lewis et al. (2021); Mustafa et al. (2022); Dai et al. (2022); Fedus et al. (2022), the controller observes validation and routing statistics. Transition operations—incorporating temperature control, LFB Lepikhin et al. (2021); Wang et al. (2024)-style routing-bias adjustment, history reset, bounded residual blending, and cooldown—prevent volatility and support structured adaptation.

To steer the base LLM toward dynamic knowledge pivot alignment, DHKR integrates three core mechanisms:

1. **Portfolio Release**: Changes which configured intervention paths participate in routing and optimization.
2. **Portfolio Allocation**: Computes input-conditioned weights over the currently available paths.
3. **Transition Control**: Updates routing state and initialization when the available portfolio changes.

To validate stable dynamic capacity expansion in sub-networks, DHKR evaluates both cross-entropy-only (CE-only) and regularized objectives. The original Controller-versus-Full experiments utilize the CE-only realization, whereas the subsequent four-way identification experiment applies the same regularized objective across the Controller, Static, Scheduled, and Full conditions (comparing conditions strictly within each objective family without cross-objective loss comparisons).

3.2 PROBLEM SETTING AND FINAL REPRESENTATION BLENDING

We study representation adaptation with a frozen, quantized backbone. DHKR does not target trainable-parameter minimality; it controls the execution and optimization exposure of an external intervention portfolio. Given a frozen, 4-bit quantized pretrained LLM backbone, we inject an external, trainable KSL immediately after the final hidden representation $h \in \mathbb{R}^{B\times S\times H}$. Since this layer is directly appended to the base model, no additional hooks or complex adapter management are required. Unlike KADA's fixed integration, DHKR blends KSL outputs with the frozen model's representation through explicit latent interventions while preserving the pretrained backbone weights, scaling automatically via an EMA-smoothed norm ratio. Rather than strictly minimizing trainable parameters, DHKR's primary objectives are intervention allocation and control over active intervention availability, providing a different placement of trainable computation from weight-space PEFT.

The final blended representation at token t is formulated as:

$$h_{\text{final},t} = h_t + \omega_t \cdot a_t, \qquad \omega_t = \text{EMA}\left(\text{clip}\left(\gamma \cdot \frac{\|a_t\|_2}{\|h_t\|_2}, 0, \omega_{\max}\right)\right)_t.$$

where a_t is the aggregated subnetwork output, ω_t is the dynamically adjusted intervention strength based on the norm ratio, and $\|\cdot\|$ denotes the ℓ_2 norm over the hidden dimension. Exponential Moving Average (EMA) and a ramp-up schedule prevent over-intervention during early training phases. The scalar hyperparameters—namely the scaling factor γ, the clipping ceiling $\omega_{\max}$, the EMA decay rate, and the total ramp-up steps N—are detailed in the supplementary material.

3.3 Hierarchical Routing Architecture: Gating and Subnetworks

We extend KADA's flat dynamic weighting into a two-level hierarchy: a first-level router allocates weight across candidate intervention families, while a second-level router optionally adds local transformation freedom within a selected parent. These paths are not assumed to correspond to predefined domains, concepts, or semantic experts. Routing stability is handled separately by the mechanisms described below. The KSL employs a two-level gating mechanism over subnetworks. The hierarchical $K \times L$ structure provides two axes for expanding the available intervention function class, while routing stabilizers—including Loss-Free Balancing (LFB), temperature annealing, Trust-Region (TR) regularization, and Posterior Regularization (PR) Ganchev et al. (2010)—prevent volatility and load imbalance. For each token $h_t \in \mathbb{R}^H$:

$$z_t^{(1)} = g^{(1)}([h_t \parallel \bar{p}]) \in \mathbb{R}^{K_{\max}}, \quad p_t^{(1)} = \mathrm{softmax}\left(\frac{z_t^{(1)}}{\tau}\right),$$
$$z_{t,k}^{(2)} = g_k^{(2)}(h_t) \in \mathbb{R}^{L_{\max}}, \quad p_{t,k}^{(2)} = \mathrm{softmax}\left(\frac{z_{t,k}^{(2)}}{\tau}\right).$$

The final mixture weight for subnetwork (k, l) is:

$$\pi_{t,k,l} = p_{t,k}^{(1)} \cdot p_{t,k,l}^{(2)}, \qquad \sum_{k,l} \pi_{t,k,l} = 1.$$

At inference, we optionally apply top-k masking at each level (e.g., $k_1 = 1, k_2 = 1$) to reduce FLOPs. The temperature τ is annealed from τ_{start} to τ_{end} over the first M steps to transition from exploratory to deterministic routing. During training, we use Gumbel-Softmax relaxation Maddison et al. (2017); Jang et al. (2017) for differentiable sampling.

Each subnetwork $f_{k,l} : \mathbb{R}^H \to \mathbb{R}^H$ is a 2-layer MLP with layer norm. Its output is:

$$a_t = \sum_{k=1}^{K_{\text{act}}} \sum_{l=1}^{L_{\text{act}}} \pi_{t,k,l} \cdot f_{k,l}(h_t).$$

To stabilize routing, raw routing logits z are sequentially post-processed via max subtraction, $\tanh$ compression, NaN/Inf numerical guards, and zero-mean centering before being passed to the softmax operator.

3.4 Objective Families and Their Experimental Roles

The Dynamic Hierarchical Knowledge Routing (DHKR) framework supports two optimization formulations: a pure cross-entropy objective and a regularized objective. The primary CE-only formulation is defined as:

$$\mathcal{L}_{\text{CE}} = \mathrm{CE}\left(y, \hat{y}(h_{\text{final}})\right). \tag{1}$$

Alternatively, the regularized formulation incorporates auxiliary distribution penalties:

$$\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{CE}} + \lambda_{\text{KL}}\mathcal{L}_{\text{TR}} + \lambda_{\text{bal}}\mathcal{L}_{\text{bal}} + \lambda_{\text{ent}}\mathcal{L}_{\text{ent}} + \lambda_{\text{KA}}\mathcal{L}_{\text{anchor}} + \beta_{\text{MDL}}\mathcal{L}_{\text{MDL}}. \tag{2}$$

These objective families serve distinct experimental roles. The original CE-only runs establish baseline comparisons under the initial implementation, whereas subsequent identification experiments apply the regularized objective consistently across Controller, Static, Scheduled, and Full conditions to isolate active-set variations within a single objective family. Consequently, absolute training-loss values are not compared across distinct objective families. The active-set controller operates externally to both task objectives. By monitoring validation statistics, routing entropy, usage imbalance, and stability metrics, the out-of-loop controller manages structural availability and coordinates post-transition adaptations—including routing-bias adjustments, history resets, temperature schedules, bounded residual blending, and cooldown intervals—without injecting multi-objective regularization terms into the backpropagated task loss.

3.5 DATA-DEMANDED ACTIVE-CAPACITY CURRICULUM

Capacity curriculum. DHKR distinguishes the configured portfolio envelope from the active portfolio. The envelope defines admissible intervention slots. A slot may be resident but inactive. Once released, the slot participates in routing and receives gradient updates: $\mathcal{C}_t = \{j : m_{t,j} = 1\}, \quad \mathcal{C}_t \subseteq \mathcal{C}_{\max}$. Starting from a restricted active set, the controller may retain the current set or release a path within the configured envelope. This constitutes a curriculum over active capacity rather than example difficulty, providing paths when routing and optimization signals indicate demand.

Routing-pressure-triggered release: Unlike KADA, which fixes its subnetwork count at design time, DHKR monitors multiple optimization- and routing-side signals—including validation statistics, normalized routing entropy, usage imbalance, and routing stability—under a configurable decision rule. A release is accepted within maximum-capacity bounds and cooldown constraints, showing that the configured routing-pressure conditions can trigger release without validation-loss stagnation.

Path release changes more than the number of active computations. It changes the intervention function class available to the optimizer: $\mathcal{F}_t \subseteq \overline{\mathcal{F}}_{t+1}$ when $\mathcal{C}_t \subseteq \mathcal{C}_{t+1}$, where the closure accounts for finite-logit softmax routing. The earlier composition is approached as allocation to newly available paths tends to zero. Parameter optimization changes functions within the current active portfolio, whereas release changes which interventions can participate in subsequent optimization.

Transition stabilization: Path release changes the routing simplex and introduces newly available transformations into the intervention portfolio. DHKR therefore couples each release with usage-history reset, bounded residual blending, temperature control, and an LFB-style routing bias. A cooldown interval then reserves an adaptation stage before another release can occur. The controller and stabilizers thus serve complementary roles: the controller decides when capacity becomes available, while transition control allows the router and subnetworks to adapt to the enlarged active set.

Initialization: We initialize a newly released path from an active path using a Net2WiderNet-inspired Chen et al. (2016) warm start. The current implementation selects the source path by usage or routing-distribution divergence. The copied parameters are perturbed, and routing bias is updated to expose the new path to training. These operations do not preserve the complete model function exactly.

Second-level release: Second-level availability introduces a distinct refinement axis. First-level release expands candidate intervention families, whereas second-level release expands local transformation freedom within a selected parent. Neither axis is required to reach its configured maximum; realized K and L depend on training trajectories. We prioritize expanding K until $K_{\max}$ is reached, growing L selectively when the configured second-level release condition is met. Unlike hierarchical mixture-of-subnetworks paradigms Jordan & Jacobs (1994); Lewis et al. (2021), this demand score combines overload ratio and maximum routing probability within $p^{(2)}$.

4 EXPERIMENTS

4.1 EXPERIMENTAL QUESTIONS

The evaluation addresses four questions: **1. Allocation:** Does preserving input-to-weight correspondence affect validation loss? **2. Path removal:** Do removal effects remain positive after renormalizing the remaining path weights? **3. Availability:** Does a restricted active set differ from soft allocation over a full-from-start portfolio? **4. Release:** Do released paths receive routing mass while optimization continues?

Following KADA Kawamae (2025) for dataset and baseline parity, DHKR initializes from a compact configuration within an admissible capacity envelope defined by $(K_{\max}, L_{\max})$. Mechanism probes use a fixed 100-example Alpaca subset from seed 42. Static, Scheduled, and Full controls test alternative explanations for the observed Controller result. Generation quality is reported in Table 6, MCQA calibration in Table 7, and efficiency in Tables 8-9, complemented by routing visualizations.

4.2 ROLE OF THE FOUR REGULARIZED CONDITIONS.

All four conditions use the same data, regularized objective, maximum configured portfolio, training horizon, and evaluation protocol. Full-from-start tests permanent maximum availability. Static $K = 2$ tests whether the smaller terminal active set alone explains the result. Scheduled $K : 1 \to 2$ tests whether the prescribed discrete sequence alone explains the result. The Controller condition evaluates the complete Controller execution under the same objective. Each comparison therefore targets a different alternative explanation. The four-way comparison is not compute-matched: conditions execute different active portfolios. Active-path compute is reported separately.

4.3 MODEL AND DATASETS

We fine-tuned the open-source `Meta-Llama-3-8B-Instruct` model AI@Meta (2024)[1]. To reduce memory usage and enable efficient experimentation, we applied 4-bit NF4 quantization via Hugging Face's `bitsandbytes`Dettmers et al. (2022). We selected a frozen 8B model to ensure experimental reproducibility independently of overarching model performance variations, and separate intervention behavior from backbone capability changes while enabling efficient single-GPU evaluation. Only KSL modules and DHKR routing components were trained while the base model weights were kept frozen. The training and evaluation pipeline conformed to the `transformers` API, including usage of `GenerationMixin` and `output_hidden_states=True`, ensuring full compatibility with the Hugging Face PEFT framework Mangrulkar et al. (2022).

To align with the compact model scale and isolate behavioral changes from backbone variations, we employed controlled, limited datasets that ensure high reproducibility while preventing additional intervention parameters from providing an unfair experimental advantage. We used three datasets: Alpaca Taori et al. (2023) for instruction fine-tuning, and OpenBookQA Mihaylov et al. (2018) and SciQ Welbl et al. (2017) for factual question answering. Fine-tuning was conducted in a causal language modeling (CLM) setup. We converted all prompts to match LLaMA-3-style input using tokenizer . apply_chat_template . For causal language modeling, non-response tokens were masked with `label=-100`, and sequences were padded or truncated to 1024 tokens.

All experiments were run on a single NVIDIA GPU (`CUDA_VISIBLE_DEVICES="0"`) using `float16`. Fixed random seeds were used for reproducible initialization and data sampling. The regularized four-condition comparison uses seeds 42, 43, and 44; the same-model mechanism probes use seed 42. To ensure a fair comparison, we selected library-default PEFT implementations, QLoRA Dettmers et al. (2023), AdaLoRA Zhang et al. (2023), and BitFit Zaken et al. (2022) from Hugging Face's `peft` library, as reference configurations without method-specific tuning. These results are not presented as a tuned Pareto-optimal comparison.

4.3.1 MECHANISM VALIDATION OF LEARNED INTERVENTION WEIGHTING

To isolate the functional role of input-conditioned intervention weighting from capacity availability, we conducted same-model counterfactual probes at the final $K = 5, L = 2$ state, keeping the trained intervention bank and active capacity fixed while selectively altering the learned weighting. The learned composition achieves an evaluation loss of 1.6107, compared with 1.6668 under uniform weighting and 1.6679 when first-level weights are averaged within each sequence. Replacing each sample's joint intervention weights through a cyclic cross-sample permutation within each evaluation batch increases the loss to 1.6668. Thus, the performance advantage of the learned composition is not explained solely by the presence of the intervention bank, a global weighting profile, or available active capacity; rather, it depends on preserving the correspondence between the input and its learned intervention weights.

4.3.2 RENORMALIZED PATH REMOVAL

We next evaluated the functional contribution of individual intervention paths using leave-one-path-out counterfactuals. Without renormalization, removing an active path increased evaluation loss by 0.0293 to 0.1912, with a mean increase of 0.0827. After removing each active path and renormal-

[1] `https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct`

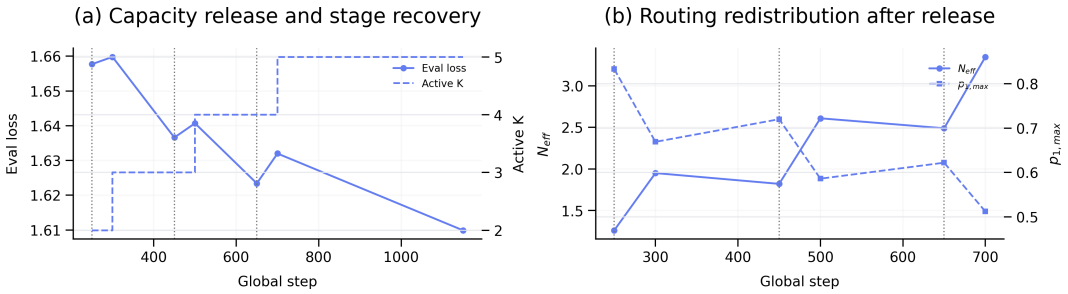

Figure 2: Capacity-transition behavior in the Alpaca/SFT capacity-enabled run. Left: evaluation loss and active first-level capacity. First-level paths are released after steps 250, 450, and 650; each release is followed by a short-lived loss increase and subsequent within-stage recovery. Right: effective routing breadth N_{eff} and maximum first-level allocation $p_{1,\max}$. Newly released paths receive routing mass, increasing N_{eff} and reducing routing concentration.

izing the remaining weights, all ten active paths produced positive marginal loss changes, ranging from 0.0111 to 0.0326, with a mean of 0.0197. Thus, restoring the total mixture weight reduces the mean removal effect by approximately 76%. The large difference between the raw and renormalized effects indicates that unnormalized leave-one-path-out analysis partly captures reduced total intervention magnitude and the consequences of weight redistribution, whereas the renormalized result tests whether the remaining trained paths recover the removed path under the specified redistribution rule. The positive loss change for all ten removals shows that the removed path is not completely recovered by the tested renormalization over the remaining trained paths.

4.3.3 Weight-Removal Association

We examined whether the learned path weights are positively associated with the same-model counterfactual removal effect. For each sample x, we define the utility of path (k, l) as

$$U_{k,l}(x) = L_{\text{renorm},-(k,l)}(x) - L_{\text{full}}(x),$$

where $L_{\text{renorm},-(k,l)}(x)$ is the loss obtained after removing path (k, l) and redistributing its weight over the remaining active paths. Across $n = 100$ Alpaca evaluation examples, the learned joint path weights showed a positive association with this counterfactual utility, yielding a Pearson correlation of $r = 0.472$. The path receiving the highest weight for each sample exhibited a mean utility of 0.0179, whereas the lowest-weight path averaged -0.00125, resulting in a mean top-versus-low difference of 0.0191. Taken together with the cross-sample weight-substitution control, these findings support the interpretation that the learned weighting reflects input-specific functional differences among the active interventions. For the seed-42 trained realization and the fixed 100-example subset, all three allocation substitutions increase validation loss. The reported estimates use the final trained model state from seed 42 and a fixed 100-example evaluation subset; they do not yet establish uncertainty bounds or seed-level robustness.

Consequently, the mechanism probes provide empirical support for input-conditioned intervention allocation within the tested Alpaca/SFT setting. However, they do not establish that the weighting mechanism is causally optimal, that the hierarchical parameterization outperforms a budget-matched flat parameterization, or that dynamic capacity release inherently yields an additional benefit. The latter question is addressed separately below.

4.4 Portfolio-Release Trajectory

Figure 2 and Table 1 visualize the capacity curriculum over the complete Alpaca/SFT run. Active capacity changes discretely, while routing breadth and evaluation loss adapt within each stage. As detailed in Table 1, each capacity transition consistently expands effective routing breadth (N_{eff} increasing from e.g., 1.26 to 1.95 at $K : 2 \to 3$) and reduces routing concentration ($p_{1,\max}$ decreasing). The transition trajectory shows that newly released paths receive routing mass and that optimization continues after each release.

Table 1: Observed first-level capacity transitions in the Alpaca/SFT capacity-enabled run. Immediate post-release values are measured at the first subsequent evaluation, and stage-end values are measured immediately before the next release.

Transition	Pre-loss	Post-loss	Stage-end	Neff pre/post	$p_{1,\max}$ pre/post
$K: 2 \to 3$	1.6577	1.6598	1.6366	1.26/1.95	0.833/0.669
$K: 3 \to 4$	1.6366	1.6407	1.6234	1.82/2.61	0.720/0.586
$K: 4 \to 5$	1.6234	1.6320	1.6099	2.49/3.35	0.622/0.512

Table 2: Matched regularized active-capacity comparison on Alpaca/SFT. Static $(K = 2)$ tests active-set restraint, while the prespecified $(K: 1 \to 2)$ schedule tests whether the discrete capacity sequence alone explains the Controller result. Values are mean $\pm$ sample standard deviation over seeds $(42, 43, 44)$. Lower evaluation loss and higher token accuracy are better.

Condition	Eval loss	Token accuracy
Controller execution, bootstrap, $(K: 1 \to 2)$	$(\mathbf{1.5379 \pm 0.0012})$	$(\mathbf{0.6584 \pm 0.0006})$
Static, $(K = 2)$	(1.5604 ± 0.0065)	(0.6487 ± 0.0018)
Scheduled, $(K: 1 \to 2)$	(1.5645 ± 0.0017)	(0.6477 ± 0.0017)
Full, $(K = 5)$	(1.5820 ± 0.0040)	(0.6456 ± 0.0020)

Each first-level path release produces a short-lived increase in evaluation loss, paired with an immediate increase in effective routing breadth and a reduction in maximum allocation. Within each subsequent stage, evaluation loss recovers below its pre-release value. At the reported first-level release points, the controller logs `stagnate=False`, `imbalance=True`, and `stability=True`, showing that the reported releases occurred with `imbalance=True` and `stagnate=False` under the configured composite rule.

4.5 Matched Availability Comparison

To isolate capacity scheduling from the intervention-composition mechanism, we evaluate active-capacity availability across four conditions under identical training configurations (dataset, regularized objective, maximum intervention portfolio, horizon, and evaluation protocol) over seeds 42, 43, and 44: **Controller Condition:** Begins at $K = 1, L = 1$, performs a predefined bootstrap to $K = 2$ at step 5, and retains $L = 1$. **Scheduled Condition:** Begins at $K = 1, L = 1$ and releases the second first-level path at the prespecified step 5 without using a training-signal-dependent release decision. **Static-$K = 2$ Condition:** Exposes $K = 2, L = 1$ from the beginning of training and keeps this active set fixed during optimization. **Full-from-start Condition:** Exposes $K = 5, L = 1$ throughout training. After the seed-42 pilot, we specified evaluation loss as the primary replication metric for seeds 43 and 44, with token accuracy and active-path FLOPs as secondary metrics. The replication prediction was $L_{\text{Controller},s} < L_{\text{Full},s}$.

As shown in Table 2, the Controller achieves the best performance (loss: 1.5379 ± 0.0012, accuracy: 0.6584 ± 0.0006). The Controller has lower validation loss than the Static, Scheduled, and Full conditions in all three seeds. Static $K = 2$ also has lower validation loss than Full $K = 5$ in all three seeds. These results support several conclusions:

- **Controller vs. Static ($K = 2$):** Terminal active capacity alone is insufficient to explain the Controller's performance, ruling out static capacity limits as the sole driver.
- **Controller vs. Scheduled:** The prescribed discrete sequence alone does not reproduce the Controller result. The remaining difference may include transition-state handling and other Controller-associated execution details.
- **Static vs. Full:** Static $K = 2$ consistently outperforms Full $K = 5$ (mean paired loss difference of 0.0216 ± 0.0028), validating active-capacity restraint. Notably, this is achieved while consuming approximately 37.5% fewer active-path FLOPs (167.8M vs. 268.5M FLOPs/token; see Appendix).

Table 7 in the Appendix reports calibration results from the earlier objective-regularized protocol. These results are retained as evidence for that alternative formulation but are not used to explain the transition behavior of the main strategy-controlled model. The SciQ block is included exclusively as a sanity check for the OpenBookQA-compatible MCQA evaluator; as its accuracy is near-saturated, it is not used as primary evidence of generalization.

5 Discussion and Limitations

In the seed-42 mechanism analysis, learned allocation has lower validation loss than uniform, sequence-averaged, and cross-sample substitutions. Learned routing weights exhibit a positive correlation with counterfactual removal effects. Although a fixed configuration ($K = 2, L = 1$) yields peak generation quality, structural availability serves as a crucial optimization constraint: matched controls show that neither terminal capacity nor the prescribed discrete sequence reproduces the Controller result. The remaining difference is associated with the Controller execution and transition state.

Under the finite-data, fixed-horizon protocol, Static ($K = 2, L = 1$) outperforms Full ($K = 5, L = 1$) across three seeds, demonstrating divergent outcomes between full-from-start and restricted active portfolios. While structural availability acts as an optimization constraint, current experiments do not isolate gradient variance, representation distortion, or active compute as primary causal drivers. The $L = 2$ trajectory confirms second-level routing execution rather than an intrinsic hierarchical quality advantage.

Progressive release is not universally optimal; Controller conditions maintain $K = 2, L = 1$ post-bootstrap. Separate expanding trajectories instead evaluate whether execution paths can be staged dynamically during ongoing optimization. Limitations include heuristic routing thresholds, reliance on single-seed probes with a 100-example subset, and evaluation restricted to Alpaca and OpenBookQA, underscoring the need for broader cross-task stability and non-heuristic allocation strategies.

6 Conclusion

Dynamic Hierarchical Knowledge Routing (DHKR) is a framework for adapting frozen language model backbones via decoupled structural intervention and input-conditioned allocation. DHKR treats structural availability as a variable separate from parameter optimization. Counterfactual probes support input-specific allocation in the tested trained realization. In the matched regularized comparison, Full ($K = 5$) does not reproduce Static ($K = 2$), and neither terminal capacity nor the prescribed sequence reproduces the Controller execution. Separate trajectories show routing redistribution and continued optimization after release. Current evidence does not establish universal advantages for hierarchy, portfolio expansion, or feedback timing under these conditions.

References

AI@Meta. Llama 3 model card. 2024.

G.W. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78 (1):1–3, 1950.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.

Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. In *ACL*, pp. 7085–7095, 2022.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *ICLR*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *NeurIPS*, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39, 2022.

Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *J. Mach. Learn. Res.*, 11:2001–2049, 2010.

Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, Zhaopeng Tu, and Tao Lin. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. In *ICLR*, 2025.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *ICML*, pp. 2790–2799, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net, 2022.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Noriaki Kawamae. Knowledge-aligned domain shift tuning for efficient adaptation in large language models. In *KDD*, pp. 1128–1138, 2025.

Alon Lavie and Abhaya Agarwal. METEOR: an automatic metric for MT evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation, WMT@ACL 2007*, pp. 228–231, 2007.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *ICLR*. OpenReview.net, 2021.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. BASE layers: Simplifying training of large, sparse models. In *ICML*, pp. 6265–6274, 2021.

Chin-Yew Lin. Rouge: a package for automatic evaluation of summaries. In *ACL-workshop*, pp. 25–26, 2004.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*, 2017.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

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 *EMNLP*, 2018.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Evan Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, and et al. Olmoe: Open mixture-of-experts language models. In *ICLR*. OpenReview.net, 2025.

Basil Mustafa, Carlos Riquelme Ruiz, Joan Puigcerver, Rodolphe Jenatton, and Neil Houlsby. Multimodal contrastive learning with LIMoe: the language-image mixture of experts. In *NeurIPS*, 2022.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *AAAI*, pp. 2901–2907, 2015.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pp. 311–318, 2002.

Table 3: Reproducibility-relevant implementation details.

Item	Setting
Backbone	Meta-Llama-3-8B-Instruct, frozen, 4-bit NF4
Seed	42, 43, 44 for the regularized four-way comparison; 42 for same-model mechanism probes
Evaluation	label-only, 1-token, NLL ranking
Decoding	allowed-token filtering for A–D
Data	cached preprocessing, max length 1024
Module save/load	KSL stored separately

Matt Post. A call for clarity in reporting BLEU scores. In *WMT*, pp. 186–191, 2018.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP-IJCNLP*, 2019.

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*, 2017.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A Strong, Replicable Instruction-Following Model, mar 2023. URL `https://crfm.stanford.edu/2023/03/13/alpaca.html`.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *CoRR*, abs/2408.15664, 2024.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Reft: Representation finetuning for language models. In *NeurIPS*, pp. 63908–63962, 2024.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *ACL Volume 2: Short Papers*. ACL, 2022.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *ICLR*. OpenReview.net, 2023.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. In *ICLR*, 2020.

A AI USE STATEMENT

We used these tools to edit the manuscript for grammar and readability, assist with literature searches and identify relevant literature, format references, and edit and format tables. All reported results were produced by the authors' experimental pipeline. We manually verified the cited literature, numerical results, code changes, and final claims.

B APPENDIX

B.1 IMPLEMENTATION DETAILS

Only the KSL is updated via gradient descent, while the base model remains frozen and quantized (4-bit), enabling low-memory training for the 8B-scale model used in our experiments. This method scales model capacity without modifying internal weights or requiring distributed MoE infrastructure, unlike internal MoE layers and without modifying the frozen backbone weights.

KSL is managed as an external, stand-alone module, similar to a LoRA adapter. It does not alter base model weights, which ensures safety (no base model corruption) and portability. However,

Table 4: Conceptual relationship and design contrasts between prior MoE/PEFT paradigms and DHKR

Prior work	Mechanism	DHKR Design Relation
GShard Lepikhin et al. (2021)	Expert balancing	Motivation for conditional routing
BASE Layers Lewis et al. (2021)	Linear assignment	Contrast with bias-based KSL routing
Switch Transformer Fedus et al. (2022)	Token-conditioned routing	Routing mechanism baseline
StableMoE Dai et al. (2022)	Two-stage training	Motivation for transition monitoring
DynMoE Guo et al. (2025)	Auto-tuning capacity	Active-capacity curriculum baseline
OLMoE Muennighoff et al. (2025)	Open MoE LLM	External KSL backbone integration
Net2Net Chen et al. (2016)	Function-preserving growth	Net2Wider subnetwork growth
QLoRA Dettmers et al. (2023)	Quantized PEFT	4-bit frozen backbone foundation

it requires a separate module loading step, which may be less streamlined than integrated LoRA methods.

KSL parameters are trained in fp16/bf16, with activations h dequantized from int4 on the fly. While the MoE architecture theoretically maintains constant FLOPs proportional to active subnetworks, real-world inference latency is influenced by gating and all-to-all communication overhead. With top-1 routing and system-level optimizations (e.g., DeepSpeed-MoE, Tutel), latency remains low and comparable to other methods. Nevertheless, adding an external KSL layer introduces a small latency overhead, typically a few percent—a trade-off to consider during deployment.

- Prompt Reconstruction: We used a prompt recovery method to extract instruction-only segments from input IDs, ensuring accurate input slicing for evaluation.
- Allowed Token Filtering: During decoding, output tokens were restricted to plausible label tokens via a mapping using tokenizer fallbacks.
- Calibration Metrics: We computed both Expected Calibration Error (ECE) Naeini et al. (2015) and Brier Brier (1950) scores from the label probability distribution.
- Similarity Metrics: For generation quality, we computed BLEU, ROUGE-L, METEOR, BERTScore, and SBERT cosine similarity using Hugging Face's `datasets.Metric.load()` and `sentence-transformers`.
- Hierarchical Subnetwork Growth: DHKR supports progressive two-level growth of KSL subnetwork modules. First-level growth (K) is prioritized until a predefined cap is reached, after which second-level subnetworks (L) are selectively grown based on demand-driven signals such as overload and routing confidence.
- Loss-Free Balancing (LFB) Extension: We apply LFB not only to the primary subnetwork routing logits (K) but also to the secondary layer (L), ensuring new subnetworks at both levels are smoothly integrated during training.
- Robust Growth Triggers: Subnetwork growth is gated by composite criteria including stagnation, overload, routing stability, and entropy ratio. A cooldown period (`min_gap_steps`) is enforced to prevent over-expansion and ensure training robustness.
- Subnetwork Initialization Strategies: New subnetworks are initialized via a hybrid mechanism that combines usage statistics and KL-divergence sampling. This allows for both exploitation (copying dominant subnetworks) and exploration (diversifying functionality), improving adaptation speed.
- Training Configuration and Constraints: KADA requires a local base model (no automatic downloading), 4-bit quantization with `bitsandbytes`, and CUDA-enabled environments. Training uses `bf16=True`, with fallback to `fp16=True` if bfloat16 is not supported. subnetwork weights are stored separately via `save_pretrained()`.
- Dataset Caching and Processing: On first run, datasets like OpenBookQA are cached locally in a normalized format. Repeated runs reuse the same directory to ensure consistency.

Algorithm 1 DHKR Training with Active-Set Control

Input: A pre-trained base LLM (frozen), training data x, hyperparameters
Initialize: KADA modules and gating networks
for each epoch t **do**
  $h \leftarrow \text{base_llm}(x)$ ▷ Frozen 4-bit model
  $p^{(1)} \leftarrow \text{gate}_1([h; \text{soft_prompt_mean}])$ ▷ First-level hierarchical gating
  $p^{(2)} \leftarrow \{k : \text{gate}_{2,k}(h) \text{ for } k \in \text{active_K}\}$ ▷ Second-level gating
  $\pi \leftarrow \text{combine}(p^{(1)}, p^{(2)})$ ▷ Compute mixture weights
  $a \leftarrow \text{mixture}(\pi, \text{subnetworks}, h)$ ▷ Subnetwork output
  $\omega \leftarrow \text{ema}(\text{clip}(\gamma \cdot \text{norm}(a)/\text{norm}(h), 0, \omega_{\max}))$
  $h_{\text{final}} \leftarrow h + \omega \cdot a$ ▷ Adaptive residual connection
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{CE}}$
  if objective mode is regularized **then**
    $\mathcal{L} \leftarrow \mathcal{L} + \lambda_{\text{KL}}\mathcal{L}_{\text{TR}} + \lambda_{\text{bal}}\mathcal{L}_{\text{bal}} + \lambda_{\text{ent}}\mathcal{L}_{\text{ent}} + \lambda_{\text{KA}}\mathcal{L}_{\text{anchor}} + \beta_{\text{MDL}}\mathcal{L}_{\text{MDL}}$
  end if ▷ Total loss
  backprop($\mathcal{L}$) ▷ Gradients flow to KADA modules only
  $d_t \leftarrow \textsc{ReleaseDecision}(S_t; \tau, \text{history}, \text{cooldown})$
  if $d_t = 1$ and $\mathcal{C}_t \subset \mathcal{C}_{\max}$ **then**
    initialize and release one configured intervention path
  else if the configured second-level release condition is met **then**
    initialize and release one parent-local refinement path
    grow_L_for_selected_K()
  end if
end for

Algorithm 2 Post-hoc Temperature Scaling (Validation NLL Minimization)

Require: Validation logits $\{\ell_i\}_{i=1}^N$ (pre-softmax), labels $\{y_i\}_{i=1}^N$, initial $T > 0$
Define $\text{NLL}(T) = -\sum_{i=1}^N \log \text{softmax}(\ell_i/T)[y_i]$
Optimize $T^\star = \arg\min_{T>0} \text{NLL}(T)$ ▷ e.g., LBFGS, grid + local search
Inference: for test logits ℓ, output $\hat{p} = \text{softmax}(\ell/T^\star)$
return Calibrated temperature $T^\star$

Table 5: Control families and their inference-time relevance.

Family	Examples	Training role	Inference cost
Schedule	temperature annealing	exploration-to-confidence	none
Task objective	causal LM cross-entropy	optimize active KSL paths	none
Controller observations	validation, entropy, imbalance, stability	estimate capacity demand	none
Transition control	temperature, LFB-style bias, history reset	adaptation after path release	none/fixed
Capacity controller	composite rule, thresholds, cooldown	active-path release	none
Architecture	active K, L	selected capacity	affects KSL FLOPs

B.2 Algorithm

C Evaluation Metrics

We used the Hugging Face Metrics Hub[2] to ensure consistency and reproducibility.

C.1 SciQ (Multiple-Choice).

To evaluate factual question answering and calibration, we implemented a multi-perspective evaluation protocol: (1) Accuracy (1-Token) via constrained decoding using prefix_allowed_tokens_fn; (2) Accuracy (Label-only) by taking the highest log-probability among options (A–D); and (3) Accuracy (NLL Mean) based on the average negative log-likelihood across all tokens in each choice text. Additionally, model calibration is assessed using Brier Score and Expected Calibration Error (ECE) over a 10-bin reliability diagram.

For the multiple-choice evaluation, let $p_i(c)$ denote the predicted probability of choice $c \in \{A, B, C, D\}$ for example i, and let y_i denote the gold answer. Accuracy, and Brier score is computed as

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\left[\arg\max_c p_i(c) = y_i\right] \quad \text{Brier} = \frac{1}{N}\sum_{i=1}^{N} \sum_{c\in\{A,B,C,D\}} \left(p_i(c) - \mathbb{I}[y_i = c]\right)^2.$$

NLL, and average confidence is

$$\text{NLL} = -\frac{1}{N}\sum_{i=1}^{N} \log p_i(y_i), \quad \text{AvgConf} = \frac{1}{N}\sum_{i=1}^{N} \max_c p_i(c).$$

ECE is computed by binning examples by confidence and measuring the weighted difference between bin accuracy and bin confidence. These metrics allow us to distinguish correct but overconfident methods from methods that are accurate and probabilistically reliable.

C.2 Instruction-Tuned Generation

We used a diverse set of automatic metrics, BLEU Papineni et al. (2002), ROUGE-L Lin (2004), METEOR Lavie & Agarwal (2007), SacreBLEU Post (2018) as Lexical Metrics, BERTScore Zhang et al. (2020), and SBERT cosine similarity Reimers & Gurevych (2019) as Semantic Metrics. For instruction-generation experiments, these metrics are computed between generated responses and reference responses in the Alpaca/SFT setting. For multiple-choice QA experiments, the same lexical and semantic metrics may also be computed over predicted and reference answer texts, but we treat MCQA accuracy and calibration as the primary metrics for OpenBookQA.

D Static and Capacity-Enabled Comparisons

Table 6 replicates experiments from KADA Kawamae (2025), with additional comparisons to DHKR. By disabling additional subnetwork expansion in KADA+DHKR and comparing it to the KADA baseline (with Epoch $E = 1$, Batch size $B = 4$, and $K = 2$), we verify DHKR's training robustness.

[2] https://huggingface.co/docs/datasets/v2.21.0/en/how_to_metrics

Table 6: Generation-quality comparison across the original KADA/PEFT protocol and the latest Alpaca/SFT instruction-generation runs. R-L, ME, BS, SB, and SBC denote ROUGE-L, METEOR, BERTScore, SacreBLEU, and SBERT CosSim. Because the original protocol rows and current Alpaca/SFT rows use different reference constructions, scores are compared only within each block. The latest block separates two fixed active-capacity controls from a capacity-enabled DHKR run. All three conditions instantiate the same preallocated maximum KSL structure; they differ in active-path availability and capacity trajectory rather than in the total preallocated KSL parameter budget. For the dynamic condition, the reported bootstrap is an initialization event, whereas the maximum capacity is an upper bound rather than a required endpoint.

Configuration / Model	Epoch	Batch	BLEU	R-L	ME	BS	SB	SBC
Standard Baselines								
Base model	N/A	N/A	0.0524	0.2772	0.3432	0.8734	5.2378	0.8000
QLoRA	1	2	0.0770	0.2055	0.2905	0.8725	7.6992	0.7312
AdaLoRA	1	2	0.0606	0.2813	0.3533	0.8731	6.0584	0.7997
KADA Protocol ($K = 5$)								
Standard	1	2	0.0715	0.2337	0.3678	0.8774	7.1542	0.8015
Standard	1	4	0.0703	0.2260	0.3831	0.8748	7.0344	0.7968
Standard	5	4	0.0978	0.3120	0.4324	0.8909	9.7784	0.8584
KADA Protocol ($K = 10$)								
Standard	1	4	0.0686	0.2269	0.3755	0.8779	6.8606	0.8039
Original KADA-compatible DHKR Comparison								
Control ($K = 2$)	1	4	0.0538	0.2633	0.3875	0.8737	5.3789	0.7672
DHKR ($K : 1 \rightarrow 2$)	1	4	0.0474	0.2641	0.3503	0.8719	4.7358	0.8143
Latest Alpaca/SFT Instruction-Generation Runs								
KSL fixed ($K = 2, L = 1$)	1	4	**0.0845**	**0.2668**	**0.3684**	**0.8760**	**8.4495**	**0.7499**
KSL fixed ($K = 3, L = 1$)	1	4	0.0837	0.2604	0.3585	0.8747	8.3741	0.7451
DHKR capacity-enabled	1	4	0.0802	0.2510	0.3647	0.8713	8.0233	0.7370

Table 7: Evaluation results on OpenBookQA and SciQ. OpenBookQA is used primarily for calibration and reliability evaluation. SciQ is included as a sanity-check MCQA evaluation. Dashes indicate metrics that were not computed for that run. E, R-L, ME, BS, SB, and SBC denote Epoch, ROUGE-L, METEOR, BERTScore, SacreBLEU, and SBERT CosSim.

Model	K/L	E	Acc 1Tok	Acc LblOnly	NLL mean	ECE	Brier	BLEU	R-L	ME	SB	BS	SBC
KADA	1	1	0.7359	0.7359	0.6179	0.1359	0.1046	0.7900	0.7722	0.6361	78.9954	0.9716	0.8336
KADA	2	1	0.7349	0.7349	0.6179	0.1376	0.1046	0.7886	0.7715	0.6353	78.8597	0.9715	0.8331
KADA+DHKR	2	1	0.7339	0.7329	0.5958	0.0225	0.0923	0.7962	0.7738	0.6373	79.6231	0.9713	0.8356
KADA	2	5	0.7480	0.7490	0.6069	0.0640	0.0961	0.7907	0.7847	0.6471	79.0735	0.9729	0.8439
KADA+DHKR	2	5	0.7349	0.7349	0.6028	0.0207	0.0927	0.7919	0.7741	0.6380	79.1883	0.9711	0.8369
KADA	5	5	0.7429	0.7480	0.6079	0.0626	0.0961	0.7931	0.7782	0.6432	79.3115	0.9719	0.8390
KADA+DHKR	5	5	0.7389	0.7389	0.5877	0.0210	0.0923	0.7979	0.7777	0.6412	79.7916	0.9722	0.8390
BitFit+QLoRA	-	1	0.8448	0.8448	0.6300	0.0149	0.0558	0.8715	0.8711	0.7172	87.1545	0.9844	0.9073
AdaLoRA	-	1	0.7228	0.7218	0.6220	0.1518	0.1072	0.7831	0.7600	0.6260	78.3113	0.9691	0.8258
SciQ sanity-check MCQA evaluation													
DHKR	2/1	1	0.9974	0.9974	0.8878	0.0018	0.0011	–	–	–	–	–	–
LoRA	–	1	0.9940	0.9940	0.8223	0.0027	0.0021	0.0000	0.8333	0.3500	0.0000	0.9571	0.8275

Table 8: Runtime analysis of fine-tuning methods on OpenBookQA. All methods utilize a 4-bit quantized and frozen base model. Trainable parameters are calculated as the sum of all parameters with `requires_grad=True`, while total parameters are the sum of all parameters in the model.

Method	Trainable Params	Total Params	Trainable %	Time / Iter (sec)
BitFit	1,703,936	8,031,965,184	0.0212%	18.54
AdaLoRA	5,112,576	8,035,373,888	0.0636%	15.00
KADA+DHKR	637,923,344	5,816,447,004	10.97%	5.67

In Table 6, *KSL control* denotes a fixed-capacity KADA-compatible setting in which the Knowledge Steering Layer is active but dynamic growth is limited. *KSL fixed* denotes a static active-capacity control, whereas *DHKR capacity-enabled* permits the active-capacity state to change under the configured initialization and controller rules. The maximum KSL capacity is an upper bound and need not be reached. Fixed $K = 2$ achieves the strongest generation scores and the lowest evaluation loss in this seed. Although Fixed $K = 3$ and the capacity-enabled configuration share the same preallocated maximum KSL structure, they exhibit distinct active-path configurations and capacity trajectories.

Because the three latest conditions instantiate the same maximum KSL structure, their performance differences cannot be attributed to different total preallocated KSL parameter counts. Instead, they reflect the active-path configuration, routing allocation, and training trajectory. This comparison also reveals that the smallest sufficient capacity is task- and trajectory-dependent and can only be identified through comparison with larger or capacity-enabled configurations.

This non-monotonic trend indicates that downstream performance is not a strict monotonic function of active capacity or the total preallocated parameter budget. Among the tested fixed capacities, $K = 2, L = 1$ is sufficient for the evaluated training horizon. The capacity-enabled run is used as a mechanism experiment rather than as evidence of superior generation quality: it shows that released paths receive routing mass, alter the routing distribution, and support continued optimization after each transition.

E Legacy Calibration Evaluation

Table 7 should be read primarily as a calibration and reliability evaluation. On OpenBookQA, KADA can occasionally achieve higher raw accuracy, but KADA+DHKR consistently reduces ECE to around 0.02, compared with the 0.06–0.13 range observed for fixed-K KADA.

E.1 Runtime Analysis

Table 8 presents the runtime efficiency of BitFit, AdaLoRA, and DHKR on the OpenBookQA benchmark. All methods were evaluated under identical conditions (batch size, sequence length, quantization, and hardware configurations); further implementation details are provided in the appendix.

Table 9: Active-path per-token FLOPs decomposition and one-sample approximate GPU latency. The upper rows report fixed active-hierarchy stabilizer ablations; the lower rows report observed final runs grouped by task.

Setting	FLOPs				Latency
	Gate-1	Gate-2	Experts FFN	Sum KSL	(ms)
Fixed Stabilizer Ablations (Composite)					
Composite (all ON)	16,797,696	83,988,480	3,355,443,200	3,456,229,376	146.31
TR off	16,797,696	83,988,480	3,355,443,200	3,456,229,376	146.87
Entropy off	16,797,696	83,988,480	3,355,443,200	3,456,229,376	167.75
Anchor off	16,797,696	83,988,480	3,355,443,200	3,456,229,376	144.52
Balance off	16,797,696	83,988,480	3,355,443,200	3,456,229,376	146.12
Observed Final Runs: Alpaca/SFT					
Fixed $K = 2, L = 1$	16,797,696	0	1,258,291,200	1,275,088,896	177.44
Fixed $K = 3, L = 1$	16,797,696	0	1,258,291,200	1,275,088,896	204.71
Capacity-enabled ($K = 5, L = 2$)	16,797,696	83,988,480	503,316,480	604,102,656	162.19
Observed Final Runs: SciQ MCQA					
Gate-2 active	16,797,696	83,988,480	1,258,291,200	1,359,077,376	180.1–181.3

Runtime measurements include both forward and backward passes averaged over N iterations after warm-up.

We distinguish trainable-parameter minimality from runtime/system efficiency. DHKR is not designed to minimize the number of trainable tensors; instead, it reduces the effective training time under a frozen quantized backbone by using an integrated KSL path and avoiding PEFT-wrapper fragmentation.

Although KADA+DHKR introduces more trainable parameters, it achieves faster per-iteration training than both BitFit and AdaLoRA. Because runtime is not monotonic in parameter count, and DHKR differs from the PEFT baselines in module placement, execution path, active computation, and software integration, these factors offer plausible explanations for the observed timing—though current end-to-end measurements do not isolate specific overhead contributions. Similar to the original KADA design, KADA+DHKR embeds its core modules directly into the quantized base model.

The methods differ in module placement, active computation, and software integration. The current end-to-end measurements do not attribute timing differences to individual kernels, data movement, or precision conversion. It circumvents repeated precision conversions between `int4` and `FP16`/`BF16`—a primary latency bottleneck in quantized LoRA methods such as QLoRA—thereby enabling larger fused kernels and more efficient GPU execution.

As shown in the upper block of Table 9, the Sum KSL FLOPs remain identical (3,456,229,376 FLOPs) across all stabilizer configurations (Composite, TR off, Entropy off, etc.). This invariance indicates that the stabilizer mechanisms in DHKR do not alter the compute budget of the KSL or the underlying routing architecture. The small differences in measured latency (e.g., 146.31 ms vs. 146.87 ms) stem from kernel-scheduling effects and measurement variability rather than any change in computational load.

The fixed $K = 2, L = 1$ and $K = 3, L = 1$ controls keep the second-level router inactive and therefore incur zero Gate-2 FLOPs. In contrast, the capacity-enabled run reaches $K = 5, L = 2$ and incurs 83,988,480 Gate-2 FLOPs/token. Under the implemented accounting, second-level routing incurs compute when $L > 1$. This observation does not by itself establish that hierarchical factorization improves task quality.

Under the latest active-path accounting, Gate-1 and Gate-2 FLOPs together are approximately 20% of the active Experts-FFN FLOPs in the final $K = 5, L = 2$ state. We therefore report the decomposition directly rather than characterizing the routing cost as negligible. In particular, the logical parameter count of the Llama-3-8B base model remains unchanged—only the number of parameter tensors exposed to PyTorch differs due to the integrated kernel design and the compact representation of quantized weights. Finally, because the KSL module is applied externally after the final hidden state, the computational cost of the base LLM remains unaffected.

Table 10: Ablation and sensitivity results on OpenBookQA. Rows above the separator evaluate stabilizer and temperature variants; rows below evaluate sensitivity to the entropy-growth threshold τ_{ent}. R-L, ME, and BS denote ROUGE-L, METEOR, and BERTScore. Dashes indicate metrics not computed for the sensitivity runs.

Config	Temp	Acc	ECE	Brier	BLEU	R-L	ME	BS
No Aux Losses	0.7	0.7329	0.0260	0.0924	0.7865	0.7721	0.6358	0.9714
Anchor Off	0.7	0.7339	0.0225	0.0923	0.7962	0.7738	0.6373	0.9713
High Temp	1.5	0.7359	0.0292	0.0919	0.7985	0.7743	0.6380	0.9717
Low Temp	0.7	0.7329	0.0248	0.0923	0.7963	0.7722	0.6369	0.9713
$\tau_{\text{ent}} = 0.45$	–	0.7238	0.0288	0.0966	–	–	–	–
$\tau_{\text{ent}} = 0.55$	–	0.7238	0.0288	0.0966	–	–	–	–
$\tau_{\text{ent}} = 0.65$	–	0.7238	0.0288	0.0966	–	–	–	–

Table 11: Ablation of DHKR stabilizers under matched hardware/data. Metrics: Accuracy (One-Token), ECE (10 bins), Brier; router statistics: $p1_{\max}$ coefficient of variation (CV), mean effective experts Neff, fraction of nonzero TR monitoring events. All runs use LLaMA-3-8B-Instruct (4-bit frozen) and identical preprocessing.

Setting	Acc↑	ECE↓	Brier↓	$p1_{\max}$ CV↓	Neff mean↑	TR frac
Composite (all ON)	0.7359	0.0257	0.0923	0.0677	1.6015	0.786
TR off (λ_{KL}=0)	0.7409	0.0299	0.0924	0.1427	1.5779	0.804
Entropy off (λ_{ent}=0)	0.7389	**0.0191**	**0.0922**	**0.0595**	1.5925	0.804
Anchor off (λ_{KA}=0)	0.7359	0.0257	0.0923	0.0594	1.5925	0.804
Balance off (λ_{bal}=0)	0.7379	0.0277	0.0923	0.0604	1.5923	0.804

E.2 Visualization of the Network

Beyond routing concentration metrics, we examined token-level activation patterns to characterize subnetwork behavior. As K grows, $p1_{\max}$ decreases while Neff increases, indicating a shift from single-subnetwork dominance to more distributed first-level routing. In the fixed-$L = 1$ Alpaca controls, Gate-2 remains inactive, with $p2_top1_mean = 1.0$, normalized second-level entropy near zero, and zero Gate-2 FLOPs. In the capacity-enabled Alpaca run, the second-level path is activated at step 10 and remains operational during subsequent first-level releases. The SciQ diagnostic independently confirms nonzero Gate-2 routing statistics and FLOPs after L becomes two.

The SciQ diagnostic validates the operational second-level execution path: after L becomes two, Gate-2 statistics and nonzero Gate-2 FLOPs are observed while training continues. Because the trajectory contains a forced-depth event, this result establishes operational hierarchy and conditional compute, not isolated evidence that the feedback trigger alone selected the expansion.

Figure 3 shows that first-level routing progressively diversifies, while Figure 4 verifies that the second level remains inactive unless a second-level bottleneck is detected. Together, these diagnostics distinguish controlled capacity growth from uncontrolled expert proliferation.

We conducted ablation studies on OpenBookQA to isolate the contributions of auxiliary losses, anchor priors, and temperature scheduling in Table 10 and Table 11.

The lower block of Table 10 details performance variance across multiple entropy-growth thresholds $\tau_{\text{ent}} \in \{0.45, 0.55, 0.65\}$. Across these setups, accuracy and ECE metrics remain constant within the reported precision, demonstrating that DHKR's calibration robustness is insensitive to hyperparameter selection. Furthermore, temperature scaling in Table 10 highlights an exploration–calibration trade-off: higher temperatures ($T = 1.5$) foster exploration, enhancing raw accuracy and BLEU scores, but degrade alignment calibration (ECE rises to 0.0292). Conversely, a lower temperature ($T = 0.7$) refines routing confidence, lowering ECE to 0.0248 at the cost of a minor reduction in raw accuracy.

Table 11 isolates the specific impact of the regularization losses. Disabling Trust-Region (TR) regularization ($\lambda_{\text{KL}} = 0$) yields a minor gain in accuracy (0.7409) but sharply accentuates routing instability, inflating the gate coefficient of variation ($p1_{\max}$ CV) from 0.0677 to 0.1427 and worsening ECE. This confirms that the TR component successfully stabilizes routing distributions without

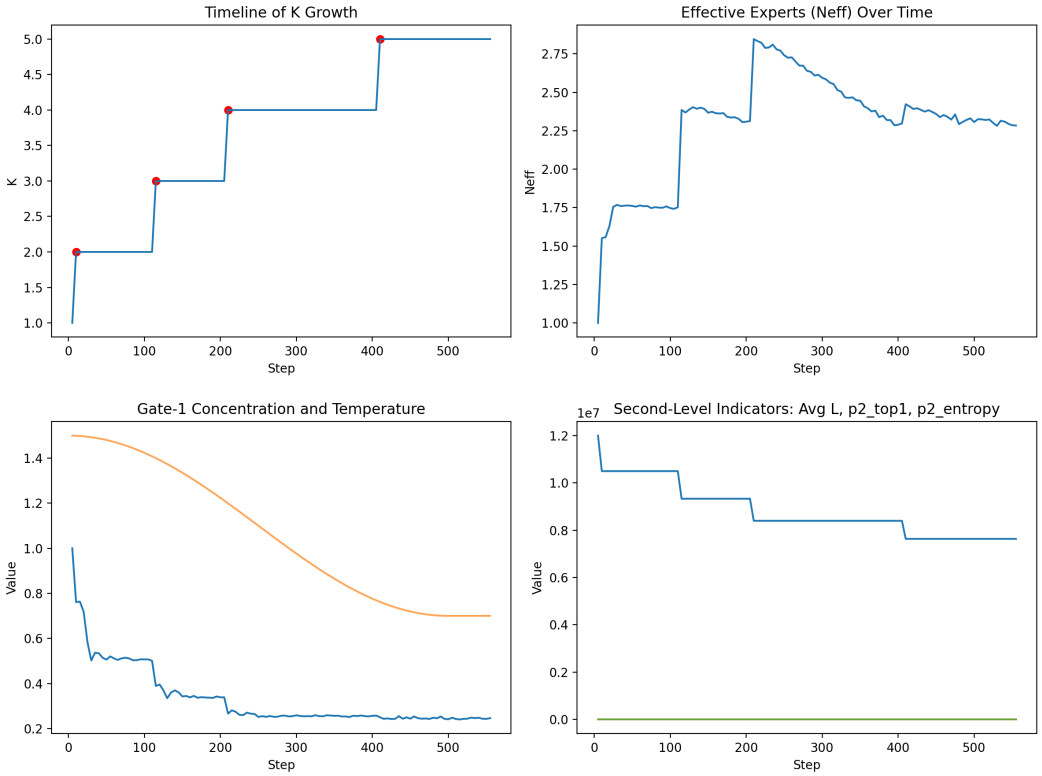

Figure 3: DHKR Subnetwork Growth and Routing Indicators: Panel (top-left): Timeline of K growth, with red markers indicating expansion steps. Panel (top-right): Effective experts (Neff) over training steps. Panel (bottom-left): Gate-1 concentration ($p1_max$) vs. temperature schedule. Panel (bottom-right): Second-level indicators (average L per parent, $p2_top1_mean$, $p2_entropy_norm$).

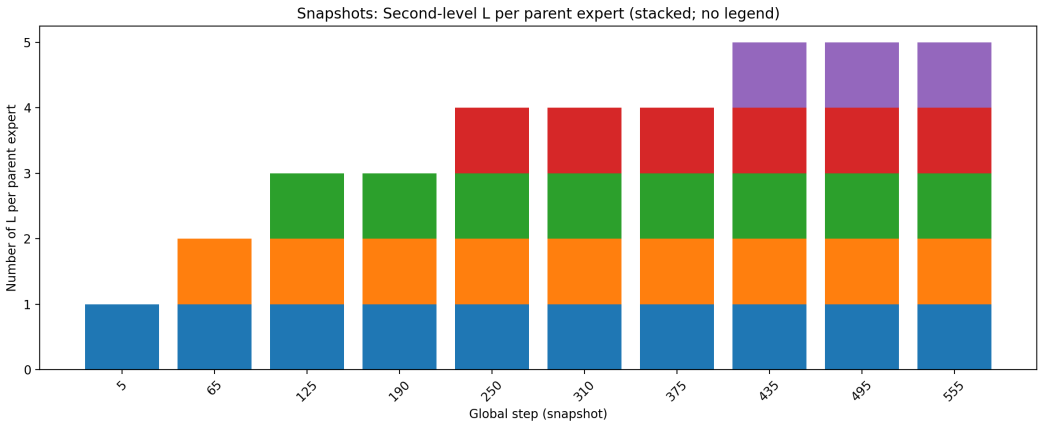

Figure 4: Snapshots of Second-level L per Parent Expert Stacked bars across selected global steps show the number of second-level subnetworks (L) assigned to each parent (first-level expert). This figure visualizes the fixed-$L = 1$ Alpaca control. The separate capacity-enabled Alpaca and SciQ runs activate $L = 2$ and exhibit nonzero Gate-2 routing statistics and FLOPs.

pursuing suboptimal raw-accuracy maximization. Interestingly, turning off entropy regularization ($\lambda_{\text{ent}} = 0$) under low temperatures achieves the best calibration (ECE of 0.0191), implying that the entropy penalty may become redundant when low evaluation temperatures naturally enforce high routing confidence.