Geometry-Aware Post-Training for Mixture-of-Experts Routers

Master Thesis

Hubert Dabrowski, Karlis Tregers, Visnu Ritesh Vijayakumaar Palanisamy

Aalborg University
Department of Energy

Copyright © Aalborg University 2026

Citation and reuse

Please cite this report as:

Hubert Dabrowski, Karlis Tregers, and Visnu Ritesh Vijayakumaar Palanisamy. 2026. *Geometry-Aware Post-Training for Mixture-of-Experts Routers*. Master's thesis, Aalborg University, Department of Energy.

This report may be cited for academic and research purposes. Unless otherwise stated, copyright and other rights remain with the relevant rights holder(s). Reuse, redistribution, or adaptation of the report should follow Aalborg University's applicable rules and any third-party licenses covering figures, datasets, models, code, or other materials referenced in the report.

Department of Energy
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Geometry-Aware Post-Training for Mixture-of-Experts Routers

Theme:
Master Thesis

Project Period:
Spring Semester 2026

Project Group:
AIAS4-1-E26

Participant(s):
Hubert Dabrowski
Karlis Tregers
Visnu Ritesh Vijayakumaar Palanisamy

Supervisor(s):
Petar Durdevic
(pdl@energy.aau.dk)
Daniel Ortiz Arroyo
(doa@energy.aau.dk)

Copies: 1

Page Numbers: 147

Date of Completion:
May 29, 2026

Abstract:

Existing Mixture-of-Expert router objectives mainly control aggregate expert utilization, capacity, and numerical stability, but they do not require nearby hidden states to receive similar routing. This thesis introduces Geometry-Aware Router Post-training (GARP), a router-only regularization method that builds local k-nearest-neighbour graphs from detached pre-router hidden states and penalizes disagreement between router distributions.

The results indicate that hidden-state geometry is a useful signal for organizing sparse MoE routing. With experts frozen and standard Top-K dispatch unchanged, GARP reduces expert-load coefficient of variation by 12.9%, router-neighbour logit distance by 9.2%, and Prompt Sensitivity Score by 1.6%, while leaving mean downstream score nearly unchanged. At the tested scale, the gains are structural rather than accuracy-driven, with some remaining trade-offs.

Contents

Preface **vi**

Nomenclature **vii**

1 Introduction **1**

2 Problem Analysis **3**

2.1 Routing as the control plane 5

2.2 Problem formulation 13

3 State of the Art **15**

3.1 Conditional Capacity at Scale 15

3.2 Balance, Capacity, Stability 17

3.3 Changing the Expert Assignment Rule 18

3.4 Changing the Expert Architecture 19

3.5 Structure-Aware Routing 20

3.6 Runtime Constraints on Routing 22

3.7 Router Refinement After Pre-training 23

3.8 How Routing Interventions Should Be Evaluated 24

3.9 Limitations of Existing Geometry-Aware Routing 27

4 Problem Solution **29**

4.1 Method Scope: Router-Focused Post-Training 33
4.2 Router Disagreement Measure 41
4.3 Geometry-Aware Routing Loss 43
4.4 Confidence-Weighted Geometry Loss 45
4.5 Complete Training Objective 46
4.6 Training Procedure 50
4.7 Evaluation Quantities 53
4.8 Expected Effects 56
4.9 Chapter Summary and Position Relative to Prior Work 57

5 Testing **58**

5.1 Testing Objective and Evidence Structure 58
5.2 Experimental Setup 61
5.3 Main Evidence I: Does Geometry Change Router Behaviour? 65
5.4 Main Evidence II: Is Geometry More Than Load Balancing? 68
5.5 Main Evidence III: What Happens as Geometry Strength Increases? 70
5.6 Supporting Evidence I: Configuration Selection and Component Roles 73
5.7 Supporting Evidence II: Consistency, Sensitivity, and External Validity 75
5.8 Supporting Evidence III: Computational Feasibility 79
5.9 Integrated Testing Summary 80

6 Discussion **83**

6.1 Interpretation of the Main Result 83
6.2 Geometry Beyond Load Balancing 84
6.3 Stability, Sensitivity, and Drift 85
6.4 Practical Operating Region 85
6.5 External Validity and Limits 86
6.6 Implications for Future Work 87

7 Conclusion **88**

A Appendix Guide **90**

B Extended Testing Results **96**

C Sensitivity and External Validity **113**

D Compute Cost **122**

E Pretraining Pilot **132**

Bibliography **141**

Preface

This thesis was prepared at Aalborg University during the Spring semester of 2026. It investigates Geometry-Aware Post-Training for Mixture-of-Experts routers.

We thank our supervisors, Petar Durdevic and Daniel Ortiz Arroyo, for their guidance and support.

Aalborg University, May 29, 2026

Karlis Tregers
<ktrege21@student.aau.dk>

Hubert Dabrowski
<hdabro21@student.aau.dk>

Visnu Ritesh Vijayakumaar Palanisamy
<vrvi21@student.aau.dk>

Nomenclature

Acronym	Definition
ACC	Accuracy.
ARC	AI2 Reasoning Challenge; used for the ARC-Challenge benchmark.
bf16	Brain floating point 16-bit numeric format.
BoolQ	Boolean Questions benchmark.
C4	Colossal Clean Crawled Corpus.
CE	Cross-entropy.
CI	Confidence interval.
CSQA	CommonsenseQA benchmark.
CV	Coefficient of variation.
DDP	Distributed Data Parallel.
EOS	End-of-sequence token or separator.
FFN	Feed-forward network.
GARP	Geometry-Aware Router Post-training.
GB	Gigabyte.
GiB	Gibibyte.
GPU	Graphics Processing Unit.
GPU-hour	One hour of runtime on one GPU.
HE	HumanEval benchmark.
IFEval	Instruction-Following Evaluation benchmark.
***k*-NN**	*k*-nearest neighbours.
KL	Kullback-Leibler divergence.
LB	Load balancing.
LM	Language model.
LLM	Large language model.
LoRA	Low-Rank Adaptation.
LSE	Log-sum-exp.
MoE	Mixture of Experts.
OLMoE	Open Mixture-of-Experts language model.

Acronym	Definition
PEFT	Parameter-efficient fine-tuning.
PPL	Perplexity.
ProSA	Prompt-sensitivity evaluation framework.
PSS	Prompt Sensitivity Score.
SKL	Symmetric Kullback-Leibler divergence.
Top-K	Selection of the K largest router probabilities or expert scores.

Chapter 1

Introduction

Recent progress in large language models has been driven by scale. Increasing parameters, data, and compute has repeatedly improved predictive performance and capability, making scaling a central principle of modern language modeling. However, dense scaling is expensive. In standard transformer architectures, every token activates the same feed-forward sublayers, so larger models increase computation, memory use, bandwidth demand, and communication cost per forward pass. The practical limits of dense scaling are therefore set not only by model quality, but also by training and serving cost [1, 2, 3].

Mixture-of-Experts (MoE) transformers address this limitation through sparse conditional computation. Instead of applying one dense feed-forward network to every token, an MoE layer contains multiple experts and a router that activates only a small subset for each input. This increases total parameter count without increasing active parameters proportionally for every token. MoE can therefore preserve the capacity advantages of larger models while scaling beyond purely dense architectures [4, 5, 6].

This architectural shift relocates the main difficulty. In sparse MoE models, routing is not a secondary implementation detail; it determines which experts receive which tokens, which parameters receive gradient signal, how traffic is distributed across the expert pool, and whether experts develop distinct functional roles. Routing directly affects specialization, optimization stability, and systems efficiency.

Existing sparse MoE training methods address this mainly through load-balancing objectives, capacity control, and router stabilization terms. These mechanisms are necessary to prevent collapse and maintain efficient distributed execution, but they mainly regulate global utilization and numerical stability. They do not explicitly require nearby or semantically similar token representations to be routed similarly

[5, 6, 7, 8, 9, 10, 11].

This omission motivates the central problem studied here. A router can satisfy global balancing while remaining locally inconsistent: tokens that are close in representation space may still receive dissimilar routing distributions or disjoint Top-K expert sets. Such fragmentation can disperse similar tokens across multiple experts rather than mapping them to a coherent subset, potentially weakening specialization and increasing routing brittleness under small hidden-state perturbations. This matters for expert utilization and for behavioral robustness, including sensitivity to prompt variations and semantically equivalent reformulations [9, 7, 10, 12, 11, 13].

This thesis investigates whether sparse MoE routing can be made more locally coherent without sacrificing the balancing properties that make sparse computation practical. It studies a geometry-aware routing regularizer that encourages nearby token representations to induce similar routing behavior. The central hypothesis is that the router should vary smoothly over the hidden-state manifold: if two tokens are geometrically close, their routing decisions should also be close unless the model has strong evidence to separate them. Rather than redesigning the sparse forward pass, this work uses a router-focused post-training setting, where the router is regularized directly while standard sparse MoE computation is preserved [11, 10, 12].

The contribution of this thesis is not a new MoE architecture, but a structured study of local routing consistency in sparse MoE transformers. It makes three contributions:

- **Method:** a geometry-aware regularization framework for post-training router adaptation that preserves standard sparse Top-K inference.
- **Evaluation:** router-structure diagnostics, load-balance measures, prompt-sensitivity analysis, downstream task scores, and computational-cost measurements.
- **Empirical finding:** geometrically aligned router regularization can improve local router coherence and modestly reduce prompt sensitivity while largely preserving language-model quality in the tested router-only setting.

In this way, the thesis positions local routing smoothness as a missing structural ingredient between globally balanced routing and genuinely specialized sparse computation [9, 10, 12, 11].

Chapter 2

Problem Analysis

Once the dense feed-forward part of a transformer is replaced by a sparsely activated expert bank, model capacity no longer depends only on how many parameters exist. It also depends on which token representations are assigned to which expert parameters. The central claim is that the standard anti-collapse machinery used in MoE training is necessary but incomplete: it can make expert usage globally balanced while leaving the routing map locally inconsistent in hidden-state space.

This distinction matters because routing determines which experts receive which tokens and therefore which parameters receive which gradient signals. If nearby token representations are repeatedly routed to unrelated expert sets, then multiple experts can learn redundant computations even when aggregate load statistics appear healthy. The problem studied in this thesis is therefore not simply how to balance experts, but how to preserve the systems benefits of balanced sparse routing while encouraging routing decisions to respect the local geometry of token representations [4, 6, 7, 10, 12, 11].

The scaling pressure described in the introduction appears most directly in the transformer feed-forward network (FFN). In a dense transformer block, every token position is passed through the same FFN after attention. If $h_t \in \mathbb{R}^d$ denotes the hidden state of token t, a simplified dense FFN has the form

$$\mathrm{FFN}(h_t) = W_2\sigma(W_1 h_t + b_1) + b_2, \tag{2.1}$$

where the same matrices W_1 and W_2 are used for every token. Increasing the intermediate width of this layer increases representational capacity, but it also increases the active matrix multiplications for every forward pass. Dense FFN scaling therefore couples total parameter count, active parameter count, memory bandwidth, and inference cost [3]. A simple example of this is shown in Figure 2.1.

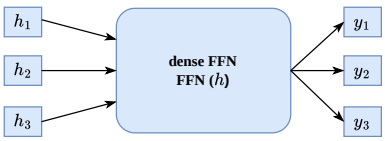


Figure 2.1: Standard dense FFN; all tokens activate the same W_1 and W_2

An MoE layer breaks this coupling by replacing the single FFN with E expert FFNs. Each expert has its own parameters,

$$f_e(h_t) = W_{2,e}\sigma(W_{1,e}h_t + b_{1,e}) + b_{2,e}, \qquad e \in \{1, \ldots, E\}. \tag{2.2}$$

Instead of evaluating all E experts for every token, a router selects a small expert set $\mathcal{S}_t$ for token t. The total number of expert parameters can then grow with E, while the number of active experts per token remains controlled by $K = |\mathcal{S}_t|$, often one or two in practical sparse MoE models [4, 5, 6]. This is also shown in Figure 2.2.

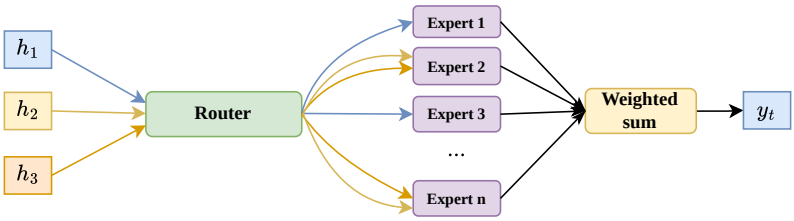


Figure 2.2: Mixture of Experts sparse FFN

Figures 2.1 and 2.2 separate the two roles. The experts provide additional parameter capacity, while the router determines which part of that capacity is used for a given hidden state. The attention and residual structure of the transformer can remain dense over the token stream, but the FFN computation becomes conditional. This is why routing becomes central: the router is the learned assignment map between token representations and the enlarged expert parameter pool.

Figure 2.3 gives a high-level overview of how sparse expert routing fits into a transformer-style architecture. The input tokens are first mapped into token embeddings, after which the representation is processed by the self-attention mechanism. This attention stage remains dense: each token can still exchange information with other tokens in the sequence before the feed-forward computation is applied.

The main architectural change appears in the feed-forward sublayer, where the standard dense FFN is replaced by a Mixture-of-Experts module. As explained previously, instead of sending every token through the same feed-forward network, a router assigns each token representation to a small subset of available experts. This attention MoE feed-forward block is stacked L times: the output hidden states from one layer are passed to the next layer, where attention is applied again before the next MoE routing step.

Operations such as residual connections, layer normalization, and output projection are retained, consistent with standard transformer architectures, as described in Chapter 4.

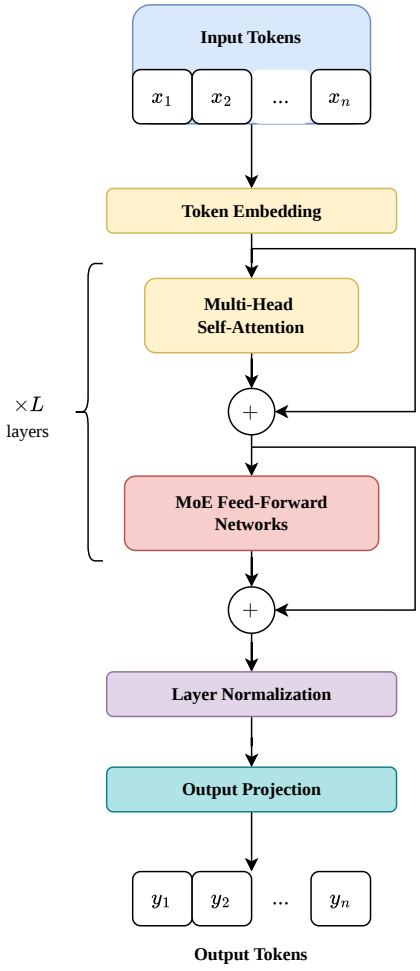


Figure 2.3: High-level MoE transformer flow.

2.1 Routing as the control plane

In a token-choice MoE layer, each token independently produces router scores over experts. Let h_t be the hidden state entering an MoE layer, and let g_ϕ denote the

router. The router computes

$$z_t = g_\phi(h_t) \in \mathbb{R}^E, \qquad p_t = \text{softmax}(z_t), \tag{2.3}$$

where E is the number of experts. A sparse selection operator then chooses a set $\mathcal{S}_t = \text{TopK}(p_t)$, and the layer output is computed from the selected experts:

$$y_t = \sum_{e \in \mathcal{S}_t} \alpha_{t,e} f_e(h_t), \tag{2.4}$$

where f_e is expert e and $\alpha_{t,e}$ is the normalized routing weight for the selected expert.

This small routing module acts as the control plane of the sparse model. It determines which experts are activated, which parameters receive gradient updates, which devices receive token traffic, and which expert outputs are combined into the residual stream. [5, 6, 7].

The discreteness of Top-K routing makes this control problem especially delicate. The probabilities p_t change continuously with the hidden state, but the selected expert set $\mathcal{S}_t$ changes discontinuously when logits cross. When two experts have nearly tied scores, a small perturbation of h_t can replace one expert with another. This is useful when it expresses a real semantic distinction, but harmful when it produces high-frequency routing changes among tokens that are locally similar. Figure 2.4 illustrates the router's role in this control flow.

Specialization vs load balancing

Two goals dominate MoE routing. The first is expert specialization. Ideally, experts should learn complementary computations and become useful on different regions of the model's representation space. Under this view, routing should be input-dependent and structured: tokens that require similar transformations should tend to activate similar experts, while tokens that require different transformations should be separable.

The second goal is load balancing. In an expert-parallel system, each expert has limited capacity. Statistically, overloaded experts receive most of the gradient signal, while underused experts may become poorly trained or effectively inactive. Operationally, overloaded experts become stragglers, increase communication imbalance, and can exceed token capacity limits [5, 6, 8].

These two goals are related but not identical. Specialization asks whether the mapping from token representations to experts is meaningful. Load balancing asks whether the aggregate amount of traffic per expert is acceptable. A router can satisfy one goal while failing the other. For example, if tokens are randomly sprayed

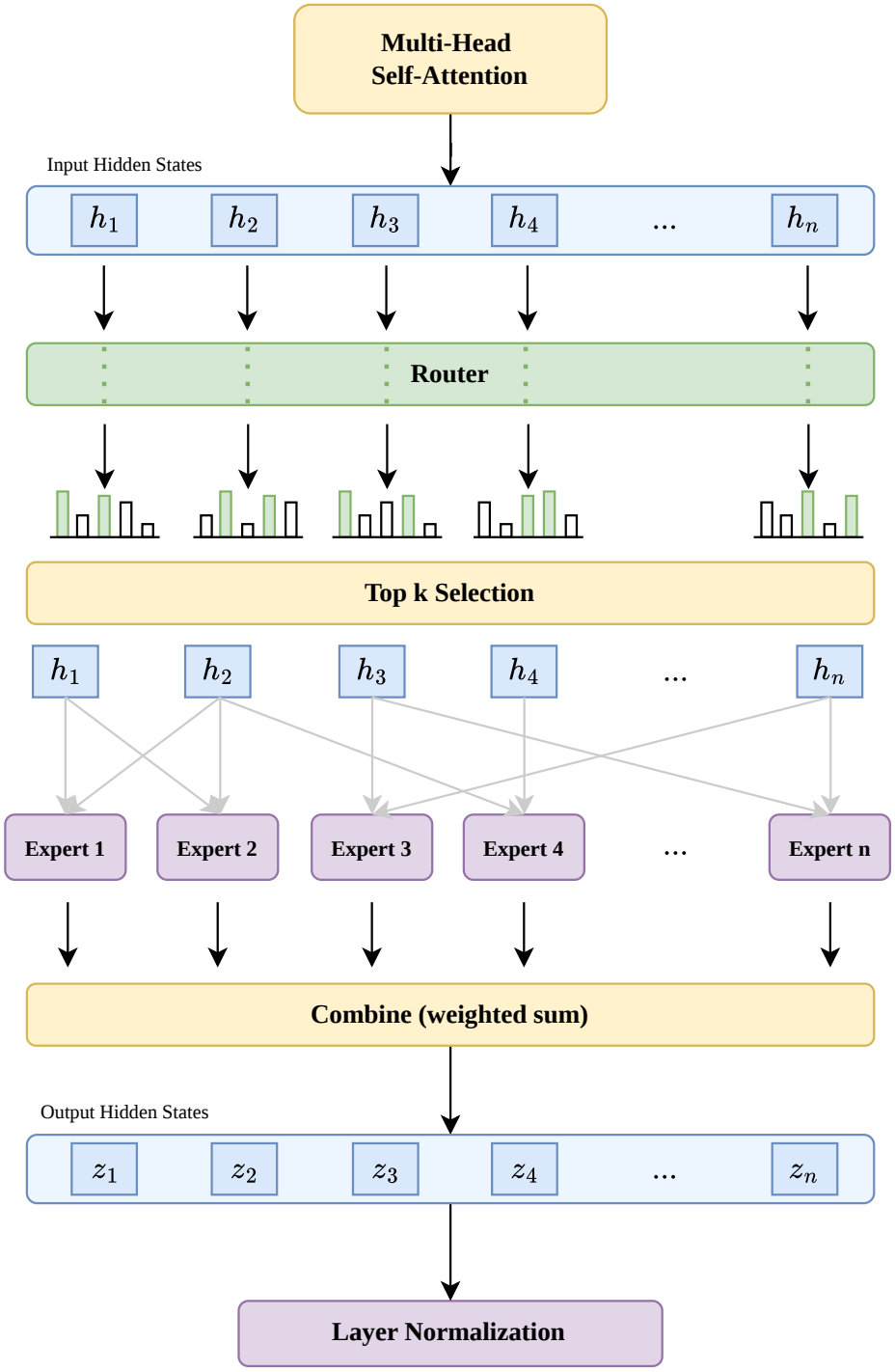


Figure 2.4: Token-choice routing inside an MoE layer. Hidden states are scored by the router, a Top-K selection sends each token to a sparse set of experts, and the selected expert outputs are combined before returning to the transformer stream.

across experts in a nearly uniform way, expert-load statistics may look healthy, but nearby or semantically similar tokens may not be routed to coherent expert subsets. This satisfies the load-balancing objective only superficially, because it uses the expert pool evenly without creating meaningful specialization.

Conversely, a router may consistently assign many tokens to the same small subset of experts. This can look structured, but it should not automatically be interpreted as specialization. If most tokens are attracted to only a few experts, the behavior may simply reflect collapse or degeneracy rather than meaningful expert assignment. Specialization requires that expert selection is organized according to input structure, hidden-state geometry, or functional differences between experts. Good routing therefore requires both meaningful local structure and sufficient global expert utilization.

Inference-time expert activation patterns further illustrate this tension. In real-world inference scenarios, expert activation is often uneven. A small subset of experts may be activated frequently, while others are selected only rarely. This long-tailed activation pattern is not automatically undesirable. Prior analyses of routed language models show that experts can develop specialization patterns related to input structure, context, language, or other semantic properties [14, 15, 16, 17]. Rarely activated experts may therefore encode specialized functionality for low-frequency domains, languages, prompt styles, or reasoning patterns that are underrepresented in the inference distribution. In that case, low activation frequency may reflect meaningful long-tail specialization rather than failure.

However, rare expert usage also introduces an important systems and modeling concern. Experts that are seldom selected still consume memory and may still need to be stored, transferred, cached, quantized, or served during deployment [18, 19, 20, 21]. If such experts are rarely used because they are redundant, poorly trained, or systematically avoided by the router, then the model carries inactive capacity without receiving much benefit from it. This connects rare activation to the broader MoE problems of expert under-utilization, load imbalance, and collapse [4, 5, 6, 7]. Inference-time expert activation patterns therefore provide a useful diagnostic for distinguishing useful long-tail specialization from inefficient expert under-utilization.

This distinction is relevant for the present thesis because a geometry-aware router should not merely make expert usage more uniform for its own sake. Some degree of unevenness may be appropriate if it reflects genuine specialization. The goal is instead to avoid pathological under-utilization caused by unstable or locally inconsistent routing. In other words, the desired routing behavior is not perfectly uniform activation, but a routing map in which expert usage remains globally feasible while nearby hidden states are assigned to experts in a locally coherent

manner [10, 12, 11].

Granularity

The granularity of the balancing signal matters because load balancing is computed over a particular set of tokens, such as a micro-batch, a sequence, or a larger effective batch. This choice changes what the router is encouraged to do. If balancing is computed over very small micro-batches, the loss can pressure each short sequence or local context to use many experts evenly. This may be undesirable because a local context can naturally belong to a narrow semantic region and may only require a small subset of experts. Forcing uniform expert usage at this scale can therefore weaken specialization by spreading related tokens across experts only to satisfy a local balancing statistic [9].

Balancing over a larger effective batch changes the constraint. It allows individual sequences, domains, or neighborhoods to use specialized expert subsets, while still encouraging the full expert pool to be used across the broader training distribution [9]. This distinction is important for the present thesis because local geometric consistency is not the same as local uniform expert usage. A geometry-aware router should make nearby hidden states route coherently, but it should not force every local neighborhood to use all experts equally [10, 12, 11].

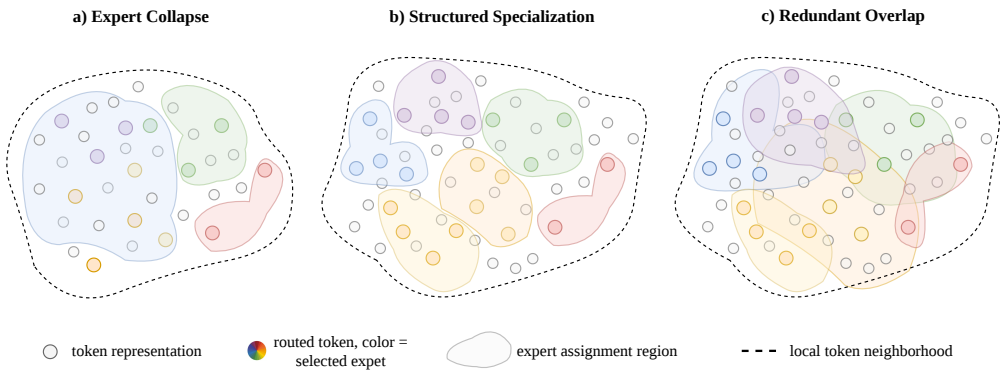


Figure 2.5: The dots represent conceptual tokens and the marked regions are experts that process the tokens. Routing quality cannot be judged from expert utilization alone. A useful router should form coherent expert regions, avoid collapse onto a small subset of experts, and avoid excessive overlap where several experts learn the same local neighborhoods.

Capacity constraints and token dropping

Practical sparse MoE systems impose capacity limits on each expert. If too many tokens are routed to the same expert in a batch, the implementation must either drop excess tokens, reroute them, or use a more expensive execution strategy. In Switch-style systems, capacity factors are used to bound how many tokens each expert processes, and overflow can lead to tokens not receiving their selected expert computation [6].

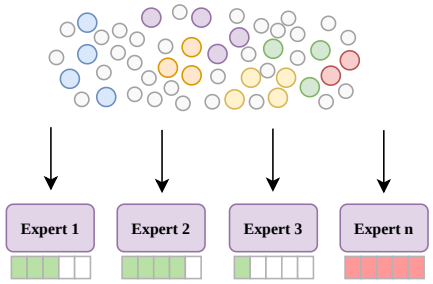


Figure 2.6: Finite expert capacity in a sparse MoE layer. The number of routed tokens per expert is bounded, so imbalanced routing can create overflow even when the total batch size is fixed.

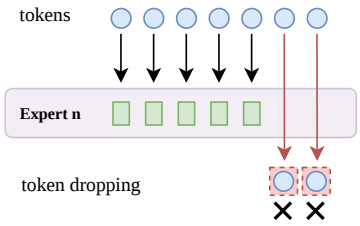
Figure 2.6 shows the basic case: the router assigns tokens to experts, but each expert can only accept a fixed number of tokens for the current batch. If many tokens choose the same expert, the expert's capacity is exhausted even if other experts still have free slots.

The simplest overflow policy is token dropping. As shown in Figure 2.7a, tokens that arrive after the selected expert is full do not receive the selected expert computation. This keeps execution bounded, but it also means that the realized model computation differs from the router's intended assignment.

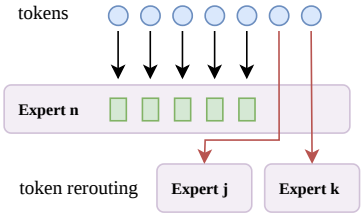
An alternative is overflow rerouting. Figure 2.7b illustrates the case where excess tokens are redirected to other experts with remaining capacity. This preserves active computation for more tokens, but it can weaken specialization because a token may be processed by an expert that was not its highest-scoring choice.

A third option is queueing. Figure 2.7c shows overflow tokens waiting for the selected expert instead of being dropped or reassigned. Queuing preserves the router's selected expert, but it introduces additional latency and makes execution less uniform across experts and devices.

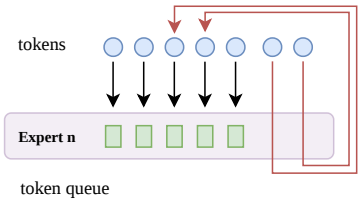
These capacity responses are systems mechanisms, but they expose the modeling problem. Collapse increases overflow risk directly, while spraying increases assignment volatility and makes token-to-expert traffic harder to predict. Routing must therefore be meaningful, balanced, and feasible under finite expert capacity.



(a) Token dropping



(b) Overflow rerouting



(c) Token queueing

Figure 2.7: Overflow handling policies under finite expert capacity. Excess tokens can be discarded, sent to another available expert, or queued for the selected expert.

Why load balancing is not enough

Load balancing solves a necessary problem: it prevents the expert pool from degenerating into a small active subset. However, it does not define what good routing means beyond aggregate utilization.

It is a marginal constraint

Most load-balancing losses operate on marginal expert usage over a batch or effective global batch. Let N be the number of valid routed tokens, and let $\mathcal{S}_t$ be the selected expert set for token t. A hard selected-expert frequency can be written as

$$f_e = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}[e \in \mathcal{S}_t], \tag{2.5}$$

while the mean probability mass assigned to expert e is

$$\bar{p}_e = \frac{1}{N} \sum_{t=1}^{N} p_{t,e}. \tag{2.6}$$

A Switch-style auxiliary term then encourages these aggregate quantities to avoid extreme imbalance, commonly through a product of hard load and soft probability mass [6]. The exact normalization differs across implementations, but the important point is that the loss is computed after token-level routing decisions have been pooled into expert-level summaries.

Here, a token refers to one hidden-state vector at a particular sequence position. The balancing loss counts how many such token representations are assigned to each expert, and how much average router probability mass each expert receives. It therefore answers an aggregate question: whether the expert pool receives a roughly acceptable amount of traffic. It does not answer a structural question: whether the token representations assigned to the same expert are similar to each other, or whether nearby token representations in hidden-state space are routed consistently. In statistical terms, balancing constrains the marginal distribution over experts, but not the local conditional structure of expert assignments inside representation-space neighborhoods.

Local neighborhoods can be inconsistent

The missing information can be expressed directly. Let $\mathcal{N}_k(t)$ denote a local neighborhood of token t in hidden-state space, for example the k-nearest neighbors under cosine similarity. A locally coherent router should make p_t similar to p_u when $u \in \mathcal{N}_k(t)$, unless there is a strong reason to separate them. A generic measure of local inconsistency is

$$\mathcal{R}_{\text{local}} = \frac{1}{N} \sum_{t=1}^{N} \sum_{u \in \mathcal{N}_k(t)} w_{t,u} D(p_t, p_u), \tag{2.7}$$

where $w_{t,u}$ is a similarity weight and D is a distance or divergence between routing distributions.

Standard load-balancing losses do not include such a term. As a result, they can be minimized while $\mathcal{R}_{\text{local}}$ remains large. This explains why local routing jitter can persist after balancing has succeeded: the balancing loss has no direct incentive to make nearby tokens share similar routing distributions.

Expert specialization can be poor

Poor local structure weakens expert specialization. If multiple experts receive overlapping samples from the same local neighborhoods, they are encouraged to learn similar transformations. The model may still use all experts, but the experts do

not contribute as much distinct capacity as their count suggests. This is a form of effective capacity loss: the model pays the memory and communication cost.

This does not imply that every local neighborhood should map to exactly one expert. Language modeling often requires compositional behavior, and Top-K routing can use multiple experts productively. The problem is not diversity itself; the problem is unstructured diversity. A useful router should be able to assign related tokens to overlapping expert subsets in a stable way while still preserving enough global diversity to use the expert pool.

Recent structure-aware routing work is motivated by this same gap. Similarity-preserving routing objectives explicitly argue that balancing alone can fail to preserve token similarity structure [10]. Token-graph and manifold-alignment approaches likewise treat routing as a structured mapping rather than only as a marginal allocation problem [12, 11]. This thesis follows the same broad diagnosis, but focuses on the specific problem of local token-level routing consistency inside a standard sparse MoE setting. This is further explored in Chapter 3.

2.2 Problem formulation

The baseline MoE layer considered in this thesis contains E experts and a token-level router. For each valid token t, the layer receives a hidden state $h_t \in \mathbb{R}^d$, computes router logits and probabilities,

$$z_t = g_\phi(h_t), \qquad p_t = \mathrm{softmax}(z_t), \tag{2.8}$$

selects a sparse expert set,

$$\mathcal{S}_t = \mathrm{TopK}(p_t), \qquad |\mathcal{S}_t| = K, \tag{2.9}$$

and combines the selected expert outputs,

$$y_t = \sum_{e \in \mathcal{S}_t} \alpha_{t,e} f_e(h_t). \tag{2.10}$$

Training typically optimizes a task loss together with auxiliary router terms,

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{LB}} \mathcal{L}_{\text{LB}} + \lambda_z \mathcal{L}_z, \tag{2.11}$$

where $\mathcal{L}_{\text{LB}}$ encourages balanced expert utilization and $\mathcal{L}_z$ stabilizes router logit scale [6, 7].

The problem is that this objective does not explicitly constrain the local geometry of the router. Let $s(h_t, h_u)$ measure hidden-state similarity, and let $\mathcal{N}_k(t)$ be the

local neighborhood of token t. A geometry-respecting router should satisfy the following informal implication:

$$s(h_t, h_u) \text{ is high} \quad \Longrightarrow \quad D(p_t, p_u) \text{ is small,} \tag{2.12}$$

subject to the constraints that expert utilization remains balanced, capacity limits are not violated, and task performance is not degraded.

This leads to the specific research problem:

> How can sparse MoE routing be made locally consistent with hidden-state geometry, so that nearby token representations induce similar routing behavior, while preserving global load balancing and avoiding expert collapse?

The rest of the thesis studies this problem through geometry-aware routing regularization. The solution chapter operationalizes the missing local constraint as a training objective over token neighborhoods, while the evaluation examines whether this improves routing stability and expert-load behavior without unacceptable degradation in language-model quality. The key point established by the analysis is that the missing constraint is not another form of global balancing. It is a local structural constraint on the router map $h_t \mapsto p_t$, and it addresses a failure mode that marginal load statistics cannot detect.

Chapter 3

State of the Art

The previous chapter established the core problem: sparse Mixture-of-Experts (MoE) models can use their experts in a globally balanced way while still routing nearby hidden states inconsistently. The state of the art is therefore reviewed here from the perspective of the router. The central question is not only how MoE models scale, but which parts of the literature actually constrain the mapping from token representations to expert choices.

The chapter follows a narrowing argument. Sparse MoE first appears as a scaling strategy. This makes routing important because the router controls which capacity is used. Classical router training then adds balancing, capacity control, and stabilization so that sparse routing remains trainable and executable. Other work changes the assignment rule or the expert architecture itself. The closest line of work treats routing as structure-aware and asks whether expert selection should preserve similarity, token relations, or representation geometry. Finally, post-training and evaluation work show how such router interventions can be studied without redesigning the full model.

3.1 Conditional Capacity at Scale

A sparse MoE transformer can increase total expert parameters while activating only a small subset of them for each token. This separation between total parameters and active parameters is a central reason why sparse MoE has become an important large-language-model design [22, 23].

The basic idea was established by sparsely gated MoE layers, where a learned gating function activates the top experts for each input [4]. GShard then showed

Table 3.1: Narrative role of each state-of-the-art area.

Literature area	What it establishes	Remaining limitation for this thesis
Sparse MoE scaling	Conditional computation can increase total capacity while keeping active computation bounded.	Scaling relies on the router, but scaling papers do not by themselves define local routing coherence.
Classical router training	Load balancing, capacity control, router stabilization, and gradient improvements make sparse routing practical.	These methods mainly regulate global utilization and trainability, not neighborhood-level structure.
Alternative routing operators	Some routing problems can be addressed by replacing token-choice Top-K with different assignment mechanisms.	Replacing the routing operator changes the computational setting that this thesis deliberately preserves.
Expert architecture design	Expert sharing, grouping, granularity, and heterogeneous capacity can encourage specialization.	These methods change the model architecture rather than regularizing a fixed router map.
Structure-aware routing	Recent work explicitly connects routing to similarity, token interactions, semantics, and manifolds.	The exact setting of local token-level hidden-state smoothness under standard sparse token-choice routing remains only partly explored.
Router post-training and diagnostics	Routers can be refined after pre-training and must be evaluated with internal routing metrics as well as task scores.	Evaluation must show a trade-off: improved local coherence without collapse or unacceptable quality loss.

that this principle could be integrated into large transformer training with expert parallelism, top-2 gating, and explicit capacity constraints [5]. Switch Transformer simplified the routing rule to top-1 dispatch and made clear that sparse MoE scalability depends on auxiliary load balancing and stable router computation [6]. GLaM and ST-MoE further strengthened the quality to efficiency case for sparse LLMs while emphasizing the importance of robust router training [24, 7].

Open MoE models are especially important for this thesis. OpenMoE reported routing phenomena such as early routing learning and drop-towards-the-end behavior, while OLMoE enabled detailed inspection of saturation, co-activation, and specialization in a fully open sparse model [14, 15]. These models shift MoE research from closed large-scale systems toward reproducible routed models whose internal behavior can be measured directly.

Table 3.2: Representative sparse MoE scaling milestones and their relevance to routing.

Work	Main contribution	Router relevance
Sparsely gated MoE [4]	Conditional computation with sparse learned expert activation.	Establishes the router as the mechanism that controls access to a large expert pool.
GShard [5]	Transformer-scale sparse MoE with expert parallelism and top-2 routing.	Connects routing to distributed execution, expert capacity, and load constraints.
Switch Transformer [6]	Simplified top-1 routing for scalable sparse transformers.	Shows that simpler sparse dispatch still requires balancing and capacity mechanisms.
GLaM and ST-MoE [24, 7]	Large sparse language models with quality; efficiency gains and router stabilization.	Emphasize that router stability is a first-order condition for successful sparse scaling.
Mixtral and OL-MoE [25, 15]	Modern open or open-weight routed language models.	Make sparse routing behavior more visible and experimentally inspectable.
Sparse upcycling and related conversion methods [26, 27, 28]	Convert or restructure dense models into sparse expert models.	Show that sparse routing can be introduced after dense pre-training, which supports router-focused adaptation settings.

The main lesson from the scaling literature is therefore: sparse MoE works by moving part of the scaling burden from dense computation to conditional expert selection. That makes the router the control plane of sparse capacity. However, the scaling literature mainly asks whether sparse expert activation is efficient and trainable. It does not fully answer whether the router behaves coherently over local regions of hidden-state space.

3.2 Balance, Capacity, Stability

The dominant sparse transformer formulation is token-choice Top-K routing. Each token independently computes scores over experts and is dispatched to the highest-scoring experts [4, 5, 6, 24, 25]. This rule is simple and compatible with sparse execution, but it creates several coupled problems: experts can become overloaded or inactive, expert capacity can be exceeded, router logits can become numerically unstable, and hard Top-K boundaries can turn small logit changes into discontinuous assignment changes [6, 7, 29].

Classical router training therefore provides a toolkit whose goal is to make sparse routing usable at scale by keeping experts active, preventing collapse, controlling overflow, stabilizing logits, and preserving distributed throughput.

Table 3.3: Classical router-training mechanisms and what they do not directly solve.

Mechanism	Purpose and representative work	Limitation for local coherence
Load or importance balancing	Encourage aggregate traffic to be spread across experts, as in sparsely gated MoE, GShard, and Switch [4, 5, 6].	Constrains marginal expert usage, not whether nearby hidden states choose related experts.
Capacity control	Bound the number of tokens assigned to each expert, as in GShard, Switch, and MegaBlocks [5, 6, 30].	Ensures feasible execution but may drop, pad, or reroute tokens without enforcing geometric structure.
Router stabilization	Improve numerical and optimization robustness, as in Switch and ST-MoE [6, 7].	Stabilizes logits but does not define which tokens should share experts.
Balancing granularity	Decide whether balancing is enforced at the microbatch, sequence, or global-batch level, as discussed in recent load-balancing implementation analyses [9].	Too-local balancing may work against specialization; larger-batch balancing still leaves neighborhood structure unconstrained.
Auxiliary-loss-free balancing	Use dynamic expert biases rather than explicit load-balancing gradients [31].	Regulates utilization without directly aligning routing with hidden-state neighborhoods.
Denser router gradients	Improve router optimization while preserving sparse inference, as in dense backpropagation and DenseMixer [32, 33].	Improves learning signal, but the signal is not necessarily geometry-aware.
Structured capacity-aware assignment	Reformulate routing under capacity constraints, as in Maximum Score Routing [34].	Addresses assignment efficiency more than token-level representation smoothness.

3.3 Changing the Expert Assignment Rule

One way to address routing pathologies is to replace token-choice Top-K routing itself. This literature is important because it shows that imbalance, brittleness, and inefficient assignment are not only loss-design problems. They can also be treated as operator-design problems.

Several families of routing operators are relevant:

- **Expert-side or globally balanced assignment.** Expert Choice lets experts select tokens, while BASE Layers formulate routing as a balanced matching problem [8, 35]. These methods move balance into the assignment procedure rather than relying only on auxiliary penalties.
- **Deterministic or routing-free dispatch.** Hash Layers remove the learned router by assigning tokens through deterministic hashing [36]. More recent routing-free formulations remove the centralized Top-K router more radically [37]. These methods reduce router optimization difficulty, but also reduce or restructure learned token-dependent routing.
- **Differentiable sparse gates.** DSelect-k, ReMoE, and DirMoE replace hard Top-K+Softmax behavior with more differentiable selection or contribution mechanisms [38, 39, 40]. These methods target the non-smoothness of routing boundaries.
- **Soft assignment.** Soft MoE relaxes hard sparse dispatch into a differentiable soft routing formulation [41]. This can reduce Top-K brittleness, but it also changes the computational character of the model.

For this thesis, operator redesign is mostly a contrast class. It demonstrates that routing failures can be attacked by changing the routing rule, but the present work asks a narrower question: can standard sparse token-choice routing be made more locally coherent without replacing the sparse forward pass?

3.4 Changing the Expert Architecture

Specialization can also be improved by changing the expert system rather than the router objective. Weak specialization is not always caused by the router alone. Expert granularity, sharing, grouping, and capacity can also shape whether experts become meaningfully distinct.

The architectural literature therefore provides useful context but not the main mechanism studied here. The present thesis does not introduce new expert granularity, sharing, grouping, or heterogeneous expert sizes. It keeps the sparse MoE architecture fixed and studies the routing map itself.

Table 3.4: Architectural approaches to expert specialization.

Direction	Core idea	Relation to this thesis
Finer granularity and shared experts	DeepSeekMoE uses finer routed experts together with shared experts so common knowledge and specialized transformations can be separated [42].	Shows that specialization can be encouraged structurally, but changes the architecture rather than the post-training router objective.
Compositional expert structure	CartesianMoE uses a Cartesian-product-style routing structure to improve knowledge sharing and reduce sensitivity to routing errors [43].	Addresses routing errors through expert composition rather than local smoothing of a fixed router map.
Grouped or hierarchical experts	Mixture-of-Clustered-Experts uses sequence-level group routing followed by token-level expert activation [44].	Introduces intermediate structure in the expert pool; the present thesis keeps the expert pool fixed.
Heterogeneous expert capacity	HMoE varies expert sizes so capacity itself becomes a specialization variable [45].	Broadens the design space, but does not directly answer whether an existing router is locally coherent.

3.5 Structure-Aware Routing

The closest prior work treats routing as a structured mapping rather than only a utilization mechanism. In this view, routing should reflect similarity, token interactions, semantic organization, or representation geometry [10, 12, 11, 46]. This is the point where the literature comes closest to the central claim of the thesis: expert selection should not vary arbitrarily among nearby token representations.

Table 3.5: Structure-aware routing literature and its distinction from the present thesis.

Line of work	What it adds	Difference from this thesis
Similarity-preserving routing	Adds a load-balancing objective that preserves token-wise relational structure and encourages similar inputs to make more consistent expert choices [10].	Closely related motivation, but it is framed as a similarity-preserving load-balancing loss rather than the router-only hidden-state kNN regularizer studied here.
Expert-representation similarity training	Connects sparse MoE training to representation collapse by measuring and reducing similarity among expert representations [46].	Supports the diagnosis that similarity and collapse matter, but it regularizes expert representations rather than directly smoothing router distributions over local hidden-state neighborhoods.
Token-graph routing	Builds token-level relational structure and lets routing depend on token–token interactions, including attention-derived token similarities [12].	Modifies the routing computation more directly; this thesis preserves independent token-choice inference and adds a training regularizer.
Routing manifold alignment	Uses router-only post-training to align routing weights with neighborhoods defined in an external task-embedding space [11].	RoMA defines neighborhoods over samples or tasks using a separate embedding model. In contrast, this thesis defines neighborhoods over tokens using the MoE LLM's own pre-router hidden states.
Routing probes and semantic analyses	Study whether expert overlap or activation patterns reflect word sense, context, language, or other semantic structure [16, 47, 17].	Diagnostic rather than directly regularizing local routing smoothness.

A good router should map related inputs to related expert behavior when the representation space gives a reliable reason to do so. For token-level MoE routing, this suggests a local smoothness criterion: if hidden states h_t and h_u are close, then their router distributions p_t and p_u should usually be close as well, unless there is evidence that the model should separate them.

At the same time, the literature leaves several design choices open:

- Whether structure should be imposed during pre-training or post-training [6, 11];
- Whether the routing operator should be modified or kept unchanged at inference [8, 41, 11];
- Whether neighborhoods should be defined over tokens, sequences, tasks, or samples [12, 44, 11];
- Whether similarity should be semantic, attentional, geometric, or externally supervised [16, 12, 10, 48];
- How local consistency should be balanced against global expert utilization [6, 7, 10].

The present thesis occupies a specific point in this design space. It studies token-level neighborhoods in the model's own hidden-state geometry, applies the constraint in a router-focused post-training setting, and preserves standard sparse token-choice inference.

3.6 Runtime Constraints on Routing

Routing is also a systems problem. In expert-parallel MoE execution, selected tokens must be dispatched to experts and gathered back. Routing decisions therefore determine communication patterns, padding waste, capacity pressure, memory movement, and inference-time expert reuse [5, 49, 30, 18, 19].

The systems literature reinforces three constraints that any local-routing method must respect:

- **Execution must remain sparse.** Tutel and related systems optimize expert dispatch, adaptive parallelism, and all-to-all communication for sparse expert execution [49]. A method that destroys sparse execution would no longer solve the same deployment problem.
- **Expert capacity still matters.** Conventional MoE execution maps routed tokens into bounded expert buffers. MegaBlocks shows that block-sparse execution can avoid the token-dropping versus padding trade-off more effectively, but the underlying issue remains routing-induced load shape [30].

- **Inference cost depends on activation patterns.** ExpertFlow, MoE-Infinity, and related inference work show that expert activation, caching, offloading, and reuse patterns affect latency and memory pressure [18, 19, 50].

Compression and quantization add another constraint. MoE quantization is affected by expert-specific activation patterns and outliers, so changes in routing can interact with low-bit deployment in non-uniform ways [20, 21, 51]. Tooling such as MixtureKit also reflects the practical need to inspect and compare routed models beyond aggregate task scores [52].

For this thesis, the implication is that local coherence cannot be optimized in isolation. A geometry-aware regularizer should not merely make nearby tokens route similarly; it must also preserve the sparse execution assumptions and global utilization constraints that make MoE useful.

3.7 Router Refinement After Pre-training

The present thesis studies router-focused post-training rather than full sparse MoE pre-training. This setting is supported by recent work showing that router behavior can remain adjustable after the main model has been trained.

Dense backpropagation and DenseMixer target the sparse-gradient problem: the forward pass remains sparse, but the router receives richer update information [32, 33]. Router knowledge distillation and calibration methods show that router-only updates can help recover behavior after expert pruning, merging, or compression [53].

Parameter-efficient adaptation work broadens this conclusion. LoRA and DoRA provide general mechanisms for lightweight model adaptation [54, 55]. Mixture-of-LoRA-Experts, LLaVA-MoLE, GuiLoMo, and LD-MoLE show that routing questions also arise when the experts are adapter modules rather than full feed-forward experts [56, 57, 58, 59]. These methods are not identical to sparse FFN MoE, but they support the broader point that routing remains important in parameter-efficient expert composition.

The post-training literature therefore makes the thesis setting plausible. If routers can be refined after pre-training, then it is meaningful to ask whether a hidden-state geometry signal can refine the router map while leaving the experts and the standard sparse forward pass fixed.

Table 3.6: Post-training and adaptation literature relevant to router-only refinement.

Direction	Main idea	Relevance
Dense router updates	Improve router-gradient quality while preserving sparse inference [32, 33].	Shows that the router can be improved internally without changing the inference-time sparse path.
Router calibration or distillation	Update routers after expert modification or compression [53].	Supports the view that the router remains an effective post-training control point.
Manifold-aligned router post-training	Align routing behavior with external task or representation manifold structure [11].	Closest post-training precedent, but generally coarser than local token-level hidden-state smoothing.
Expertized PEFT	Route among LoRA-style or adapter experts [56, 57, 58, 59].	Shows that routing quality matters beyond full FFN experts, although the mechanism differs from this thesis.

3.8 How Routing Interventions Should Be Evaluated

Routing interventions cannot be evaluated only by downstream task accuracy. A method may improve an internal routing statistic while damaging language-model quality, or it may preserve perplexity while causing expert collapse, excessive concentration, or unstable expert assignments. This is especially important for sparse MoE models because the router affects predictive quality, expert utilization, local routing structure, behavioural stability, and sparse-execution feasibility at the same time. A suitable evaluation protocol must therefore combine external model-quality metrics with internal routing diagnostics.

The purpose of this section is to organize these evaluation quantities into metric families. Each family answers a different question about the effect of a routing intervention.

Language-Model Quality

The first requirement for any routing intervention is that it should not substantially degrade the observable quality of the language model. Standard model-quality metrics include negative log-likelihood, perplexity, and downstream benchmark scores such as accuracy, exact match, F1, or pass@k, depending on the task type [60, 61, 62, 63, 64, 65]. These metrics check whether the modified model still

performs its intended language-modeling or task-solving function.

Perplexity is especially relevant in a post-training router-refinement setting because it directly indicates whether next-token prediction has been harmed. Since the proposed method changes router behavior after pre-training, a large increase in perplexity would suggest that the router has been regularized in a way that disrupts the pretrained model. Downstream benchmarks provide a complementary view, because a routing change may preserve average language-model likelihood while still affecting task behaviour. However, downstream scores alone are insufficient: a model may obtain similar benchmark scores while routing tokens through experts in a collapsed, unstable, or inefficient way. Language-model quality is therefore necessary, but not sufficient, for evaluating routing interventions.

Global Routing Health

The second metric family concerns global routing health. Sparse MoE models rely on the router to distribute tokens across experts in a way that avoids collapse, under-utilization, and execution imbalance. Common diagnostics include expert load, load variance, load coefficient of variation, active expert count, maximum expert share, routing entropy, capacity overflow, and token dropping [5, 6, 7, 14, 15].

Expert-load metrics describe how much traffic each expert receives. [5, 6, 7]. Load CV summarizes how unevenly tokens are distributed across experts, while routing entropy indicates how broadly router probability mass is spread across the expert pool.

It must be noted that perfectly uniform expert usage is not always desirable, because useful specialization may naturally produce uneven activation patterns. A rarely activated expert may still be valuable if it handles a specific domain, language, prompt type, or reasoning pattern.

Local Routing Structure

The limitation of global routing metrics is that they do not measure whether the router behaves coherently over local regions of representation space. Two models may have similar expert-load histograms while having very different local routing structures.

Local routing metrics therefore measure the relationship between representation-space neighborhoods and router behaviour. Typical quantities include Top-K over-

lap, Jaccard similarity between selected expert sets, router-distribution distance, Jensen-Shannon divergence, symmetric KL divergence, and neighborhood consistency [10, 12, 11]. Hard overlap metrics directly reflect sparse expert assignment at inference, while distributional metrics compare the router probabilities before the final Top-K selection.

This is the most direct internal diagnostic for the proposed geometry-aware regularizer. Since the method acts on hidden-state neighbourhoods, the evaluation must test whether nearby hidden states actually induce more similar routing behaviour

Routing Jitter and Assignment Stability

Routing jitter concerns the stability of expert assignments under small changes in the input, hidden state, or prompt formulation. Jitter can be evaluated through low-overlap or disjoint expert assignments among related token states. This is motivated by analyses of sparse Top-K gating and by recent structure-aware routing work that identifies routing fluctuation as a source of non-robustness [29, 10, 12].

Jitter is closely related to local routing structure, but it emphasizes instability rather than similarity alone. The goal is not to make all related tokens select identical experts, but to reduce arbitrary expert switching among locally similar hidden states.

Behavioural Robustness and Prompt Sensitivity

Internal routing metrics are only meaningful if they can be related to model behaviour. For language models, one relevant behavioural property is prompt sensitivity: the degree to which semantically similar prompt formulations produce different outputs or task results. Robustness-oriented evaluation can include prompt-sensitivity, adversarial-robustness, and truthfulness evaluations [13, 66, 67].

PromptSensiScore is a relevant example as it tests whether improved local router coherence corresponds to more stable behaviour across prompt variants [13]. At the same time, prompt sensitivity must be interpreted alongside downstream score, since a model could appear less sensitive by producing more conservative or lower-quality outputs.

Systems Feasibility

Expert assignments determine communication patterns, padding waste, dropped tokens, expert loading, cache reuse, and inference-time memory movement. Systems-oriented metrics therefore include communication imbalance, padding overhead, dropped-token rate, expert activation frequency, expert loading behaviour, and cache or offloading efficiency [49, 30, 18, 19].

Local smoothing could concentrate many tokens onto a small set of experts and increase overflow or communication imbalance even if perplexity remains acceptable. Systems feasibility is also relevant for compression and low-bit deployment, because MoE quantization is affected by expert-specific activation patterns and outliers, so routing changes can interact with quantization in non-uniform ways [20, 21, 51]. Tooling such as MixtureKit further reflects the need to inspect routed models beyond aggregate task scores [52].

Integrated Interpretation

The evaluation of sparse MoE routing interventions requires a multi-metric interpretation. Language-model quality checks whether the modified model still performs well. Global routing-health metrics test whether the expert pool remains usable and avoids collapse. Local routing-structure metrics test whether related hidden states induce related routing behaviour. Jitter metrics measure assignment stability under small changes. Behavioural robustness metrics test whether internal stability corresponds to more stable outputs. Systems metrics check whether the method remains compatible with sparse execution.

3.9 Limitations of Existing Geometry-Aware Routing

The literature establishes the importance of sparse MoE routing, but no single line of work covers the full setting required by this thesis. Scaling work shows that conditional computation can increase total model capacity while keeping active per-token computation bounded [4, 5, 6, 24, 25, 15]. Classical router training then makes this setting practical through balancing, capacity control, stabilization, and improved router gradients [6, 7, 31, 32, 33]. However, these mechanisms mainly regulate utilization and trainability. They do not directly define a local structural constraint on the router map.

Alternative assignment rules and expert-architecture changes address related routing pathologies, but they also change the computational setting. Expert-choice

routing, balanced assignment, routing-free dispatch, soft routing, shared experts, grouped experts, or heterogeneous expert capacity each modify either the routing operator or the expert system [8, 35, 36, 38, 41, 42, 45]. These approaches are useful contrast cases, but they do not answer what can be achieved when the standard sparse token-choice Top-K forward pass and the expert architecture are left unchanged.

The closest recent work is geometry-aware or structure-aware routing, which connects expert choice to similarity, token graphs, semantic organization, or representation manifolds [10, 12, 11, 46, 16]. This literature supports the motivation for local structure, but it still leaves a narrower gap. Existing methods do not jointly address all of the following requirements:

- Preserve standard sparse token-choice Top-K routing at inference;
- Keep the expert architecture fixed rather than changing expert sharing, grouping, or capacity;
- Refine the router after pre-training rather than retraining the full model;
- Define structure at the level of token-level neighbourhoods in the model's own hidden-state space;
- Evaluate local router coherence rather than only aggregate load statistics;
- Preserve global load balance, collapse avoidance, and language-model quality.

This gap motivates the problem solution in Chapter 4. The state of the art suggests that geometry-aware routing is promising, but it does not yet establish whether local hidden-state neighbourhoods can be used as a router-only regularization signal while keeping sparse MoE execution, expert structure, and model-quality constraints intact.

Chapter 4

Problem Solution

Chapter 3 identifies the limitation left by existing routing work. This chapter turns that limitation into a concrete method.

Sparse Mixture-of-Experts (MoE) training contains a structural gap. Load-balancing objectives such as the Switch auxiliary loss encourage healthy aggregate expert usage [6], and recent work shows that the granularity of load balancing matters for expert specialization [9]. However, such objectives do not require nearby hidden states to induce nearby routing decisions. The proposed method addresses this gap through a *geometry-aware routing regularizer*. During post-training, the method constructs a local graph of token representations inside each targeted MoE layer and penalizes router-distribution disagreement across high-similarity graph edges.

The method does not introduce a new expert architecture and does not change inference-time sparse dispatch. The sparse MoE forward pass remains the standard token-choice Top-K router. The contribution is an additional training signal that makes the router map

$$h_t^{(\ell)} \mapsto p_t^{(\ell)}$$

locally smoother over the hidden-state manifold, where $h_t^{(\ell)}$ denotes the hidden state of token t in layer ℓ, and $p_t^{(\ell)}$ denotes the corresponding router probability distribution over experts. The aim is therefore not to replace sparse routing, but to regularize the router so that nearby hidden states tend to produce compatible expert-assignment distributions while retaining the anti-collapse machinery required by practical sparse MoE systems.

The training objective can be summarized as

$$\boxed{\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{geom}} \mathcal{L}_{\text{geom}} + \mathcal{L}_{\text{optional}}} \tag{4.1}$$

where $\mathcal{L}_{\text{task}}$ is the language-modeling task loss, $\mathcal{L}_{\text{geom}}$ is the proposed geometry-aware routing loss, and λ_{geom} controls the strength of the geometry regularization. The term $\mathcal{L}_{\text{optional}}$ denotes additional router-control terms used in specific experimental configurations, such as router stabilization, balancing, importance uniformity, confidence weighting, or sequence-level safeguards.

The geometry loss compares router states across hidden-state neighbours using a configurable router-distance function. Several distance choices are supported, including logit L_2, centered-logit L_2, symmetric KL divergence, and hybrid variants. The main experiments use symmetric KL divergence because it compares the full router probability distributions directly and therefore aligns with the goal of encouraging local agreement in expert-assignment behaviour. The remaining variants are treated as implementation options and ablation choices rather than as the main methodological focus.

Method at a Glance

Before the individual components are defined, Table 4.1 gives the complete method in one view.

Figure 4.1 illustrates this mechanism at a high level, showing how the geometry-aware auxiliary signal is added during training while the ordinary sparse Top-K dispatch remains unchanged at inference.

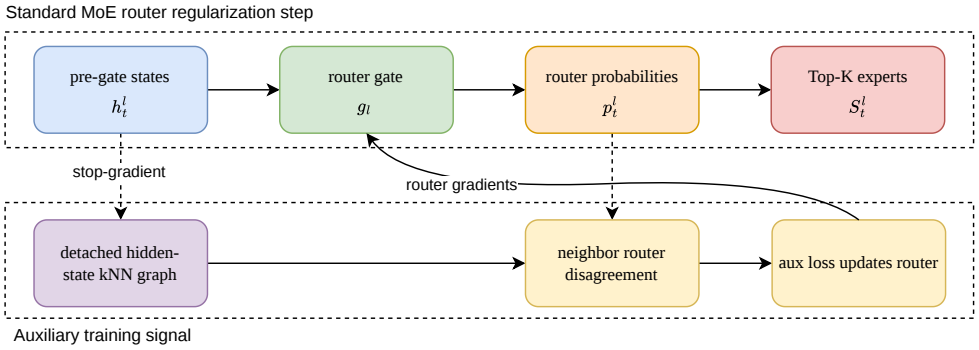


Figure 4.1: High-level mechanism of the proposed method. Pre-gate hidden states are used both by the ordinary router gate and through a stop-gradient branch by the detached hidden-state kNN graph. The auxiliary loss compares router probabilities across graph neighbors and sends gradients back to the router, while inference-time Top-K expert dispatch remains unchanged.

Table 4.1: Method at a glance for Geometry-Aware Router Post-training.

Aspect	Choice in GARP	Purpose
Input signal	Pre-router hidden states $h_{b,t}^{(\ell)}$ from targeted MoE layers	Uses the representation space seen by the router.
Frozen components	Experts, dense backbone, embeddings, and language-model head	Isolates the intervention to routing behaviour.
Trainable components	Router gate parameters W_ℓ, b_ℓ in targeted MoE layers	Keeps post-training lightweight and router-focused.
Neighbourhood graph	Valid tokens are connected by windowed kNN within each batch row using cosine similarity over detached hidden states	Defines local hidden-state geometry without updating the representation space through the graph.
Graph controls	Masking, optional positional windowing, edge weighting, and symmetrization	Keeps edges local, valid, weighted by similarity, and usable for pairwise router comparison.
Router comparison	Dispatch-aligned router distributions compared with symmetric KL divergence	Penalizes disagreement between neighbouring router states.
Confidence control	Entropy-based edge confidence	Down-weights uncertain router states and noisy neighbourhoods.
Training objective	$\mathcal{L}_{\text{task}} + \lambda_{\text{geom}}\mathcal{L}_{\text{geom}} + \mathcal{L}_{\text{optional}}$	Combines task preservation, local geometry, and optional router controls.
Optional controls	Global load balancing, router z-loss, importance uniformity, sequence guard, and coefficient scheduling	Prevents collapse, stabilizes logits, controls concentration, and tunes intervention strength.
Inference behaviour	Standard sparse token-choice Top-K dispatch remains unchanged	GARP changes training signal, not deployment-time routing mechanics.

MoE forward pass

This subsection introduces the minimal notation needed to describe that preserved computation. The notation follows the standard sparse MoE and token-choice routing formulation used in sparsely-gated MoE and Switch-style routing [4, 6]. Consider a transformer layer ℓ with E experts.

For token t, let $h_t^{(\ell)} \in \mathbb{R}^d$ denote the hidden state presented to the MoE router,

where d is the hidden dimension. A standard linear router computes expert logits and router probabilities as

$$z_t^{(\ell)} = g_\ell(h_t^{(\ell)}) = W_\ell h_t^{(\ell)} + b_\ell, \qquad p_t^{(\ell)} = \mathrm{softmax}(z_t^{(\ell)}) \in \Delta^{E-1}, \tag{4.2}$$

where $z_t^{(\ell)} \in \mathbb{R}^E$ contains one score for each expert, and $p_t^{(\ell)}$ is the corresponding probability distribution over the E experts. The notation Δ^{E-1} denotes the $(E-1)$-dimensional probability simplex: the set of non-negative E-dimensional vectors whose entries sum to one. Thus, $p_{t,e}^{(\ell)}$ can be interpreted as the router's probability mass assigned to expert e before sparse expert selection.

Sparse computation is obtained by selecting only the highest-scoring experts. The dispatch operator chooses a small expert set

$$\mathcal{S}_t^{(\ell)} = \mathrm{TopK}(p_t^{(\ell)}, K), \qquad |\mathcal{S}_t^{(\ell)}| = K \ll E, \tag{4.3}$$

where $\mathrm{TopK}(\cdot, K)$ returns the indices of the K largest router probabilities. The set $\mathcal{S}_t^{(\ell)}$ therefore contains the experts selected for token t in layer ℓ. The condition $K \ll E$ expresses the sparse-computation setting: only a small number of experts are executed for each token, even though the layer contains many available experts.

The MoE output is then formed by combining the outputs of the selected experts:

$$y_t^{(\ell)} = \sum_{e \in \mathcal{S}_t^{(\ell)}} \alpha_{t,e}^{(\ell)} f_e^{(\ell)}(h_t^{(\ell)}), \tag{4.4}$$

where $f_e^{(\ell)}$ is the feed-forward network corresponding to expert e in layer ℓ, and $\alpha_{t,e}^{(\ell)}$ is the normalized routing weight assigned to that selected expert. In practice, $\alpha_{t,e}^{(\ell)}$ is obtained from the router probabilities restricted to the selected Top-K experts. Thus, the router first produces a full distribution over the expert pool, but the forward pass only executes a sparse subset of experts.

The main source of discontinuity is the TopK selection. The router probabilities change smoothly as the hidden state changes, but the selected expert set $\mathcal{S}_t^{(\ell)}$ can change abruptly when two experts exchange rank. For example, if two expert logits are nearly tied,

$$z_{t,i}^{(\ell)} \approx z_{t,j}^{(\ell)},$$

then a small perturbation δ of the hidden state changes the logits to

$$z_t^{(\ell)\prime} = W_\ell(h_t^{(\ell)} + \delta) + b_\ell = z_t^{(\ell)} + W_\ell \delta.$$

Even when $\|\delta\|$ is small, this change can be sufficient to alter the Top-K ranking and replace one selected expert with another. This is the local mechanism behind

routing jitter: nearby hidden states may induce different sparse expert assignments even when their router distributions differ only slightly.

The proposed method does not remove this sparse dispatch mechanism. Instead, it regularizes the router probability map before the TopK decision is applied. The goal is to make nearby hidden states less likely to fall on unrelated sides of router decision boundaries, while preserving the standard sparse MoE forward pass at inference time.

4.1 Method Scope: Router-Focused Post-Training

The method is designed for a pretrained sparse MoE language model. In the implementation, the main model is `allenai/OLMoE-1B-7B-0125-Instruct`, selected because it is an open MoE language model with accessible router modules [15]. The method is not full fine-tuning. It uses a small trainable boundary:

- Router gate parameters g_ℓ in the targeted MoE router layers are always trainable;
- Dense backbone weights, frozen experts, embeddings, and the language-model head remain fixed.

If $\mathcal{I}_{\text{target}}$ is the set of targeted MoE layers, the router-only trainable parameter set is

$$\theta_{\text{router}} = \{W_\ell, b_\ell : \ell \in \mathcal{I}_{\text{target}}\}. \tag{4.5}$$

The thesis experiments can therefore isolate whether the router itself can be made more geometrically coherent while the expert functions, dense backbone, embeddings, and language-model head remain fixed.

Assumptions Behind the Proposed Solution

The proposed method relies on six explicit assumptions:

1. Detached pre-router hidden states contain meaningful local structure for routing, so their neighbourhoods can provide a useful signal for the router map.
2. Cosine-similarity kNN within the same batch row and optional positional window is a usable approximation of local hidden-state geometry.

3. Nearby hidden states should have compatible router distributions, but they should not necessarily be forced to use identical experts.
4. Router-only post-training helps isolate routing effects, but it also limits the ability of experts, attention layers, embeddings, and the language-model head to adapt to changed routing patterns.
5. The task loss and router-control terms are required because geometry should not be optimized in isolation from model quality, expert utilization, or router stability.
6. Standard token-choice Top-K inference is assumed to remain unchanged, so the method must act as a training-time regularizer rather than a replacement routing operator.

Design Principle: Local Router Smoothness

The local smoothness assumption does not mean that every nearby token must use exactly the same expert. That would destroy the compositionality that Top-K routing is meant to provide. The assumption is weaker:

> If two token representations are close in the hidden-state geometry of a routed layer, then their router probability distributions should usually be close, unless the language-model task or the global balancing constraint gives a reason to separate them.

For a routed layer ℓ, define the router map

$$r_\ell(h) = \mathrm{softmax}(g_\ell(h)).$$

The desired local property is a neighborhood-level smoothness condition:

$$\mathrm{sim}\left(h_t^{(\ell)}, h_u^{(\ell)}\right) \uparrow \quad \Longrightarrow \quad D_{\mathrm{router}}\left(r_\ell\left(h_t^{(\ell)}\right), r_\ell\left(h_u^{(\ell)}\right)\right) \downarrow . \tag{4.6}$$

This condition directly addresses the failure mode identified in Chapter 2: balanced routing histograms can coexist with locally incoherent routing maps. The geometry loss adds exactly the missing conditional constraint.

Hidden-State Neighborhood Graph

The geometry loss begins by constructing a token graph inside each targeted layer. For batch element b, layer ℓ, and valid token position t, let $h_{b,t}^{(\ell)}$ be the pre-gate hidden state captured immediately before the router. The implementation

builds neighborhoods independently within each batch row, which corresponds to a packed sequence or block. It never connects tokens across different batch rows. Because the training data can be packed with EOS separators, a single block may contain text from more than one raw example. Therefore, the graph is best described as sequence-local or block-local rather than strictly document-local.

Figure 4.2 shows this graph-construction flow before the individual operations are defined formally.

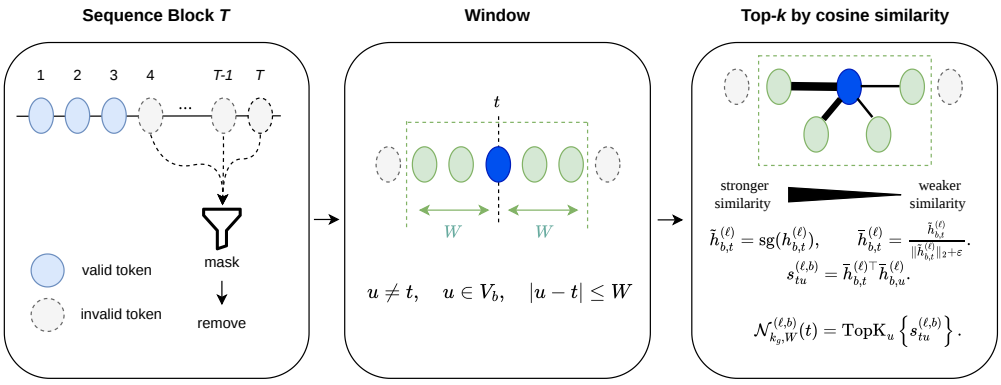


Figure 4.2: Overview of the windowed kNN graph construction. Valid token positions are selected inside a sequence block, candidate neighbors are restricted to the same batch row and optional window, and the final neighborhood is chosen by cosine similarity in detached hidden-state space.

Algorithm 1 summarizes the graph-construction step. The purpose of the algorithm is to show the operational flow: hidden states are detached, valid candidates are selected within each batch row, cosine similarity is computed, and the resulting kNN graph is weighted and symmetrized before it is used by the geometry loss. The equations following the algorithm define these operations more precisely.

Algorithm 1 Windowed hidden-state kNN graph construction

Require: Pre-gate hidden states $h_{b,t}^{(\ell)}$, valid-token sets V_b, neighbour count k, window W, temperature τ

Ensure: Symmetrized graph edges $\mathcal{E}_{\text{sym}}^{(\ell,b)}$ and weights $w_{\text{sym},tu}^{(\ell,b)}$

for each targeted layer ℓ **do**
  for each batch row b **do**
    Detach pre-gate hidden states: $\tilde{h}_{b,t}^{(\ell)} \leftarrow \text{sg}(h_{b,t}^{(\ell)})$
    Normalize detached states to obtain $\bar{h}_{b,t}^{(\ell)}$
    for each valid token $t \in V_b$ **do**
      Form candidate set:

$$C_t \leftarrow \{u \in V_b : u \neq t,\ |u - t| \leq W\}$$

      Compute cosine similarities between t and candidates $u \in C_t$
      Select up to k nearest neighbours:

$$\mathcal{N}_{k,W}^{(\ell,b)}(t) \leftarrow \text{TopK}_u(s_{tu}^{(\ell,b)}, k)$$

      Convert selected similarities into normalized edge weights $w_{tu}^{(\ell,b)}$
    end for
    Symmetrize directed kNN weights to obtain $w_{\text{sym},tu}^{(\ell,b)}$
  end for
end for

Figure 4.3 gives an end-to-end view of the geometry pipeline. The following subsections walk through each of the five steps in order: detaching the pre-gate hidden states, normalising them to unit vectors, selecting cosine-similarity neighbours, converting selected scores to temperature-weighted edge strengths, and aggregating weighted router-disagreement penalties into the final geometry loss.

Detached pre-router states

The five operations that make up the geometry computation; detaching the pre-gate hidden states, normalising them to unit vectors, selecting cosine-similarity neighbours, converting selected scores to temperature-weighted edge strengths, and aggregating router-disagreement penalties into the final loss; depend on each other in a fixed order. Before each is defined formally, Figure 4.3 shows how they connect end to end.

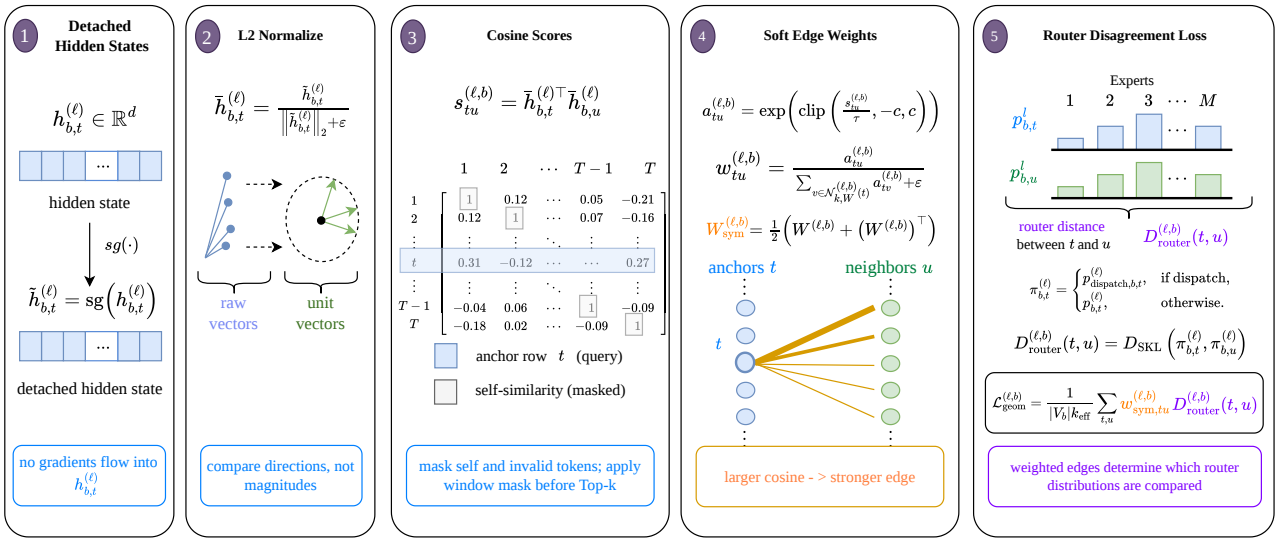


Figure 4.3: End-to-end view of the geometry pipeline. Detached pre-router hidden states are normalized before cosine scores are computed. The selected kNN edges are converted into temperature-weighted edge strengths, and each edge contributes a weighted router-disagreement penalty to $\mathcal{L}_{\text{geom}}$.

As shown in Figure 4.4, the graph first uses detached hidden states:

$$\tilde{h}^{(\ell)}_{b,t} = \mathrm{sg}\left(h^{(\ell)}_{b,t}\right). \tag{4.7}$$

The stop-gradient operator is essential because it makes the hidden-state graph act as a fixed training signal for the router. In the router-only setting used in this thesis, it ensures that the auxiliary geometry objective updates the gate parameters without propagating gradients into frozen or non-targeted representation parameters. In broader trainable-scope variants, the same principle also prevents the geometry loss from reducing itself by moving hidden representations rather than by improving the router map. Without this separation, the objective could encourage a shortcut solution in which hidden states are moved to make the neighbor graph easier to satisfy. Detaching the hidden states therefore keeps the geometry signal focused on the router rather than on the representation space itself.

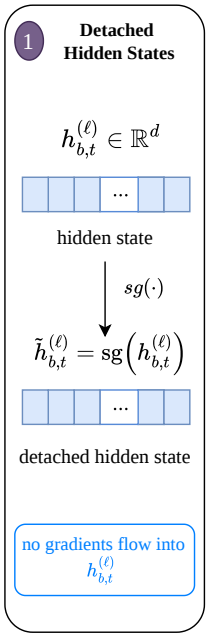


Figure 4.4: Step 1

In the current implementation, the router logits used by the auxiliary losses are

recomputed from detached pre-gate states:

$$z^{(\ell)}_{b,t,\text{aux}} = g_\ell\left(\text{sg}(h^{(\ell)}_{b,t})\right), \qquad p^{(\ell)}_{b,t,\text{aux}} = \text{softmax}(z^{(\ell)}_{b,t,\text{aux}}). \tag{4.8}$$

Thus $\mathcal{L}_{\text{geom}}$, $\mathcal{L}_{\text{bal}}$, and $\mathcal{L}_z$ update the gate parameters directly.

Similarity metric

Figure 4.5 corresponds to the second step, where the detached states are L_2-normalized:

$$\bar{h}^{(\ell)}_{b,t} = \frac{\tilde{h}^{(\ell)}_{b,t}}{\|\tilde{h}^{(\ell)}_{b,t}\|_2 + \varepsilon}. \tag{4.9}$$

Similarity is then the dot product

$$s^{(\ell,b)}_{tu} = \bar{h}^{(\ell)\top}_{b,t} \bar{h}^{(\ell)}_{b,u}. \tag{4.10}$$

Cosine similarity is used because hidden-state magnitude can vary for reasons unrelated to semantic neighborhood structure or layer normalization scale. The router, however, is sensitive to directions in hidden space because a linear gate computes expert logits through projections. Cosine similarity therefore provides a stable approximation to local direction-based geometry.

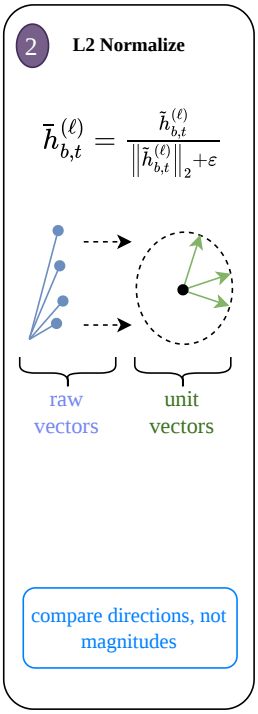


Figure 4.5: Step 2

Windowed k-nearest neighbors

For each token t, the method selects k nearest hidden-state neighbors:

$$\mathcal{N}^{(\ell,b)}_{k,W}(t) = \text{TopK}_u\left\{s^{(\ell,b)}_{tu} : u \neq t,\ u \in V_b,\ |u-t| \leq W\right\}, \tag{4.11}$$

where V_b is the valid-token set for batch row b, k is the geometry-neighbor count, and W is the positional candidate window. The effective neighbor count is

$$k_{\text{eff}} = \min(k, |V_b| - 1).$$

Figure 4.6 highlights the neighbor-selection step. The window W is not a claim that semantic neighborhoods are purely positional. Full-sequence kNN can connect tokens across distant formatting patterns, repeated punctuation, or unrelated conversation turns. A finite window restricts the eligible neighbor set to nearby positions while still using hidden-state cosine similarity, not adjacency alone, to choose the final neighbors. Larger windows increase the number of eligible long-range candidates and may increase noisy edges; smaller windows reduce accidental long-range matches, but can make the regularizer behave closer to local positional smoothing.

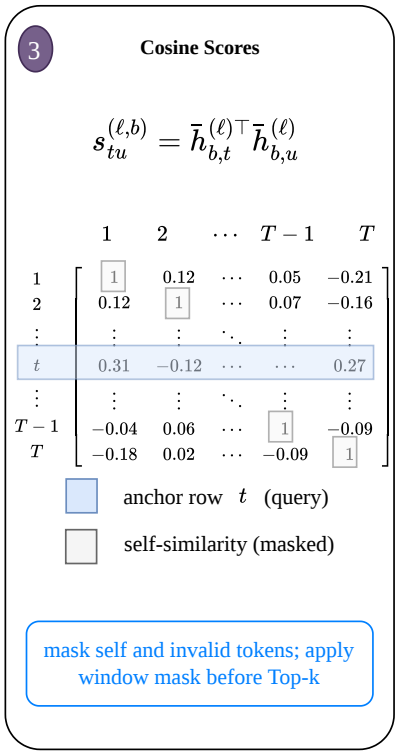


Figure 4.6: Step 3

Implementation note. In the current implementation, the kNN search is performed with chunked exact similarity blocks: the window masks out ineligible token pairs after each chunked similarity computation. Therefore, W mainly changes the graph that is optimized, not the dominant peak memory term. Increasing W does not create a full $S \times (2W+1)$ tensor. Peak scratch memory is controlled by the fixed row and column chunk sizes and by the retained $S \times k$ neighbor tensors. The larger activation overhead comes from storing pre-gate hidden states and router probabilities for the targeted router layers. With chunk sizes C_r and C_c, the peak kNN scratch memory is approximately $O(C_r C_c + Sk + SkE)$, while the main captured-activation cost scales with the number of targeted layers as approximately $O(|\mathcal{I}_{\text{target}}|BTD)$.

Edge weighting and symmetrization

Selected similarities are normalized into non-negative edge weights:

$$a_{tu}^{(\ell,b)} = \exp\left(\text{clip}\left(\frac{s_{tu}^{(\ell,b)}}{\tau}, -c, c\right)\right), \tag{4.12}$$

then normalized over the selected neighbors:

$$w_{tu}^{(\ell,b)} = \frac{a_{tu}^{(\ell,b)}}{\sum_{v \in \mathcal{N}_{k,W}^{(\ell,b)}(t)} a_{tv}^{(\ell,b)} + \varepsilon}. \tag{4.13}$$

Figure 4.7 illustrates the weighting step. The temperature τ controls how concentrated the graph signal is. A low temperature makes the strongest neighbor dominate the loss, while a high temperature distributes weight more evenly across the selected neighborhood. Low temperature can overfit to noisy nearest-neighbor edges; too high a temperature can blur distinct local regions and weaken specialization.

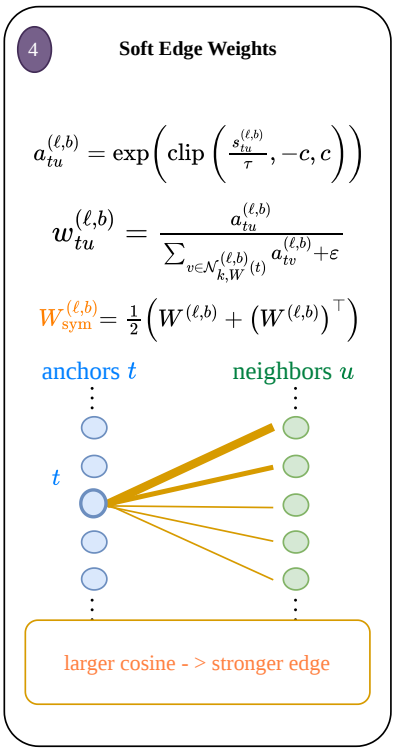


Figure 4.7: Step 4

To ensure that the geometry constraint acts on an undirected neighborhood graph, the weighted adjacency matrix is symmetrized before computing the final loss. Let $W^{(\ell,b)}$ denote the directed adjacency matrix whose entries are $w_{tu}^{(\ell,b)}$. The symmetrized adjacency is defined as

$$W_{\mathrm{sym}}^{(\ell,b)} = \frac{1}{2}\left(W^{(\ell,b)} + \left(W^{(\ell,b)}\right)^{\top}\right). \tag{4.14}$$

This makes the geometry penalty bidirectional. If token u is selected as a neighbor of token t, then the router disagreement between t and u is penalized symmetrically, even if t is not selected as a neighbor of u in the original directed kNN graph. This prevents the regularizer from depending on the arbitrary direction of the neighbor query and better matches the intended interpretation of local geometric consistency.

Router State Representation

The router produces a probability vector over E experts, but the sparse MoE forward pass only dispatches tokens to a selected subset of experts. The geometry loss therefore compares a router-state representation $\pi_{b,t}^{(\ell)}$ rather than the expert outputs themselves. This keeps the regularizer focused on the assignment behaviour of the router, not on the content computed by the selected experts.

In the problem formulation, local router disagreement was written abstractly as $D(p_t, p_u)$. In the implemented geometry loss, this comparison is applied to a dispatch-aligned router distribution $\pi_{b,t}^{(\ell)}$. This distribution represents the expert weights that are actually used by the sparse MoE dispatch. When the backend exposes the selected expert indices and their dispatch weights, the sparse Top-K

weights are scattered into an E-dimensional vector, with zero mass assigned to experts that were not selected. If these sparse dispatch tensors are not exposed, the method falls back to the full router softmax:

$$\pi_{b,t}^{(\ell)} = \begin{cases} p_{\text{dispatch},b,t}^{(\ell)}, & \text{if sparse dispatch indices and weights are available,} \\ p_{b,t}^{(\ell)}, & \text{otherwise.} \end{cases} \tag{4.15}$$

For divergence-based router distances, $\pi_{b,t}^{(\ell)}$ is clamped with a small ε and renormalized before comparison. This avoids undefined logarithms or infinite divergences caused by zero entries in sparse dispatch-aligned distributions.

4.2 Router Disagreement Measure

The geometry loss requires a disagreement measure for comparing router states across nearby hidden-state representations. This measure should be symmetric, because local neighbourhood consistency has no natural direction, and sensitive to expert-support mismatch, because local routing inconsistency often appears as disagreement over which experts receive meaningful probability mass.

The main experiments use symmetric KL divergence as the probability-space router disagreement measure. Although it is not a metric in the strict mathematical sense, it is suitable here because it compares how two neighbouring tokens distribute probability mass over the expert pool.e

Algorithm 2 Router-disagreement computation for one graph edge

Require: Router logits $z_{b,t}^{(\ell)}$, $z_{b,u}^{(\ell)}$; optional sparse dispatch indices and weights; numerical constant ε

Ensure: Router disagreement $D_{\text{router}}^{(\ell,b)}(t,u)$

1: Construct $\pi_{b,t}^{(\ell)}$ from sparse dispatch weights if available; otherwise use $\text{softmax}(z_{b,t}^{(\ell)})$

2: Construct $\pi_{b,u}^{(\ell)}$ from sparse dispatch weights if available; otherwise use $\text{softmax}(z_{b,u}^{(\ell)})$

3: Clamp and renormalize both distributions to obtain $\hat{\pi}_{b,t}^{(\ell)}$ and $\hat{\pi}_{b,u}^{(\ell)}$ (Eq. 4.18)

4: Compute $D_{\text{router}}^{(\ell,b)}(t,u) = D_{\text{SKL}}\left(\hat{\pi}_{b,t}^{(\ell)}, \hat{\pi}_{b,u}^{(\ell)}\right)$ (Eqs. 4.16–4.17)

Distribution-space disagreement

Let $\pi_{b,t}^{(\ell)}$ and $\pi_{b,u}^{(\ell)}$ denote the dispatch-aligned router distributions for two neighbouring tokens in batch row b and layer ℓ, as defined in Section 4.1. To simplify notation in this subsection, the batch and layer indices are omitted, and the two distributions are written as π_t and π_u.

The symmetric KL divergence is defined as

$$D_{\mathrm{SKL}}(\pi_t, \pi_u) = \mathrm{KL}(\pi_t \| \pi_u) + \mathrm{KL}(\pi_u \| \pi_t), \tag{4.16}$$

where

$$\mathrm{KL}(\pi_t \| \pi_u) = \sum_{e=1}^{E} \pi_{t,e} \log \frac{\pi_{t,e}}{\pi_{u,e}}. \tag{4.17}$$

This choice is useful because it compares complete router distributions rather than only hard expert identities. A hard Top-K overlap metric can show whether two tokens selected the same experts, but it ignores how much probability mass was assigned to the remaining experts before or during sparse dispatch. Conversely, logit-space distances can measure numerical changes in router scores, but they do not directly express disagreement as probability mass over the expert pool. Symmetric KL is therefore closer to the intended object of regularization: the expert assignment distribution induced by the router.

The most important property of symmetric KL in this setting is its sensitivity to support mismatch. If π_t assigns meaningful probability to expert e, while π_u assigns almost none, then the corresponding KL term becomes large. In routing terms, this discourages nearby hidden states from assigning substantial probability mass to different expert subsets without a clear reason. This directly targets the local spraying behaviour discussed in Chapter 2, where neighbouring token representations may be scattered across unrelated experts even when aggregate load statistics appear balanced.

Because KL-based divergences are sensitive near zero, the distributions are numerically stabilized before comparison. For each distribution, entries are clamped by a small constant and then renormalized:

$$\hat{\pi}_{t,e} = \frac{\max(\pi_{t,e}, \varepsilon)}{\sum_{j=1}^{E} \max(\pi_{t,j}, \varepsilon)}, \qquad \hat{\pi}_{u,e} = \frac{\max(\pi_{u,e}, \varepsilon)}{\sum_{j=1}^{E} \max(\pi_{u,j}, \varepsilon)}. \tag{4.18}$$

The stabilized distributions $\hat{\pi}_t$ and $\hat{\pi}_u$ are then used when evaluating the KL terms. This avoids undefined logarithms and infinite divergences caused by zero entries in sparse dispatch-aligned distributions.

Final router disagreement

The final router-disagreement measure used by the geometry objective is symmetric KL applied to the stabilized dispatch-aligned router distributions:

$$D^{(\ell,b)}_{\text{router}}(t,u) = D_{\text{SKL}}\left(\hat{\pi}^{(\ell)}_{b,t}, \hat{\pi}^{(\ell)}_{b,u}\right). \tag{4.19}$$

This choice makes the geometry loss compare the expert-distribution states closest to the sparse dispatch decision, while still remaining differentiable with respect to the router probabilities. Router logit-scale stability is handled separately by the optional router z-loss, rather than by mixing a centered-logit term into the main geometry distance. In this way, the geometry term focuses specifically on local agreement in expert-assignment behaviour.

4.3 Geometry-Aware Routing Loss

As summarized in Figure 4.8, the geometry loss applies the router disagreement measure from Section 4.2 over the hidden-state graph constructed in Section 4.1. For each targeted layer, nearby token representations are connected by weighted graph edges, and the loss penalizes large router disagreement across those edges. In this way, the regularizer suppresses high-frequency variation of the router over local hidden-state neighborhoods, without forcing all tokens in the batch to use the same experts.

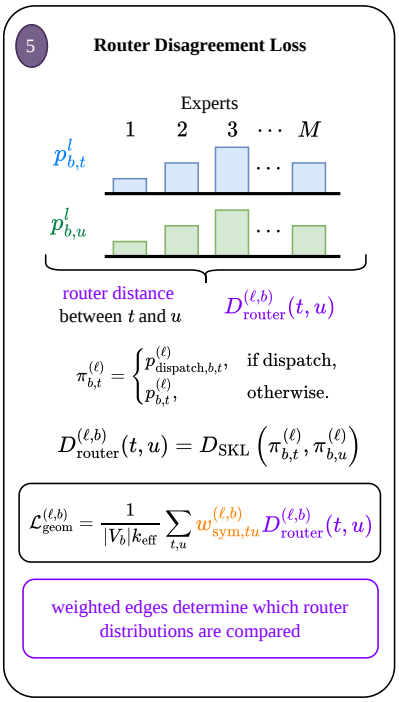


Figure 4.8: Step 5

For one batch row b and layer ℓ, let V_b denote the set of valid token positions. Let $\mathcal{E}^{(\ell,b)}_{\text{sym}}$ denote the symmetrized graph edge set, and let $w^{(\ell,b)}_{\text{sym},tu}$ denote the symmetrized edge weight between tokens t and u. The geometry loss for this batch row and layer is defined as the weighted average of router disagreement over the graph:

$$\mathcal{L}^{(\ell,b)}_{\text{geom}} = \frac{\sum_{(t,u)\in\mathcal{E}^{(\ell,b)}_{\text{sym}}} w^{(\ell,b)}_{\text{sym},tu} D^{(\ell,b)}_{\text{router}}(t,u)}{\sum_{(t,u)\in\mathcal{E}^{(\ell,b)}_{\text{sym}}} w^{(\ell,b)}_{\text{sym},tu} + \varepsilon}. \tag{4.20}$$

This formulation makes the role of each component explicit. The neighborhood graph determines which token pairs are compared. The symmetrized edge weight determines how strongly each pair contributes. The router disagreement measure determines which form of routing mismatch is penalized. The normalization makes the loss comparable across batch rows with different numbers of valid tokens or graph edges.

The total geometry loss averages this graph-level loss over valid batch rows and targeted MoE layers:

$$\mathcal{L}_{\text{geom}} = \frac{1}{|\mathcal{I}_{\text{target}}|} \sum_{\ell \in \mathcal{I}_{\text{target}}} \frac{1}{|\mathcal{B}_{\text{valid}}|} \sum_{b \in \mathcal{B}_{\text{valid}}} \mathcal{L}_{\text{geom}}^{(\ell,b)}. \tag{4.21}$$

Here, $\mathcal{I}_{\text{target}}$ is the set of targeted MoE layers, and $\mathcal{B}_{\text{valid}}$ is the set of batch rows with enough valid tokens to form at least one neighborhood edge.

Gradient flow and trainable parameters

The geometry loss updates the router through the same router-disagreement measure used in the final method. For one graph edge (t, u), the edge contribution can be written as

$$\ell_{tu} = w_{\text{sym},tu}^{(\ell,b)} D_{\text{SKL}} \left(\hat{\pi}_{b,t}^{(\ell)}, \hat{\pi}_{b,u}^{(\ell)} \right),$$

where $w_{\text{sym},tu}^{(\ell,b)}$ is the symmetrized edge weight and $\hat{\pi}_{b,t}^{(\ell)}$, $\hat{\pi}_{b,u}^{(\ell)}$ are the stabilized dispatch-aligned router distributions. The edge weight controls how strongly the local neighborhood relation contributes to the loss, while the symmetric KL term determines how much the two router distributions disagree.

The optimization path is

$$\mathcal{L}_{\text{geom}} \to D_{\text{SKL}}(\hat{\pi}_{b,t}^{(\ell)}, \hat{\pi}_{b,u}^{(\ell)}) \to \hat{\pi} \to z_{\text{aux}} \to W_\ell, b_\ell.$$

Thus, the geometry loss changes the router by changing the logits that produce the router distribution. The hidden states determine which graph edges exist and how strongly those edges are weighted, but they are detached in the geometry branch. Consequently, in router-only mode, the geometry loss does not update the hidden representations, the experts, the dense backbone, the embeddings, or the language-model head. It updates only the targeted router gate parameters.

For intuition, the symmetric KL term pushes neighboring router distributions toward agreement in both directions. If token t assigns substantial probability mass to an expert that token u assigns almost no mass to, the disagreement term increases. The reverse mismatch is also penalized because the divergence is symmetric. Therefore, the gradient encourages nearby hidden states to produce more

compatible expert-assignment distributions, while the detached graph ensures that this pressure is applied to the router map rather than to the representation space itself.

4.4 Confidence-Weighted Geometry Loss

The geometry loss assumes that hidden-state neighbours provide useful local consistency constraints. However, not every selected neighbour edge is equally reliable. Tokenization artifacts, punctuation, repeated formatting patterns, boilerplate text, or ambiguous hidden states can create high-similarity edges that should not strongly determine the router update. Confidence weighting addresses this issue by assigning a reliability factor $\rho_{tu}^{(\ell,b)}$ to each graph edge.

With confidence weighting, the geometry loss for one batch row b and layer ℓ becomes

$$\mathcal{L}_{\text{geom}}^{(\ell,b)} = \frac{\sum_{(t,u)\in\mathcal{E}_{\text{sym}}^{(\ell,b)}} \rho_{tu}^{(\ell,b)} w_{\text{sym},tu}^{(\ell,b)} D_{\text{router}}^{(\ell,b)}(t,u)}{\sum_{(t,u)\in\mathcal{E}_{\text{sym}}^{(\ell,b)}} \rho_{tu}^{(\ell,b)} w_{\text{sym},tu}^{(\ell,b)} + \varepsilon}. \tag{4.22}$$

Here, $w_{\text{sym},tu}^{(\ell,b)}$ is the symmetrized hidden-state edge weight, $D_{\text{router}}^{(\ell,b)}(t,u)$ is the router-disagreement measure, and $\rho_{tu}^{(\ell,b)}$ controls how strongly the edge contributes to the loss. The case $\rho_{tu}^{(\ell,b)} = 1$ for all edges recovers the unweighted geometry loss.

The reported experiments use entropy-based router confidence. The motivation is that a very uncertain router distribution should not be treated as a strong target for local agreement. If the router assigns nearly uniform probability mass across experts, then the model has not made a clear expert-assignment decision for that token. Forcing neighbouring tokens to agree with such an uncertain state could encourage premature or noisy specialization. In contrast, a sharper router distribution indicates that the router has a more definite assignment preference and can therefore provide a more reliable local smoothing signal.

For token t, router confidence is computed from normalized entropy:

$$H(p_t) = -\sum_{e=1}^{E} p_{t,e} \log p_{t,e}, \qquad c_t = \left(1 - \frac{H(p_t)}{\log E}\right)^{\gamma}. \tag{4.23}$$

The entropy $H(p_t)$ is largest when the router distribution is close to uniform and smaller when the distribution is concentrated on fewer experts. The normalized term $H(p_t)/\log E$ lies between zero and one, so c_t is high for confident, low-entropy router states and low for uncertain, high-entropy router states. The exponent γ controls how strongly confidence differences affect the geometry loss.

Larger values of γ make the weighting more selective by further down-weighting uncertain router states.

The edge confidence is obtained by averaging the confidence of the two endpoint tokens:

$$\rho_{tu}^{(\ell,b)} = \frac{1}{2}\left(c_{b,t}^{(\ell)} + c_{b,u}^{(\ell)}\right). \tag{4.24}$$

This makes an edge contribute more strongly when the router is confident about the assignments of the connected tokens, and less strongly when one or both endpoints are uncertain. The confidence term therefore does not change which neighbours are selected; it changes how much their router disagreement matters after the graph has already been constructed.

Confidence weighting is not required to define the core geometry-aware method. It is an additional control mechanism that makes the geometry loss more conservative. Without confidence weighting, all selected graph edges contribute according to their hidden-state similarity. With entropy confidence weighting, the regularizer focuses more strongly on neighbour pairs for which the router already has a clearer assignment preference. This helps reduce the risk that noisy or ambiguous neighbourhoods dominate the auxiliary objective.

4.5 Complete Training Objective

The geometry regularizer is not optimized in isolation. During post-training, the router must become more locally coherent without drifting away from the pretrained language-model behaviour. For this reason, the causal language-modeling objective remains part of the training loss. It anchors the adapted router to the original next-token prediction task and prevents the auxiliary router objectives from dominating the model's functional behaviour.

Task loss as behavioural anchor

Let $o_{b,t}$ denote the language-model logits for batch element b at position t, and let $y_{b,t+1}$ denote the next-token target. Positions with label -100 are ignored, following the standard masking convention for causal language-model training. The task loss is therefore

$$\mathcal{L}_{\text{task}} = \frac{\sum_{b,t} \mathbf{1}[y_{b,t+1} \neq -100]\, \mathrm{CE}(o_{b,t}, y_{b,t+1})}{\sum_{b,t} \mathbf{1}[y_{b,t+1} \neq -100]}. \tag{4.25}$$

Here, $\mathrm{CE}(o_{b,t}, y_{b,t+1})$ is the token-level cross-entropy,

$$\mathrm{CE}(o_{b,t}, y_{b,t+1}) = -\log \frac{\exp(o_{b,t,y_{b,t+1}})}{\sum_{v \in \mathcal{V}} \exp(o_{b,t,v})},$$

and $\mathcal{V}$ is the vocabulary.

The term $\mathcal{L}_{\text{task}}$ measures whether the post-trained model still predicts the next token correctly on the adaptation data. In the proposed method, it serves as the main behavioural constraint: the router should become smoother over hidden-state neighborhoods, but not at the cost of substantially degrading language-model quality.

In addition to the task loss and the geometry loss, some experiment families include optional router-level control terms. These terms are not the core contribution of the thesis. They are included as ablation dimensions to test how geometry-aware smoothing interacts with global load balancing, router-logit stabilization, soft importance uniformity, and sequence-level concentration control.

Optional global load balancing

Geometry smoothness can introduce a global-utilization risk. If large local regions become too coherent, they may route to too few experts, increasing concentration and reducing effective expert usage. For this reason, some experiment families pair local smoothness with an explicit global load-balancing objective.

For a target layer ℓ, let N be the number of valid tokens in the effective balance scope. In the implementation, this scope can be synchronized across distributed workers and accumulated across gradient-accumulation microsteps, so it approximates the optimizer-step global batch rather than a single microbatch. Let $\mathcal{S}_t^{(\ell)}$ be the hard selected expert set. The hard selected-expert frequency is

$$f_e^{(\ell)} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}\left[e \in \mathcal{S}_t^{(\ell)}\right], \tag{4.26}$$

and the mean router probability mass is

$$\bar{p}_e^{(\ell)} = \frac{1}{N} \sum_{t=1}^{N} p_{t,e}^{(\ell)}. \tag{4.27}$$

Following Switch Transformers [6], the Switch-style balancing term is

$$\mathcal{L}_{\text{bal}}^{(\ell)} = E \sum_{e=1}^{E} \mathrm{sg}\left(f_e^{(\ell)}\right) \bar{p}_e^{(\ell)}. \tag{4.28}$$

The hard frequency is detached because Top-K counts are discrete. Gradients flow through $\bar{p}_e^{(\ell)}$, which pushes probability mass away from overloaded experts.

The global scope is important. If load balancing is computed over a tiny micro-batch or a single short sequence, it can become an unintended instruction that every local context must use all experts uniformly. That directly fights specialization. Global-batch balancing instead asks for healthy utilization over a larger traffic sample, leaving local neighborhoods free to become coherent. This design follows the distinction emphasized in recent work on the granularity of MoE balancing [9].

Optional router z-loss

Router logits can become numerically unstable during sparse MoE training. The z-loss controls the scale of the router log-normalizer [7]. For token t,

$$\mathrm{LSE}(z_t) = \log \sum_{e=1}^{E} \exp(z_{t,e}), \tag{4.29}$$

and

$$\mathcal{L}_z^{(\ell)} = \frac{1}{N} \sum_{t=1}^{N} \mathrm{clip}(\mathrm{LSE}(z_t), -c_z, c_z)^2. \tag{4.30}$$

The z-loss does not encode geometry or specialization. Its role is numerical hygiene: it keeps the router from solving auxiliary objectives by letting logits grow or shrink pathologically. This is especially important when the geometry term changes the relative shape of router probabilities.

Optional importance uniformity

Importance uniformity penalizes deviations of the mean router probability mass from a uniform expert distribution. Let $\bar{p}_e^{(\ell)}$ denote the average router probability assigned to expert e over the relevant balancing scope. The importance-uniformity loss for layer ℓ is

$$\mathcal{L}_{\mathrm{imp}}^{(\ell)} = \sum_{e=1}^{E} \left(\bar{p}_e^{(\ell)} - \frac{1}{E} \right)^2. \tag{4.31}$$

This objective is softer than hard selected-expert balancing. It does not directly count how many tokens are routed to each expert after the Top-K decision. Instead, it acts on the router's average probability mass before or around the sparse selection step. It therefore encourages the router distribution to remain broadly spread across experts, but it does not by itself guarantee balanced realized traffic.

In the context of this thesis, importance uniformity is useful as a secondary diagnostic and ablation term: it tests whether controlling soft router mass provides additional stability when combined with local geometry regularization.

Optional sequence guard

Sequence guarding is introduced to prevent a possible failure mode of local smoothing. A geometry-aware loss encourages nearby hidden states to have similar router distributions. If many tokens within a sequence form a coherent region, the geometry loss may push them toward the same small subset of experts. This can improve local consistency, but it can also cause one expert to dominate an entire sequence. Sequence guarding limits this behaviour by penalizing excessive within-sequence concentration.

For a sequence b of length L_b, let $c_{b,e}^{(\ell)}$ be the number of tokens in that sequence assigned to expert e at layer ℓ:

$$c_{b,e}^{(\ell)} = \sum_{t=1}^{L_b} \mathbf{1}\left[e \in \mathcal{S}_{b,t}^{(\ell)}\right]. \tag{4.32}$$

The corresponding sequence-level expert frequency is

$$f_{b,e}^{(\ell)} = \frac{c_{b,e}^{(\ell)}}{L_b}. \tag{4.33}$$

For Top-K routing with $K > 1$, these per-expert frequencies should be interpreted as selection rates, not as a probability distribution over experts, because one token may contribute to more than one expert count.

The sequence guard applies a quadratic penalty only when this frequency exceeds a maximum allowed threshold $\rho_{\max}$:

$$\mathcal{L}_{\text{seq}}^{(\ell)} = \frac{1}{B}\sum_{b=1}^{B}\sum_{e=1}^{E} \max\left(0, f_{b,e}^{(\ell)} - \rho_{\max}\right)^2. \tag{4.34}$$

This term is not intended to make every sequence use all experts uniformly. That would conflict with the specialization goal discussed in Chapter 2. Instead, it acts as a guardrail against extreme sequence-level monopolization. Below the threshold $\rho_{\max}$, no penalty is applied. Above the threshold, the penalty increases smoothly, discouraging a single expert from dominating too much of a sequence. Sequence guarding is therefore complementary to global load balancing: global balancing controls aggregate usage across the batch or optimizer step, while sequence guarding prevents local smoothing from concentrating too much of an individual sequence into one expert.

Final loss function and scheduling

The complete objective combines the language-modeling loss, the proposed geometry regularizer, the router z-loss, and any optional auxiliary terms. At optimizer step s, the objective is

$$\mathcal{L}(s) = \mathcal{L}_{\text{task}} + \lambda_{\text{geom}}(s)\mathcal{L}_{\text{geom}} + \lambda_{\text{bal}}(s)\mathcal{L}_{\text{bal}} + \lambda_z(s)\mathcal{L}_z + \lambda_{\text{imp}}(s)\mathcal{L}_{\text{imp}} + \lambda_{\text{seq}}(s)\mathcal{L}_{\text{seq}}. \tag{4.35}$$

If an optional term is not used in a particular experiment, its coefficient is set to zero. The base geometry method uses the task loss and the geometry loss, with entropy confidence weighting applied inside the geometry term. The reported families then add z-loss, importance uniformity, sequence guard, or explicit load balancing depending on the comparison.

The coefficients are scheduled during training. For a loss component i, the coefficient at step s is

$$\lambda_i(s) = \lambda_{i,\text{base}} \min\left(1, \frac{s+1}{S_{i,\text{warm}}}\right), \tag{4.36}$$

where $\lambda_{i,\text{base}}$ is the target coefficient and $S_{i,\text{warm}}$ is the number of warmup steps.

Warmup is important because router probabilities and nearest-neighbor graphs can be noisy at the beginning of post-training. If the geometry term is applied at full strength immediately, the model may smooth accidental neighborhoods or reinforce unstable early router decisions. Gradually increasing λ_{geom} allows the task loss, load balancing, and router stabilization terms to keep the model anchored while the local smoothness constraint becomes active. In some configurations, the geometry coefficient can also be decayed later in training, allowing the method to first shape the router map and then consolidate the adapted routing behaviour under the task and balancing objectives.

The objective therefore reflects the central design principle of the proposed method. The task loss preserves language-model behaviour, the geometry loss encourages local routing consistency, the load-balancing loss prevents global collapse, the z-loss stabilizes router logits, and the optional terms test additional forms of expert-usage control. The method is not intended to optimize smoothness alone, but to improve the trade-off between local coherence, expert utilization, and predictive quality.

4.6 Training Procedure

The training procedure follows the objective in Section 4.5. Each optimizer step combines the causal language-modeling loss with router-level auxiliary losses com-

puted from the targeted MoE layers. The ordinary model forward pass remains unchanged: the auxiliary geometry branch uses detached pre-router hidden states to construct the neighbourhood graph, so gradients update only the trainable router parameters.

Algorithm 3 summarizes one optimizer step. It separates the standard sparse MoE forward pass from the auxiliary router-control branch used during post-training.

Algorithm 3 Router-focused geometry-aware post-training step

Require: Batch x, targeted MoE layers $\mathcal{I}_{\text{target}}$, scheduled coefficients $\lambda_i(s)$
Ensure: Updated router gate parameters

Run the pretrained sparse MoE model on a sequence or packed-block batch.
Compute the shifted causal language-modeling loss $\mathcal{L}_{\text{task}}$.
Capture pre-gate hidden states $h_{b,t}^{(\ell)}$ for all $\ell \in \mathcal{I}_{\text{target}}$.
for each targeted layer $\ell \in \mathcal{I}_{\text{target}}$ **do**
    Recompute router logits from detached pre-gate states: $z_{b,t,\text{aux}}^{(\ell)} \leftarrow g_\ell(\text{sg}(h_{b,t}^{(\ell)}))$.
    Construct the dispatch-aligned router distributions $\pi_{b,t}^{(\ell)}$.
    Build the windowed hidden-state kNN graph independently within each batch row.
    Compute the confidence-weighted symmetric-KL geometry loss.
    Compute enabled auxiliary terms: router z-loss, importance uniformity, sequence guard, or explicit global load balancing.
end for
Average auxiliary losses over valid batch rows and targeted MoE layers.
Apply scheduled coefficients and form the total loss $\mathcal{L}(s)$.
Backpropagate the total loss.
Clip gradients and update only the trainable router gate parameters.

Thus, the model still produces outputs through the standard sparse Top-K MoE dispatch. The additional geometry branch only supplies a post-training signal that reshapes the router map.

Algorithm 3 is the operational version of the choices summarized at the start of the chapter in Table 4.1.

Capacity implications

The geometry loss changes the router's intended expert assignments, but practical sparse MoE systems also operate under finite expert capacity, load-balancing, and systems-level dispatch constraints [6, 7, 49, 30]. If many tokens are assigned to the same expert, the execution backend may drop overflow tokens, reroute them to available experts, or queue them, depending on the implementation. Capacity should therefore be treated as a separate systems constraint from local router smoothness.

The proposed method separates two concerns:

- $\mathcal{L}_{\text{geom}}$ encourages nearby hidden states to produce compatible router distributions;
- global expert-usage controls, such as $\mathcal{L}_{\text{bal}}$ when enabled, discourage aggregate traffic from concentrating too heavily on a small subset of experts.

The capacity implications depend on the backend. If token dropping is used, excessive geometry strength can increase overflow risk by assigning many related tokens to the same expert. If rerouting is used, excessive concentration can still be harmful because the realized expert may differ from the router's intended expert, weakening specialization. If queueing is used, the main cost may appear as latency rather than dropped computation. Capacity metrics should therefore be interpreted jointly with local routing metrics, expert-load metrics, and language-model quality rather than treated as an afterthought.

Geometry Hyperparameters and Trade-offs

The geometry objective is controlled by four main hyperparameters. Each one affects the trade-off between local router coherence, expert specialization, global utilization, and language-model quality.

These parameters should not be treated as monotonic quality knobs. The useful regime is expected to be moderate: the geometry term should improve local routing coherence without erasing useful expert specialization, concentrating too much load, or degrading language-model quality. The method is therefore evaluated as a multi-objective trade-off rather than as a single-metric improvement.

Table 4.2: Geometry hyperparameters and their main trade-offs.

Hyperparameter	Low / small value	High / large value	Main risk
Neighbour count k	More local, cheaper, less noisy	More smoothing signal, broader neighbourhood	Too many weak or superficial neighbours
Candidate window W	Fewer accidental long-range edges	Allows broader in-sequence matches	Too small misses useful neighbours; too large adds noisy edges
Edge temperature τ	Sharp edge weights; strongest neighbours dominate	Softer weights across neighbours	Too sharp overtrusts noisy edges; too soft oversmooths
Geometry strength λ_{geom}	Weak regularization; little routing change	Stronger local smoothing	Over-smoothing, capacity concentration, or smoothly wrong routing

4.7 Evaluation Quantities

The metrics are grouped into two categories. Internal diagnostics measure properties of the router itself, such as global expert usage, routing concentration, and local assignment continuity. External metrics measure whether the adapted model still performs the language modeling and downstream tasks well. This distinction is important because a routing intervention can improve an internal diagnostic while still harming predictive quality, or preserve perplexity while producing undesirable routing structure. For readability, batch and layer indices are omitted when the meaning is clear.

Internal router diagnostics

Load coefficient of variation measures how evenly selected expert assignments are distributed across the expert pool. For each routed layer, expert load is computed by counting selected expert assignments over valid tokens. Let $\mathcal{S}_{b,t}^{(\ell)}$ be the selected Top-K expert set for token t in batch row b, and let V_b be the valid-token set. The count for expert e is

$$c_e^{(\ell)} = \sum_b \sum_{t \in V_b} \mathbf{1}\left[e \in \mathcal{S}_{b,t}^{(\ell)}\right].$$

The mean and standard deviation of expert load are

$$\mu_c^{(\ell)} = \frac{1}{E}\sum_{e=1}^{E} c_e^{(\ell)}, \qquad \sigma_c^{(\ell)} = \sqrt{\frac{1}{E}\sum_{e=1}^{E}\left(c_e^{(\ell)} - \mu_c^{(\ell)}\right)^2}.$$

The load coefficient of variation is

$$\mathrm{LoadCV}^{(\ell)} = \frac{\sigma_c^{(\ell)}}{\mu_c^{(\ell)} + \varepsilon}.$$

Lower values indicate more even expert usage. In this thesis, Load CV is used as a collapse and global-utilization diagnostic. A geometry-aware method should not improve local consistency by concentrating too much traffic on a small subset of experts.

Routing entropy measures how concentrated or diffuse the router probability distribution is. For router probabilities $p_{t,e}$, token-level entropy is

$$H_t = -\sum_{e=1}^{E} p_{t,e} \log p_{t,e}.$$

The reported entropy is averaged over valid tokens and routed layers. Higher entropy means the router distribution is less concentrated across experts, while lower entropy means the router is more peaked.

Entropy complements Load CV. Load CV measures realized selected-expert usage, while entropy measures the softness of the router distribution before or around sparse selection. In this thesis, entropy is used to detect whether geometry regularization makes routing more broadly distributed or more concentrated. However, entropy is not interpreted as a standalone quality metric: very high entropy can also indicate unstructured uncertainty rather than useful expert specialization.

Routing jitter measures local instability in selected expert assignments. Let $\mathcal{S}_t$ be the selected expert set for token t. The reported adjacent-token jitter is the fraction of valid adjacent token pairs whose selected expert sets are disjoint:

$$\mathrm{Jitter} = \frac{\sum_t \mathbf{1}[\mathcal{S}_t \cap \mathcal{S}_{t+1} = \emptyset]}{\#\{\text{valid adjacent token pairs}\}}.$$

Lower jitter indicates more continuity in adjacent-token expert assignment. This metric is a simple operational proxy for routing instability: if many adjacent valid tokens select disjoint expert sets, the router may be changing too abruptly across the sequence. It does not capture all forms of local geometry consistency, but it provides a useful diagnostic for whether routing becomes less fragmented after geometry-aware post-training.

External model-quality metrics

Evaluation loss and perplexity. Language-model quality is measured by evaluation cross-entropy loss and perplexity:

$$\mathrm{PPL} = \exp(\mathcal{L}_{\mathrm{eval}}).$$

A useful router intervention should not substantially increase evaluation loss or perplexity. In this thesis, perplexity acts as a quality-preservation metric. It checks whether the adapted router still supports the pretrained model's next-token prediction behaviour.

Prompt Sensitivity Score has been studied through prompt-variant and prompt-format evaluation protocols [13, 68]. In this thesis, Prompt Sensitivity Score (PSS) denotes a mean absolute pairwise score-difference metric over prompt variants (ProSA - PromptSensiScore). It measures how much model performance varies across different prompt formulations of the same benchmark instance. For an instance evaluated under m prompt variants with scores $s_1, \ldots, s_m$, the score is computed as the mean absolute pairwise score difference:

$$\mathrm{PSS} = \frac{2}{m(m-1)} \sum_{i<j} |s_i - s_j|.$$

Lower PSS means that the model's score varies less across prompt variants, and therefore that the model is less sensitive to prompt wording under this evaluation protocol. The reported overall PSS is averaged over benchmark-level PSS values.

PSS is relevant because one motivation for geometry-aware routing is to reduce abrupt internal expert switching under small prompt-induced representation changes. However, lower PSS is not automatically better if it is achieved by making the model uniformly worse or less responsive. For this reason, PSS is interpreted together with downstream benchmark score.

Benchmark score and accuracy measure ordinary downstream performance. These metrics are treated as behavioural guardrails: a configuration is not useful if it improves routing diagnostics or PSS while substantially damaging downstream task performance. In the testing chapter, downstream scores are therefore interpreted together with perplexity, Prompt Sensitivity Score, and the internal router diagnostics.

4.8 Expected Effects

The expected outcome of the proposed method is not that every metric improves simultaneously. Geometry-aware routing introduces an additional structural constraint on the router, so the result should be interpreted as a trade-off between local coherence, global expert utilization, and language-model quality. A successful configuration should therefore show that the router becomes more locally organized without causing expert collapse or unacceptable degradation in predictive performance.

The expected evidence chain is as follows:

1. The geometry objective is active during training, meaning that valid neighbourhood edges are formed and the weighted router-disagreement loss is non-zero;
2. Local router organization changes, shown by lower neighbour router disagreement, increased Top-K overlap among nearby hidden states, or reduced local routing jitter;
3. Global routing health is preserved or improved, reflected by acceptable expert-load coefficient of variation, active expert usage, and routing entropy;
4. Behavioural stability improves where possible, especially through reduced prompt sensitivity under prompt-variant evaluation;
5. Language-model quality remains acceptable, as measured by evaluation loss, perplexity, and downstream benchmark score.

The method therefore requires a layered evaluation. Local kNN router divergence, Top-K neighbour overlap, and routing jitter test whether hidden-state neighbourhoods induce more consistent routing behaviour. Expert-load coefficient of variation and routing entropy test whether this local coherence is compatible with healthy expert utilization. Perplexity and benchmark score test whether the adapted router preserves language-model quality. Prompt Sensitivity Score tests whether internal routing changes correspond to more stable behaviour across prompt variants. Edge-quality diagnostics help verify that the geometry objective is acting on meaningful neighbourhoods rather than only on noisy formatting or tokenization artifacts.

The central expected effect is therefore a better operating point, not a universal improvement. The desired result is a router that is locally smoother over hidden-state neighbourhoods, globally usable across the expert pool, and still compatible with the pretrained language model's predictive behaviour.

4.9 Chapter Summary and Position Relative to Prior Work

This chapter introduced **Geometry-Aware Router Post-training (GARP)**, a router-only post-training method for sparse MoE models. GARP addresses the gap that global load balancing can keep expert usage healthy, but does not require nearby hidden states to produce compatible routing decisions. It adds this local constraint by building a per-layer kNN graph from detached pre-gate hidden states and penalizing router-distribution disagreement across high-similarity edges.

GARP does not change the expert architecture, token-choice Top-K routing, or inference-time sparse dispatch. The pretrained MoE forward pass is preserved, and only the router gates are updated. The geometry loss acts on dispatch-aligned router distributions using symmetric KL divergence. Optional terms from Section 4.5 test interactions with load balancing, router z-loss, importance uniformity, and sequence guarding.

The intended trade-off is local smoothness without erasing expert specialization, overloading capacity, or degrading language-model behaviour. Compared with RoMA [11], which aligns routing weights using successful neighbours in a task-embedding space during router post-training, GARP constructs token-level hidden-state neighbourhoods inside the model and uses them only as an auxiliary training signal. Compared with SIMBAL [10], which proposes a similarity-preserving load-balancing loss, GARP is not primarily a new balancing mechanism; it is a router post-training regularizer based on hidden-state neighbourhoods. Compared with graph-of-token methods [12], its graph is only an auxiliary training signal, not part of the inference-time forward pass.

Chapter 5 tests whether GARP improves local router organization while preserving expert-load health, perplexity, downstream performance, and prompt-sensitivity behaviour.

Chapter 5

Testing

5.1 Testing Objective and Evidence Structure

This chapter evaluates the geometry-aware router regularization method proposed in Chapter 4. The preceding chapters identified a structural limitation in sparse Mixture-of-Experts (MoE) routing: aggregate expert utilization can appear acceptable while nearby hidden states still induce inconsistent routing decisions. The proposed method addresses this gap by adding a local hidden-state geometry constraint to the router objective while preserving the standard sparse MoE forward pass.

The central testing question is:

> Does a router-only regularizer based on windowed k-nearest-neighbour graphs of detached pre-router hidden states improve sparse MoE routing in a useful sense?

Here, "geometry" means the local neighbourhood structure induced by cosine similarity between detached pre-router hidden states inside each targeted MoE layer. "Useful" does not mean higher raw benchmark accuracy by itself. It means that the intervention measurably improves router structure, such as lower neighbour router disagreement and healthier expert-load distribution, while preserving validation loss, perplexity, downstream score, and sparse-execution feasibility within acceptable limits.

The empirical claim tested in this chapter is deliberately specific. The method is not evaluated as a general raw-accuracy improvement technique. Instead, the chapter

tests whether geometry-aware regularization reshapes the router map in a measurable and useful way: neighbouring hidden states should receive more similar router outputs, expert usage should become less imbalanced, prompt sensitivity should decrease when possible, and ordinary model quality should remain close to the reference setting at moderate regularization strength.

The central empirical argument is therefore:

> Geometry-aware routing is evaluated here as a controllable change to router behaviour: it should alter local and aggregate routing diagnostics, may reduce prompt sensitivity, and must remain practical in the router-only regime.

The evidence is organized into main evidence and supporting evidence. The main evidence consists of the primary router-only comparison, the global-load-balancing control, and the geometric-coefficient sweep. These sections test the central mechanism, the main alternative explanation, and the strength–cost trade-off. Supporting evidence consists of the configuration analysis, component ablations, matched-seed checks, trainable-scope sensitivity, model/data sweeps, and computational feasibility analysis. These sections explain how the method behaves under different controls and how far the observed pattern extends beyond the primary comparison.

Table 5.1 summarizes how each claim is tested. The purpose of the table is to make the claim-to-evidence structure explicit before the numerical results are introduced.

Table 5.1: Evidence strategy for the testing chapter.

Claim	Evidence used	Main location
The geometric objective is active	Non-zero weighted geometry loss, active neighbours, and active token fraction.	Section 5.3.
Geometry changes router structure	Lower Load CV, higher router entropy, and lower router-neighbour logit distance.	Sections 5.3 and 5.5.
Geometry is not equivalent to global load balancing	Geometry is compared with explicit global balancing alone and with geometry added on top of global balancing.	Section 5.4.
Geometry introduces a controllable trade-off	Local router structure and prompt sensitivity improve as geometry strength increases, while drift, jitter, or perplexity can worsen.	Section 5.5.
Supporting components control the intervention	Component removals test confidence weighting, router-logit stabilization, sequence guard, and importance uniformity.	Section 5.6.
The effect is not isolated to one run	Matched-seed checks, trainable-scope sensitivity, and model/data sweeps.	Section 5.7.
The method is computationally feasible	Runtime, throughput, peak memory, and trainable-parameter summaries on one RTX 6000 Blackwell-series GPU.	Section 5.8.

Detailed tables, task-level results, matched-seed results, sensitivity checks, model/data sweeps, and computational measurements are reported in the appendices. Appendix A defines the appendix reporting conventions and metric definitions. Full main-router evidence is provided in Appendix B. Sensitivity and external-validity results are provided in Appendix C. Computational-overhead measurements are provided in Appendix D. A separate pretraining pilot is reported in Appendix E; it is treated as supporting evidence rather than part of the main router-only post-training claim.

5.2 Experimental Setup

Model, Trainable Boundary, and Data

All main experiments use `allenai/OLMoE-1B-7B-0125-Instruct` as the base model with the `olmoe` backend. This model was selected because it is an open-weight sparse MoE language model with an accessible router implementation.

The main experiments follow the trainable boundary defined in Section 4.1. Only router gate parameters are trainable. Attention layers, token embeddings, expert weights, and the language-model head remain frozen. This isolates the trainable change to the routing policy while preserving the pretrained backbone and expert functions. Therefore, differences in router diagnostics, prompt sensitivity, and downstream behaviour are interpreted primarily as consequences of router-only finetuning under the selected auxiliary objectives. This attribution remains empirical rather than absolute because seed variation, evaluation noise, dataset slicing, and metric definitions can still affect measured outcomes.

The main router-finetuning dataset is `Felladrin/ChatML-ultrachat_200k`. The training split uses `train[:95%]`, and the held-out validation split uses `train[95%:]`. The text field is `text`. This dataset is used for router adaptation only. Downstream behaviour is evaluated separately using task datasets.

The reported local training runs were executed on a single RTX 6000 Blackwell-series GPU. Therefore, the GPU-hour values reported in this chapter and Appendix D are interpreted as single-GPU runtime hours for that hardware setting.

A later trainable-scope sensitivity analysis deliberately relaxes the router-only boundary. Those runs test whether the geometry-associated router pattern disappears when more parameters are allowed to move. They are less clean methodologically because attention layers, normalization parameters, experts, or full transformer blocks may also change. For this reason, the main thesis claim is grounded in the router-only experiments, while the wider-scope results are treated as sensitivity evidence.

Training Configuration

Table 5.2 summarizes the fixed training configuration used across the main router-only finetuning experiments. The purpose of the table is to show that the main comparisons differ in the router objective rather than in training scale, optimizer, sequence length, or hardware setting. The full configuration naming scheme used

in the appendix is provided in Appendix A.

Table 5.2: Fixed configuration for the main router-only finetuning experiments.

Field	Value
Base model	`allenai/OLMoE-1B-7B-0125-Instruct`
Backend	`olmoe`
Hardware	Single RTX 6000 Blackwell-series GPU
Trainable scope	Router only
Frozen components	Attention, embeddings, experts, LM head
Training steps	5000
Primary seed	42
Micro batch size	1
Gradient accumulation	8
Effective batch size	8 sequences
Optimizer	AdamW
Learning rate	3×10^{-5}
Warmup	500 steps
Weight decay	0.01
Maximum gradient norm	0.5
Maximum sequence length	1024
Geometry confidence mode	`entropy`
Geometry confidence power	2.0
Geometry confidence minimum	0.0

Because all main runs use the same training horizon, batch structure, optimizer, and sequence length, the main comparisons focus on the effect of enabling or modifying the geometry-aware router objective.

Downstream Evaluation

Downstream behaviour is evaluated with ProSA-style prompt-sensitivity runs. ProSA evaluates each task under multiple prompt variants and reports both a task score and a prompt sensitivity score. The task score measures average task performance. The prompt sensitivity score measures how much the model's behaviour changes when the prompt wording changes. Lower prompt sensitivity is better.

The downstream task suite contains ARC-Challenge, CommonsenseQA, and HumanEval. ARC-Challenge and CommonsenseQA test multiple-choice reasoning under prompt variation. HumanEval is included as a qualitatively different code-generation setting, but it is interpreted cautiously because the evaluated subset is small and the absolute scores are low.

Configuration Names

The configuration names used throughout the chapter are descriptive rather than run-id based. Table 5.3 gives the short naming scheme used in the main results. The full naming scheme remains available in Appendix A.

Table 5.3: Configuration names used in the testing chapter.

Configuration name	Components
Baseline	No geometry.
Geometry only	Geometry + confidence weighting.
Geometry + z-loss	Geometry + confidence weighting + router z-loss.
Geometry + z-loss + importance uniformity	Score-oriented family.
Geometry + z-loss + sequence guard	Prompt-sensitivity-oriented family.
Full controlled geometry	Geometry + confidence weighting + z-loss + importance uniformity + sequence guard.
Geometry without confidence weighting	Full controlled geometry, confidence weighting removed.
Global load balancing only	Global balancing + z-loss, no geometry.
Global load balancing with geometry	Global balancing + z-loss + geometry.

Explicit global load balancing is treated as a control family rather than as an active component in every geometry-enabled configuration. It is therefore reported separately in Section 5.4 to test whether the geometry effect can be explained by ordinary expert-load regularization.

Two additional breadth analyses are used after the main result sections, as summarized in Table 5.4. These analyses support external validity and consistency checks rather than seed-level reproducibility.

Table 5.4: Additional breadth analyses used in the testing chapter.

Analysis	Changes	Purpose
Trainable-scope sensitivity	Trainable parameter boundary	Tests router-only attribution.
Model/data sweep	Adaptation data or MoE backend	Tests external validity.

Metrics and Reporting Rules

The evaluation combines four metric groups. Model-quality preservation is measured with validation loss, perplexity, and mean downstream score. Router-structure change is measured with Load coefficient of variation, router entropy, and router-neighbour logit distance. Prompt robustness is measured with prompt sensitivity score. Cost and preservation risks are measured with jitter, base-router drift, and anchor KL. Full metric definitions and directionality conventions are provided in Appendix A and Appendix A.

Table 5.5 gives the main interpretation of each metric. The table is not only descriptive; it defines how the numerical results should be read in the rest of the chapter.

Two distinctions are important for interpreting the metrics. First, router-neighbour logit distance is an evaluation diagnostic, not necessarily the exact distance used inside the geometry loss. The training objective in Chapter 4 compares router distributions over hidden-state neighbourhoods. The diagnostic reported here measures whether neighbouring hidden states receive more similar pre-softmax router outputs after training. This makes the local smoothness effect easier to interpret directly at the router-map level.

Second, local router-neighbour smoothness and jitter are not the same robustness concept. The geometry objective acts on selected hidden-state neighbourhoods, whereas the jitter diagnostic measures routing sensitivity under a separate perturbation test. A model can therefore become smoother over hidden-state neighbours while still becoming more sensitive under the perturbation diagnostic. This distinction becomes important in Section 5.3.

Confidence intervals are reported where raw evaluation samples are available. Unless otherwise stated, these intervals summarize uncertainty over evaluation samples or matched diagnostic evaluations; they should not be interpreted as full seed-to-seed uncertainty. This is especially important for single-run comparisons, where narrow intervals can reflect a large number of evaluation samples rather than independent training replications. Validation loss and perplexity are reported as point estimates in some sections because the saved standard evaluation contains aggregate values rather than raw per-batch or per-token losses.

Throughout the chapter, the term “stability” is avoided as a single aggregate claim. Instead, the results are described using more specific diagnostics: local router smoothness, prompt sensitivity, jitter, validation-quality preservation, and base-router drift. These quantities can move in different directions and are interpreted separately.

Table 5.5: Metrics used in the testing chapter.

Metric	Interpretation	Desired direction
Validation loss	Held-out causal language-model quality on the router-finetuning validation split.	Lower.
Perplexity	Exponentiated validation loss; another measure of ordinary language-modeling quality.	Lower.
Mean downstream score	Average task performance across the evaluated downstream tasks.	Higher.
Load coefficient of variation	Relative imbalance in expert usage across the evaluated tokens.	Lower.
Router entropy	Spread of router probability mass across experts. Higher entropy indicates less concentrated routing when quality is preserved.	Higher, when not accompanied by quality degradation.
Router-neighbour logit distance	Difference between router logits of neighbouring hidden-state representations. This measures local router-map disagreement.	Lower.
Prompt sensitivity score	Average variation in model behaviour across prompt variants for the same underlying task instance.	Lower.
Jitter	Routing sensitivity under a separate perturbation diagnostic. It measures perturbation sensitivity rather than neighbourhood smoothness.	Lower.
Base-router drift / anchor KL	Deviation from the pretrained router reference, measured either as an L_2-style drift diagnostic or as a KL-based anchoring diagnostic.	Lower.

5.3 Main Evidence I: Does Geometry Change Router Behaviour?

This section evaluates the effect of enabling the proposed geometric loss. The baseline uses $\lambda_{\text{geom}} = 0$, where the geometric loss is inactive. The geometry-enabled setting uses $\lambda_{\text{geom}} = 100$. The comparison isolates the direct effect of the geometry-aware objective before considering the supporting components. Full primary-comparison diagnostics are reported in Appendix B.

This is the cleanest mechanism test in the chapter because the trainable boundary is router-only. The test asks three questions: whether router structure changes in

the intended direction, whether ordinary model quality is preserved, and whether the router-level change corresponds to improved prompt sensitivity or raw downstream score.

Router-Structure Effects

Table 5.6 tests the central mechanism: whether the geometry-enabled router produces a more locally coherent and less imbalanced routing map. These metrics evaluate the local router-smoothness property introduced in Section 4.1 and optimized through the geometry loss in Section 4.3. Extended router-structure diagnostics are provided in Appendix B.

Table 5.6: Router-structure effects of enabling the geometric loss. Estimates are reported with 95% confidence intervals.

Metric	Baseline	Geometry	Change
Router entropy	3.8327 [3.8325, 3.8329]	**3.9193** [3.9191, 3.9194]	+0.0866 [0.0863, 0.0869]
Load coefficient of variation	0.9439 [0.9426, 0.9451]	**0.8223** [0.8212, 0.8232]	−0.1216 [-0.1236, -0.1196]
Router-neighbour logit distance	15.4716 [15.4449, 15.5007]	**14.0462** [14.0308, 14.0604]	−1.4254 [-1.4688, -1.3859]

The router-structure results support the central mechanism. Load coefficient of variation decreases by approximately 12.9%, indicating that expert usage becomes more balanced. Router entropy increases by approximately 2.3%, indicating that routing becomes less concentrated. Router-neighbour logit distance decreases by approximately 9.2%, showing that neighbouring hidden-state representations receive more similar router outputs after geometry-aware training.

Because the trainable boundary is router-only, these changes are best interpreted as a reshaping of the routing policy rather than as a general representation-learning effect. The matched reduction in router-neighbour logit distance is especially important because the diagnostic is measured on the same evaluation set and router-logit scale in both conditions. Together, these changes provide the strongest evidence that the geometric objective affects the intended part of the model: the router map itself.

Model Preservation and Downstream Behaviour

The purpose of table 5.7 is to test whether the structural router change is obtained without damaging ordinary model quality. Task-level downstream details are kept in Appendix B and Appendix B to keep the main chapter focused on the central evidence.

Table 5.7: Model-quality, prompt-sensitivity, and downstream effects in the primary geometry comparison. Estimates are reported with 95% confidence intervals.

Metric	Baseline	Geometry	Change
Validation loss	3.2466 [3.2401, 3.2533]	3.2464 [3.2404, 3.2523]	−0.0003 [-0.0096, 0.0097]
Perplexity	25.7051 [25.5371, 25.8770]	25.6976 [25.5456, 25.8491]	−0.0074 [-0.2459, 0.2476]
Jitter	0.6424 [0.6418, 0.6429]	0.6635 [0.6628, 0.6640]	+0.0211 [0.0202, 0.0219]
Prompt sensitivity score	0.2511 [0.2505, 0.2518]	**0.2472** [0.2464, 0.2479]	−0.0040 [-0.0047, -0.0033]
Mean downstream score	0.2667 [0.2661, 0.2672]	0.2669 [0.2663, 0.2675]	+0.0002 [-0.0007, 0.0010]

Validation loss and perplexity remain close to the baseline, and both change intervals include zero. Mean downstream score is also essentially unchanged. The individual task scores are mixed: ARC-Challenge decreases slightly, while CommonsenseQA and HumanEval increase slightly.

The clearest downstream-related effect is therefore reduced prompt sensitivity rather than improved raw task accuracy. Prompt sensitivity decreases from 0.2511 to 0.2472, while mean downstream score remains almost unchanged. This supports the interpretation that geometry-aware routing primarily affects behavioural consistency under prompt variation rather than average task performance.

The main cost in the primary comparison is jitter, which increases from 0.6424 to 0.6635. This does not contradict the router-neighbour result because the two diagnostics test different properties. The geometry objective smooths router outputs over selected hidden-state neighbourhoods, whereas jitter measures routing sensitivity under a separate perturbation diagnostic.

Figure 5.1 summarizes the same primary comparison as an improvement-oriented effect profile. Unlike a signed-change plot, the figure converts each metric so that positive values indicate improvement and negative values indicate degradation.

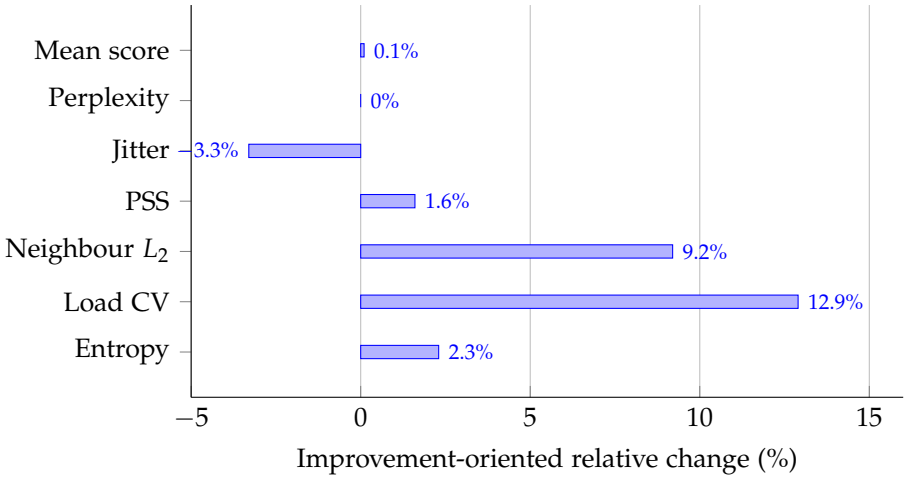


Figure 5.1: Improvement-oriented effect profile of enabling the geometric loss. Positive values indicate improvement for the corresponding metric, and negative values indicate degradation.

Primary Claim-to-Evidence Conclusion

The primary test supports the first main claim of the chapter: local hidden-state geometry provides a measurable router-level signal. Enabling the geometric loss changes router structure in the intended direction: expert usage becomes more balanced, router entropy increases, and neighbouring hidden states receive more similar router logits. The same comparison also shows a modest reduction in prompt sensitivity while preserving validation loss, perplexity, and mean downstream score.

5.4 Main Evidence II: Is Geometry More Than Load Balancing?

The primary test shows that the geometric loss improves expert-load balance. This raises a possible alternative explanation: the geometric loss might simply act as another form of load balancing. This section tests that explanation under one explicit global-balancing setting. Full global-balancing control diagnostics are provided in Appendix B.

The control logic is straightforward. If geometry were merely duplicating ordinary load balancing, then explicit global balancing without geometry should reproduce the same improvements in Load CV, router-neighbour distance, and prompt sensitivity. Conversely, if geometry provides a distinct local signal, then geometry

should improve local router diagnostics that global balancing alone does not substantially change, and adding geometry on top of global balancing should still produce additional router-structure improvements.

This control corresponds to the distinction made in Section 4.5: global balancing controls aggregate expert usage, while geometry controls local variation of the router map.

Table 5.8 directly compares the baseline, the tested global-balancing setting without geometry, and the geometry-enabled setting without explicit global balancing.

Table 5.8: Direct comparison between the baseline, explicit global balancing without geometry, and geometry without explicit global balancing.

Metric	Baseline	Global only	Geometry only
Router entropy	3.8327	3.8334	**3.9193**
Load coefficient of variation	0.9439	0.9420	**0.8223**
Router-neighbour logit distance	15.4716	15.3997	**14.0462**
Prompt sensitivity score	0.2511	0.2513	**0.2472**
Mean downstream score	0.2667	0.2657	0.2669
Perplexity	25.7051	25.5577	25.6976

The direct comparison shows that the tested explicit global-balancing setting without geometry does not reproduce the geometry-associated router-structure pattern. Relative to the baseline, global balancing changes Load coefficient of variation from 0.9439 to 0.9420 and router-neighbour logit distance from 15.4716 to 15.3997. These changes are small. In contrast, geometry regularization lowers Load coefficient of variation to 0.8223 and router-neighbour logit distance to 14.0462. It also increases router entropy and reduces prompt sensitivity.

The second control asks whether geometry still contributes when explicit global balancing is already active. Both configurations in this comparison use global balancing with $\lambda_{\text{gb}} = 0.01$ and router-logit stabilization with $\lambda_z = 5 \times 10^{-4}$. Importance uniformity and sequence guard are disabled. The primary difference is whether the geometric loss is active.

The comparison also exposes costs. Perplexity increases by approximately 1.6%, jitter increases by approximately 3.2% in the relative diagnostic summary, and base-router drift increases by approximately 40.2%. Geometry therefore produces a stronger structural intervention than global balancing without geometry, but it also moves the router further away from the base-router reference.

Table 5.9: Effect of adding geometry to explicit global balancing. Relative changes are computed when the geometric objective is added to the global-balancing setting.

Metric	Global only	Global + geometry	Relative change
Router entropy	3.8334	**3.9192**	+2.24%
Load coefficient of variation	0.9420	**0.8197**	−12.98%
Router-neighbour logit distance	15.3997	**14.0421**	−8.82%
Prompt sensitivity score	0.2513	**0.2450**	−2.52%
Mean downstream score	0.2657	**0.2666**	+0.32%
Perplexity	25.5577	25.9741	+1.63%
Base-router centered L_2	3.3607	4.7111	+40.18%

Global-Control Claim-to-Evidence Conclusion

The global-balancing control supports the second main claim of the chapter: geometry-aware regularization is not reducible to the tested global load-balancing objective. Explicit global balancing without geometry does not substantially reduce router-neighbour logit distance or prompt sensitivity, whereas geometry regularization does. Adding geometry on top of the global-balancing setting still improves Load CV, router entropy, router-neighbour distance, and prompt sensitivity.

The constructive interpretation is that global balancing and geometry regularization act on different aspects of the routing problem. Global balancing addresses aggregate expert utilization, while geometry addresses local consistency of the router map. The two are therefore complementary rather than interchangeable.

5.5 Main Evidence III: What Happens as Geometry Strength Increases?

This section evaluates how the strength of the geometric loss affects router structure, model preservation, prompt sensitivity, jitter, and base-router drift. This directly tests the trade-off described in Section 4.6: increasing λ_{geom} should strengthen local smoothness, but may also increase drift, jitter, or validation cost. Complete sweep tables and supporting figures are provided in Appendix B.

Prompt-Sensitivity and Score Effects

The sweep compares prompt sensitivity and mean downstream score across increasing values of λ_{geom}. Table 5.10 is used to identify broad trends. Per-task ProSA details are provided in Appendix B.

Table 5.10: ProSA summary for the geometric-coefficient sweep using the full controlled configuration. Rows with one run should be interpreted as trend evidence rather than seed-level estimates.

λ_{geom}	Runs	Mean downstream score	Prompt sensitivity score
0	3	0.2661 [0.2602, 0.2726]	0.2518 [0.2462, 0.2572]
50	1	**0.2670** [0.2560, 0.2776]	**0.2486** [0.2387, 0.2580]
75	1	**0.2674** [0.2557, 0.2782]	**0.2480** [0.2387, 0.2577]
100	3	**0.2676** [0.2607, 0.2741]	**0.2484** [0.2424, 0.2544]
125	1	**0.2679** [0.2568, 0.2790]	**0.2466** [0.2374, 0.2560]
150	1	0.2650 [0.2541, 0.2764]	**0.2434** [0.2345, 0.2530]

Prompt sensitivity generally decreases as λ_{geom} increases, moving from 0.2518 at the baseline to 0.2434 at $\lambda_{\text{geom}} = 150$. Mean downstream score does not follow the same monotonic pattern. It increases slightly up to $\lambda_{\text{geom}} = 125$, then decreases at $\lambda_{\text{geom}} = 150$. The important pattern is that prompt sensitivity improves more consistently than raw downstream accuracy.

Router-Structure and Cost Effects

Figure 5.2 visualizes the two structural metrics most directly tied to the geometry objective: Load coefficient of variation and router-neighbour logit distance. Both are reported as signed relative changes from $\lambda_{\text{geom}} = 0$.

The structural sweep shows the most regular trend in the chapter. Stronger geometry produces progressively lower Load coefficient of variation and lower router-neighbour logit distance. This confirms that λ_{geom} behaves as a controllable intervention strength rather than as a binary switch.

Table 5.11 summarizes the remaining trade-off metrics. These metrics are reported in a table rather than in the same plot because base-router drift changes on a much

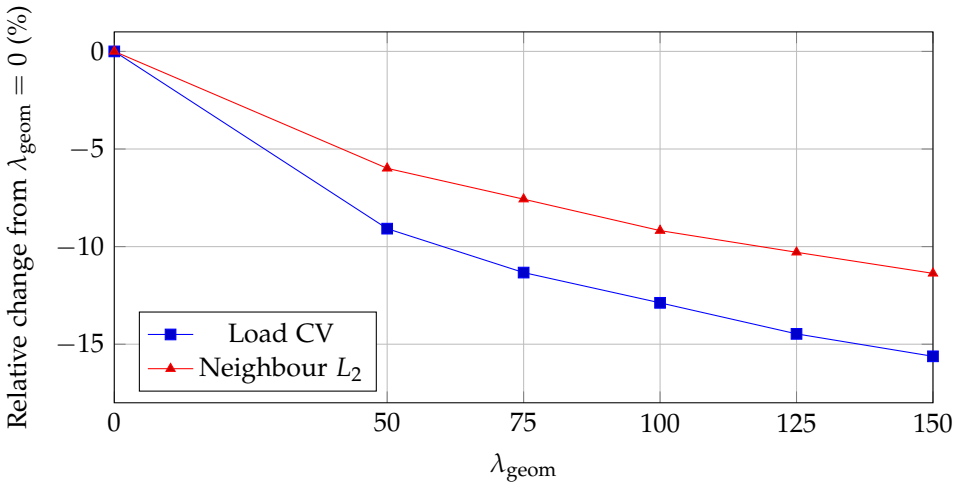


Figure 5.2: Effect of increasing λ_{geom} on the main router-structure diagnostics. Stronger geometry consistently lowers expert-load imbalance and local router-neighbour disagreement.

larger scale than score and prompt sensitivity.

Table 5.11: Trade-off metrics across the full controlled geometric-coefficient sweep. Values are signed relative changes from $\lambda_{\text{geom}} = 0$.

λ_{geom}	Perplexity	Drift L_2	PSS	Mean score
0	0.00%	0.00%	0.00%	0.00%
50	+0.01%	+10.53%	−1.27%	+0.34%
75	+1.69%	+24.51%	−1.51%	+0.49%
100	−0.05%	+39.10%	−1.36%	+0.60%
125	+0.48%	+52.93%	−2.04%	+0.69%
150	+1.63%	+65.68%	−3.32%	−0.39%

The sweep shows that geometry is controllable. Stronger geometry generally produces stronger router-structure effects: Load coefficient of variation decreases, router-neighbour distance decreases, and prompt sensitivity tends to improve. However, stronger geometry also increases base-router drift and eventually increases perplexity. The largest coefficient gives the strongest prompt-sensitivity and structural improvements in this sweep, but it is not the best score-preserving setting.

The practical implication is that geometry strength should not simply be maximized. In this sweep, the safer operating region appears to be moderate geometry strength, especially around $\lambda_{\text{geom}} = 50$ to $\lambda_{\text{geom}} = 100$. In this range, router-structure gains are already substantial while validation quality and mean down-

stream score remain close to the control. Higher coefficients are useful diagnostically, but they are less attractive as default settings because they increase drift and preservation risk.

Complete sweep tables, including drift and additional router diagnostics, are provided in Appendix B.

5.6 Supporting Evidence I: Configuration Selection and Component Roles

The previous sections test the main mechanism. This section answers two related but distinct supporting questions. First, which complete configuration family is most useful under different selection priorities? Second, what role does each supporting component play when removed from a fixed full controlled configuration? Detailed best-configuration evidence is provided in Appendix B.

Table 5.12: Configuration families preferred under different selection priorities.

Selection priority	Most relevant family	Main evidence or interpretation	Caveat
Score preservation	Geometry with router-logit stabilization and importance uniformity	Useful when preserving mean downstream score .	This does not imply that importance uniformity is the main mechanism.
Prompt sensitivity	Geometry with router-logit stabilization and sequence guard	Gives some of the clearest prompt-sensitivity reductions.	Optimized for PSS rather than for every metric.
Structural strength	High-λ_{geom} full controlled geometry	Gives the strongest Load CV and neighbour-distance reductions.	Also increases drift and can weaken preservation.
Controlled interpretability	Moderate-λ_{geom} full controlled geometry	Keeps all supporting components active.	Not necessarily optimal on every metric.

Component Roles

The full controlled geometry configuration contains several supporting components. The component tests evaluate the supporting terms introduced in Chapter 4: confidence weighting in Section 4.4, router-logit stabilization in Section 4.5, importance uniformity in Section 4.5, and sequence guard in Section 4.5.

Table 5.13 summarizes the role of each component. The confidence-weighting ablation is not a mild component removal but a failure-mode test: without confidence weighting, the weighted geometric loss increases from 0.4287 to 26.5227, perplexity increases to 8388.560, and downstream score decreases by 0.0233. This separates it from the other controlled removals, whose downstream score changes are small and have confidence intervals that include zero.

Configuration and Component Claim-to-Evidence Conclusion

The configuration and component evidence supports the fourth claim of the chapter: the practical method is not geometry alone, but controlled geometry. Geometry provides the main structural signal. Confidence weighting makes the signal numerically usable. Router-logit stabilization protects validation quality. The sequence guard contributes most clearly to prompt-sensitivity reduction. Importance uniformity has a small marginal effect in the controlled ablation, but remains practically relevant for score-oriented configuration selection.

The methodological implication is that geometry-aware routing should be evaluated as a controlled router intervention. The supporting terms are not secondary implementation details; they determine whether the geometric signal is usable, stable in scale, and compatible with preservation of the base model.

Complete component-ablation tables are provided in Appendix B.

Table 5.13: Role of each component in the controlled geometry-enabled method.

Component	Ablation or chapter signal	Role in the method	Design implication
Geometric loss	Drives lower Load CV, higher entropy, lower neighbour distance, and lower PSS.	Main router-structure mechanism.	Tune geometry strength rather than treating it as an on/off setting.
Confidence weighting	Removing it increases weighted geometric loss from 0.4287 to 26.5227, raises perplexity to 8388.560, and lowers downstream score by 0.0233.	Required mechanism for loss-scale control.	Treat no-confidence weighting as a failure case, not as a practical candidate.
Router-logit stabilization	Removing it increases perplexity by 0.739 while leaving most router-structure metrics close to the full controlled setting.	Protects language-model quality.	Use it primarily as a preservation component.
Sequence guard	Removing it worsens prompt sensitivity by 0.0037, with the clearest PSS degradation among controlled removals.	Supports prompt-sensitivity reduction.	Include it when prompt robustness is a selection priority.
Importance uniformity	Removing it has small marginal effects in the fixed ablation, but the configuration family that includes it gives the strongest score-oriented result.	Not the main mechanism, but practically relevant for score-oriented selection.	Interpret small marginal ablation effects separately from family-level selection results.

5.7 Supporting Evidence II: Consistency, Sensitivity, and External Validity

The previous sections establish the main router-only result. This section broadens the evidence base. It asks whether the geometry-associated pattern appears across configuration families, available matched seeds, wider trainable scopes, dif-

ferent adaptation datasets, and additional MoE backends. The full sensitivity and external-validity evidence is provided in Appendix C.

Matched-Seed Check

Only families with two matched control-to-geometry seed pairs are included in the available seed check. In each pair, the control configuration has the geometric loss inactive, while the geometry-enabled configuration uses $\lambda_{\text{geom}} = 100$. The matched seeds are 43 and 44.

Instead of listing all seed-level rows in the main text, Table 5.14 summarizes the matched-seed evidence by configuration family. The complete seed-level table is provided in Appendix B.

Table 5.14: Matched-seed summary by configuration family. Values report geometry-enabled minus matched control, averaged over seeds. Negative values indicate reductions; for PSS, Load CV, and neighbour L_2, lower is better.

Configuration family	Seeds	Δ PSS	Δ Load CV	Δ L_2
Geometry with router-logit stabilization	43, 44	−0.0048	−0.1244	−1.359
Full controlled geometry configuration	43, 44	−0.0035	−0.1222	−1.429
Geometry with router-logit stabilization and sequence guard	43, 44	−0.0043	−0.1175	−1.528
Geometry with router-logit stabilization and importance uniformity	43, 44	−0.0037	−0.1228	−1.400

The available matched-seed results suggest that the router-level effects are not limited to the primary seed. In every family and every seed, geometry reduces Load coefficient of variation and router-neighbour L_2. The magnitudes are also similar across families: mean Load CV reduction stays close to 0.12, and mean router-neighbour L_2 reduction stays between approximately 1.36 and 1.53.

Prompt sensitivity also improves in every seed-level comparison. Mean downstream score is less consistent: some comparisons improve slightly, while others decrease slightly.

Trainable-Scope Sensitivity

The main experiments use router-only finetuning because this gives the cleanest attribution: any behavioural change is caused by modifying the router while leaving experts, attention, embeddings, and the language-model head frozen. The trainable-scope sensitivity analysis tests whether the geometry-associated pattern disappears when this boundary is widened. This analysis is useful, but it is less clean methodologically because attention layers, normalization parameters, experts, or full blocks may also move. However, as discussed earlier, router-only post-training can be less adaptive because the routing distribution changes while the rest of the model cannot adjust to that new distribution.

Table 5.15: Trainable-scope sensitivity summary. Values report geometry-enabled minus control. Negative Load CV, neighbour L_2, score, and PSS values indicate decreases; for PSS, lower is better.

Trainable scope	Δ Load CV	Δ neighbour L_2	Δ score	Δ PSS
Router only	-0.1244	-1.352	-0.0005	-0.0024
Router + norms	-0.1137	-1.191	–	-0.0040
Router + experts	-0.1778	-1.759	$+0.0010$	-0.0065
Router + attention + norms	-0.2393	-2.447	-0.0517	–
Full block	-0.2383	-2.553	-0.0675	–

Across all five trainable scopes, geometry reduces Load coefficient of variation and router-neighbour logit distance. The effect therefore does not disappear when the trainable boundary is widened. This is the main structural finding from the sensitivity analysis: the geometry-associated router pattern remains visible even when additional parameter groups are allowed to move.

The router-only setting remains the cleanest result because it preserves attribution to the routing policy. In this setting, geometry reduces Load CV by 0.1244 and neighbour L_2 by 1.352, while mean downstream score changes only by -0.0005 and PSS improves by 0.0024. This supports the main thesis interpretation that geometry can reshape the router while largely preserving downstream behaviour.

It is also notable that trainable router parameters are around 2 million, on the other hand full expert training requires unfreezing nearly all 7B parameters, incurring significant computational overhead.

The cleanest thesis result therefore remains router-only finetuning. The broader trainable-scope sweep supports the robustness of the structural geometry effect,

but it also shows why attribution and preservation must be considered together. Full trainable-scope sensitivity details are provided in Appendix C.

Model and Data Sweep

The model/data sweep tests whether the geometry signal is specific to one adaptation dataset or one MoE backend. The OLMoE data sweep keeps the model family fixed and changes the router-adaptation data. The model sweep keeps the adaptation data fixed and changes the MoE backend. These comparisons are mostly single-seed, so they are interpreted as external-validity evidence rather than seed-level reproducibility evidence.

Table 5.16: OLMoE data sweep summary.

Adaptation data	Perplexity	Δ Load CV	Δ neighbour L_2	Δ score	Δ PSS
UltraChat anchor	26.070 → **25.919**	−0.1198	−1.463	+0.0019	−0.0019
C4	129.672 → **127.319**	−0.1510	−1.878	+0.0025	−0.0007
FineWeb-Edu	118.326 → **115.969**	−0.1658	−2.310	−0.0026	−0.0101
WikiText-103	85.945 → **81.528**	−0.1470	−1.553	+0.0024	−0.0046

Table 5.17: Additional MoE backend sweep summary.

MoE backend	Perplexity	Δ Load CV	Δ neighbour L_2	Δ score	Δ PSS
Granite 1B	3.403 → 3.404	−0.0123	−0.310	+0.0025	+0.0009
Granite 3B	3.102 → **3.098**	+0.0000	−0.068	−0.0016	+0.0002
Phi-3.5 MoE	2.942 → **2.898**	−0.0218	−0.170	–	–
Phi tiny MoE	3.325 → **3.316**	−0.0034	−0.154	−0.0061	−0.0023

The OLMoE data sweep provides the strongest external-validity evidence in this section. Across UltraChat, C4, FineWeb-Edu, and WikiText-103, geometry reduces Load CV and router-neighbour logit distance. This means that the structural effect is not specific to the original UltraChat adaptation data.

The downstream and prompt-sensitivity results are more mixed, but they still support the main interpretation. Mean downstream score improves slightly for UltraChat, C4, and WikiText-103, but decreases slightly for FineWeb-Edu. Prompt sensitivity improves for all four adaptation datasets, with the clearest reductions on FineWeb-Edu and WikiText-103. The geometry effect therefore transfers most reliably through the internal routing diagnostics, while downstream score improvements remain small and dataset-dependent.

Across other MoE model families, the effect is weaker but still structurally visible. Router-neighbour logit distance decreases for Granite and Phi variants, and entropy generally increases in the full results. However, Load CV and ProSA gains are not consistently improved. The Phi-3.5 MoE run is included only for routing and language-modeling diagnostics because ProSA evaluation was not available. The careful conclusion is therefore that geometry-aware regularization generalizes most reliably as a structural router intervention. Downstream score and prompt-sensitivity benefits are more model-dependent. Detailed backend-level results are reported in Appendix C.

5.8 Supporting Evidence III: Computational Feasibility

The geometry objective also has a practical cost. The computational-overhead analysis is based on saved training summaries containing wall-clock time, throughput, peak memory, trainable parameter counts, training steps, and sequence length. Full compute details are provided in Appendix D, including the family-level compute audit in Appendix D and memory-footprint analysis in Appendix D.

Table 5.18: Computational summary for the main local training outputs on a single RTX 6000 Blackwell-series GPU.

Quantity	Value
Complete local training runs	61
Approximate maximum training tokens	312.32M
Total single-GPU runtime	22.07 GPU-hours
Standard OLMoE router-only wall time	18–20 min per run
Standard OLMoE router-only peak memory	≈14.93 GiB
Trainable parameters in router-only runs	2.10M
Typical geometry overhead	6–13% wall-clock
Peak-memory change from geometry, router-only	negligible

Matched control-to-geometry comparisons show that adding the geometry objective introduces moderate runtime overhead. In the main local runs, geometry-aware routing typically adds approximately 6–13% wall-clock overhead, while leaving peak memory essentially unchanged for the standard OLMoE router-only setup. The dominant cost is the local-neighbour construction and router-neighbour divergence computation, not the value of λ_{geom} or the smaller auxiliary terms.

This cost profile supports the practical interpretation of the method. The intervention modifies only a small router parameter subset, adds moderate runtime overhead, and does not substantially increase peak memory in the tested router-

only OLMoE setup. The method is therefore feasible as a post-training router intervention in the tested regime.

The broader sweeps are more expensive. The model/data sweep has a higher mean peak memory because it includes larger MoE backends, and the trainable-scope sweep has a higher mean peak memory because broader parameter groups are trainable.

Computational Claim-to-Evidence Conclusion

The computational evidence supports the sixth claim of the chapter: geometry-aware routing is practical in the router-only regime. The method introduces a measurable but manageable runtime overhead, while peak memory remains essentially unchanged for the standard OLMoE router-only setup. This means that the router can be actively reshaped without requiring full-model finetuning or a large memory increase in the tested implementation.

5.9 Integrated Testing Summary

This section summarizes the supported and limited claims from the testing chapter. The distinction is important because the results support a strong router-structure claim, a modest prompt-sensitivity claim, and a careful preservation claim, but not a broad raw-accuracy improvement claim. The detailed evidence supporting these claims is organized across Appendix B, Appendix C, and Appendix D.

Table 5.19 gives the final claim status. The purpose of the table is to separate what the experiments establish from what they do not establish.

The testing chapter supports the central thesis claim that local hidden-state geometry provides a meaningful structural regularization signal for sparse MoE routers. Enabling geometry reduces Load coefficient of variation, increases router entropy, lowers router-neighbour logit distance, and modestly reduces prompt sensitivity. These effects appear in the primary comparison, remain visible when the tested global-balancing setting is already active, strengthen across the coefficient sweep, and are visible across the available matched seeds.

The broader scope and model/data sweeps strengthen the structural interpretation while clarifying its limits. Across trainable scopes, adaptation datasets, and several MoE backends, geometry-aware training reduces local router-neighbour logit distance and usually improves expert-load balance. However, wider trainable scopes

Table 5.19: Supported and limited claims from the testing chapter.

Claim	Status	Evidence or limitation
Geometry changes router structure	Supported	Load CV decreases, router entropy increases, and router-neighbour logit distance decreases in the primary test and across configuration families.
Geometry improves prompt sensitivity	Supported, modestly	PSS decreases in the primary test and improves in all available matched-seed comparisons.
Geometry preserves ordinary model quality at moderate strength	Supported cautiously	Validation loss, perplexity, and mean downstream score remain close to the baseline in the primary comparison and moderate sweep settings.
Geometry is more than global load balancing	Supported	Geometry still improves Load CV, neighbour distance, and PSS when global load balancing is already active.
Geometry generalizes as a structural router regularizer	Supported cautiously	Across trainable scopes, adaptation datasets, and several MoE backends, geometry reduces router-neighbour distance and usually improves load balance.
Geometry clearly improves raw downstream accuracy	Not supported	Mean downstream score changes are small, mixed, and often uncertain.
Geometry improves every sensitivity-related diagnostic	Not supported	Jitter increases even when local router smoothness improves.
Geometry gives model-independent downstream gains	Not supported	Downstream score and PSS improvements are not consistent across all MoE backends.
Broader trainable scopes are automatically better	Not supported	Router + attention + norms and full-block training show large score drops despite strong structural changes.

can confound prompt-sensitivity reductions with task-score degradation, and non-OLMoE backends show weaker or more model-dependent downstream effects.

The chapter also identifies the main costs. Geometry-aware regularization improves local router structure and expert-load balance, but it increases jitter and base-router drift. It also introduces a modest computational overhead because the neighbourhood graph and router-neighbour divergence must be computed during training. In the standard OLMoE router-only setting, this overhead is manageable: runs complete in approximately 18–20 minutes on one RTX 6000 Blackwell-series

GPU, peak memory remains close to 14.93 GiB, and geometry typically adds approximately 6–13% wall-clock time.

Overall, the experiments support geometry-aware routing as a practical router-behaviour regularization method. It reliably reshapes local routing structure and can reduce prompt sensitivity, but it should be treated as a controlled structural intervention rather than as a general raw-accuracy improvement method. The following discussion chapter interprets what these findings mean for the thesis argument, including why router-structure improvements do not automatically translate into downstream accuracy gains, how prompt sensitivity should be interpreted relative to jitter and drift, and what the broader sensitivity, external-validity, computational, and pretraining evidence imply for future work.

Chapter 6

Discussion

This chapter interprets the empirical evidence from Chapter 5. Its purpose is to explain what the results mean for sparse Mixture-of-Experts (MoE) routing, not to repeat the testing chapter. The central claim is deliberately scoped: Geometry-Aware Router Post-training (GARP) reshapes router behaviour in the router-only regime, but it is not a general downstream-accuracy method.

Sparse MoE models rely on a router to map token representations to active experts. Existing work has therefore emphasized utilization, stabilization, and efficient sparse dispatch [4, 5, 6, 7]. GARP adds a local question: can nearby hidden states produce more compatible router outputs while the pretrained experts, backbone, and standard Top-K inference path remain unchanged?

6.1 Interpretation of the Main Result

The primary comparison answers that question in the moderate-strength setting. Only router parameters are updated, while the experts, attention layers, embeddings, and language-model head remain fixed. The relevant outcome is therefore not new model capability, but whether the router map can be reorganized without damaging the pretrained model.

The strongest changes in Table 6.1 are internal to the router: expert-load imbalance and neighbour disagreement decrease, while perplexity and mean downstream score remain nearly unchanged. This supports the interpretation of GARP as a router-control mechanism rather than a source of new task capability.

Table 6.1: Numerical evidence behind the main discussion claim. Bold values improve relative to the baseline; jitter is left unbolded because it is a cost.

Metric	Baseline	GARP	Change
Router entropy	3.8327	**3.9193**	+0.0866
Load coefficient of variation	0.9439	**0.8223**	−0.1216
Router-neighbour logit distance	15.4716	**14.0462**	−1.4254
Prompt sensitivity score	0.2511	**0.2472**	−0.0040
Perplexity	25.7051	**25.6976**	−0.0074
Mean downstream score	0.2667	**0.2669**	+0.0002
Jitter	0.6424	0.6635	+0.0211

6.2 Geometry Beyond Load Balancing

Load balancing and local geometry act on different properties of the router. Load balancing regulates marginal expert traffic; the geometry term constrains how the router varies over hidden-state neighbourhoods. A router can satisfy the first condition while remaining locally irregular, which is why the global-balancing control is central to the thesis argument.

Table 6.2: Numerical evidence that the geometry signal is not equivalent to the tested global-balancing objective. Bold values improve relative to the baseline.

Metric	Baseline	Global only	Geometry only
Router entropy	3.8327	**3.8334**	**3.9193**
Load coefficient of variation	0.9439	**0.9420**	**0.8223**
Router-neighbour logit distance	15.4716	**15.3997**	**14.0462**
Prompt sensitivity score	0.2511	0.2513	**0.2472**
Mean downstream score	0.2667	0.2657	**0.2669**
Perplexity	25.7051	**25.5577**	**25.6976**

Table 6.2 shows that global balancing alone does not reproduce the geometry-associated pattern. Geometry gives the larger reduction in router-neighbour distance and the clearer PSS reduction, so it complements rather than replaces existing MoE routing controls.

This aligns with routing work that treats similarity, token structure, or representation geometry as useful signals beyond load counts [10, 12, 11]. GARP contributes the narrower router-only post-training case.

6.3 Stability, Sensitivity, and Drift

The stability evidence is mixed rather than uniformly positive. Router-neighbour distance evaluates smoothness over the hidden-state graph, prompt sensitivity measures variation across prompt formulations, jitter measures response to a separate perturbation diagnostic, and drift measures movement away from the original routing policy. Improving one of these quantities does not guarantee improvement in the others.

This matters because GARP can make neighbouring hidden states more consistent while also moving the router farther from its baseline decision boundaries. The method should therefore be described as improving local router coherence and modestly reducing prompt sensitivity in the tested setting, not as a complete stability solution.

6.4 Practical Operating Region

The geometric coefficient behaves as a control knob. The useful region is not simply the largest tested coefficient, but the range where router organization improves before drift and preservation costs dominate.

Table 6.3: Numerical evidence for treating geometry strength as a trade-off parameter. Values are relative changes from $\lambda_{\text{geom}} = 0$; bold values are improvements, while drift increases are left unbolded.

λ_{geom}	Load CV	Neighbour L_2	PSS	Perplexity	Drift L_2
0	0.00%	0.00%	0.00%	0.00%	0.00%
50	−9.08%	−5.99%	−1.27%	+0.01%	+10.53%
75	−11.33%	−7.57%	−1.51%	+1.69%	+24.51%
100	−12.88%	−9.18%	−1.36%	−0.05%	+39.10%
125	−14.47%	−10.29%	−2.04%	+0.48%	+52.93%
150	−15.62%	−11.37%	−3.32%	+1.63%	+65.68%

Table 6.3 makes this trade-off explicit: larger coefficients strengthen the structural effect, but they also move the router farther from its starting policy.

Two practical implications follow. First, geometry-aware router training should be tuned as a regularizer rather than maximized as an independent objective. Second, the supporting components are part of the method rather than incidental details: confidence weighting limits unreliable neighbourhood edges, router-logit

stabilization controls logit scale, and sequence-level safeguards help avoid undesirable prompt-level effects.

6.5 External Validity and Limits

The broader experiments separate what generalizes reliably from what remains setting-dependent. Matched seeds, trainable-scope variants, and data/model sweeps most consistently support the router-structure effect; behavioural gains are less uniform.

The matched-seed checks provide the clearest repeatability evidence for the structural effect. Across matched runs, Load CV improves by **-0.1175 to -0.1244** and neighbour L_2 improves by **-1.359 to -1.528**. PSS also improves by **-0.0035 to -0.0048**, but mean downstream score is mixed, which keeps the behavioural claim narrower than the router-structure claim.

The trainable-scope variants support the same distinction. All five scopes reduce Load CV and neighbour L_2, with the router-only setting giving **-0.1244** Load CV and **-1.352** neighbour L_2. Wider scopes require more caution, because they can reduce mean score, including -0.0517 and -0.0675 in the broadest settings.

The OLMoE data sweep also supports structural consistency across adaptation data. All four datasets reduce Load CV by **-0.1198 to -0.1658** and neighbour L_2 by **-1.463 to -2.310**. PSS improves by **-0.0007 to -0.0101**, while score changes remain small and mixed.

The backend sweep is the least uniform part of the external-validity evidence. Neighbour L_2 improves for all listed backends, from **-0.068 to -0.310**. However, Load CV, score, and PSS are model-dependent; Granite PSS increases by +0.0009 and +0.0002. This is why the claim should remain scoped: downstream performance depends on expert specialization, task format, prompt choice, dataset coverage, and whether routing consistency matters for the task.

The practical evidence is favourable within the tested regime. Router-only post-training updates a small parameter subset, uses a short training horizon, introduces modest training overhead, and does not alter inference-time dispatch. These properties make GARP feasible as a low-cost post-training intervention. However, the current experiments do not establish how the method behaves under longer adaptation, larger seed sets, larger MoE models, capacity-constrained serving, or pretraining-time integration.

6.6 Implications for Future Work

Future work should test where local routing coherence matters most. Larger multi-seed studies are needed to quantify uncertainty in prompt sensitivity and downstream scores. Broader task suites should examine settings in which routing consistency is likely to be important, including long-context tasks, multi-step reasoning, multilingual inputs, and domain-shifted evaluation. Systems-oriented experiments should connect router-structure changes to capacity overflow, token dropping, communication imbalance, expert-cache behaviour, and serving latency [49, 30, 18, 19].

Longer adaptation and pretraining-time experiments are also necessary. Router-only post-training can reorganize which existing expert computations are selected, but it cannot show whether geometry-aware routing shapes expert specialization while the experts themselves are learning. That question connects GARP to the broader design of sparse MoE pretraining objectives.

Overall, local hidden-state geometry provides a useful signal for shaping sparse MoE routers, with its clearest value in local routing coherence and expert-load organization.

Chapter 7

Conclusion

This thesis examined whether local hidden-state geometry can be used to improve sparse Mixture-of-Experts (MoE) routing while preserving the standard sparse router and its inference-time dispatch. Geometry-Aware Router Post-training (GARP) adds a local structural signal to router-only training: nearby hidden states are encouraged to receive more compatible router distributions. This matters because aggregate expert use can look healthy even when the router map is locally irregular. The thesis therefore turns local routing coherence into an explicit training objective, rather than treating it only as a diagnostic after training.

In the primary router-only comparison, GARP reduced Load coefficient of variation by 12.9%, reduced router-neighbour logit distance by 9.2%, and reduced Prompt Sensitivity Score by 1.6%. Mean downstream score was essentially preserved, changing by only +0.1%, and perplexity stayed effectively unchanged, changing by less than 0.1%. These results support the central finding of the thesis: a sparse MoE router can be made more locally organized while preserving ordinary model quality and leaving the inference path unchanged. The trade-off is also clear. Jitter increased by 3.3%, and router drift can increase as the geometry coefficient becomes stronger. GARP is therefore best understood as a tunable structural prior for the router, with the geometry coefficient setting an operating point between local coherence, global usage, and assignment stability.

The broader evidence strengthens this interpretation. The global-balancing controls show that local geometry contributes information beyond expert-count objectives, and the sensitivity experiments show where the method is most reliable: router-only post-training, moderate geometry strength, and evaluation regimes that measure both model quality and router structure. In that operating region, GARP improves the organization of expert assignment while keeping the practical

advantage of standard Top-K sparse dispatch.

The contribution of this thesis is therefore both methodological and empirical. Methodologically, it presents a lightweight router-only objective that connects hidden-state neighbourhoods to expert assignment behaviour. Empirically, it shows that local hidden-state geometry is an actionable signal for improving routing coherence and expert-load organization while preserving the deployed MoE architecture. For sparse MoE systems, the main lesson is that routing quality depends on both how much traffic each expert receives and whether nearby representations are routed in a coherent and stable way.

Appendix A

Appendix Guide

The source code, experiment scripts, configuration files, and supporting implementation materials for this thesis are available in the accompanying GitHub repository: `https://github.com/Ka-rlis/Geometry-Aware-Routing-Regularization`. This repository is provided to support transparency and reproducibility by making the practical implementation of the proposed method accessible alongside the quantitative results reported in this thesis.

The appendices provide the extended quantitative evidence behind Chapter 5. The main Testing chapter reports the central results needed to support the thesis argument, while the appendices preserve the full experimental record: additional metric tables, per-task results, matched-seed comparisons, trainable-scope sensitivity results, model/data sweeps, computational measurements, and the separate pretraining pilot.

The appendix material should therefore be read as an evidence archive rather than as a second testing chapter. The main interpretation remains in Chapter 5 and Chapter 6. The purpose of the appendices is to make the experimental claims traceable, reproducible at the level of reported outputs, and easier to inspect in detail.

Organization

The appendix is organized to follow the evidence structure of the Testing chapter. Table A.1 summarizes the role of each appendix chapter.

Table A.1: Appendix chapter map.

Appendix chapter	Purpose
Appendix A: Appendix Guide	Defines metric abbreviations, reporting conventions, missing-evaluation notation, and the mapping between Testing chapter claims and appendix evidence.
Appendix B: Extended Results	Reports the full router-only OLMoE post-training evidence behind the main Testing chapter, including primary geometry comparisons, coefficient sweeps, ablations, matched seeds, per-task ProSA results, and per-layer diagnostics.
Appendix C: Sensitivity and External Validity	Reports the trainable-scope sensitivity analysis and the model/data sweep. These results are used as breadth evidence rather than as the primary causal evidence.
Appendix D: Computational Cost	Reports runtime, throughput, memory, trainable parameter counts, and geometry-overhead estimates for the router-only post-training runs.
Appendix E: Pretraining Pilot	Documents a separate pretraining-from-scratch pilot. This is treated as supporting evidence and future-work motivation, not as part of the main router-only post-training claim.

Metrics

Table A.2 defines the main metrics used across the appendices. The desired direction is included to make the result tables easier to read.

The term *stability* is intentionally avoided as a single aggregate metric. Local router-neighbour smoothness, prompt sensitivity, jitter, and base-router drift are separate diagnostics. They can move in different directions and should not be collapsed into one claim.

Table A.2: Metric definitions.

Metric	Definition and interpretation	Desired direction
Loss	Cross-entropy loss on the held-out evaluation split. In the main post-training experiments this is the held-out UltraChat split unless otherwise stated.	Lower.
PPL	Perplexity, computed as exp(Loss). It is reported as a model-quality metric.	Lower.
Jitter	Fraction of adjacent-token pairs in a sequence that route to different top-1 experts, averaged over MoE layers. It measures adjacent-token routing variation and is interpreted separately from local router-neighbour smoothness and prompt sensitivity.	Lower.
Entropy	Mean routing entropy across MoE layers. Higher entropy indicates a less concentrated router distribution, but it must be interpreted together with load balance and model quality.	Higher, if quality is preserved.
Load CV	Coefficient of variation of expert token counts, averaged over MoE layers. It measures relative imbalance in expert usage.	Lower.
Load variance	Variance of per-expert token counts. It is another aggregate expert-load diagnostic.	Lower.
Router-neighbour logit distance	Mean distance between router logits of neighbouring hidden-state representations. This is the main evaluation diagnostic for local router-neighbour smoothness.	Lower.
Router L2 / drift L2	Mean L2 distance between the trained router parameters and the pretrained router reference. This measures how far the router moved from the anchor.	Lower, when preservation is desired.
Anchor KL	KL divergence between the trained router distribution and the pretrained anchor-router distribution.	Lower, when preservation is desired.
PSS	Prompt Sensitivity Score. It measures variation in task behaviour across prompt variants for the same instance.	Lower.
Avg score	Mean downstream score across the evaluated downstream tasks.	Higher.
ARC	ARC-Challenge accuracy.	Higher.
CSQA	CommonsenseQA accuracy.	Higher.
HE	HumanEval pass@1.	Higher.
Seed	Random seed used for training or evaluation setup.	Not applicable.

Table A.3: Main design choices in the proposed method.

Component	Choice	Reason
Trainable boundary	Router gates only	Keeps post-training lightweight and isolates changes to routing behaviour.
Hidden-state graph	Per-layer kNN over detached pre-gate states, built independently within each batch row	Uses the representation geometry seen by the router while avoiding edges across unrelated batch rows.
Similarity metric	Cosine similarity after L_2 normalization	Focuses on direction in hidden space and reduces sensitivity to hidden-state magnitude.
Candidate window	Optional positional window within each batch row	Reduces accidental distant edges while still selecting neighbours by hidden-state similarity rather than adjacency alone.
Router-state representation	Dispatch-aligned router distribution	Aligns the geometry loss with the expert weights used by sparse dispatch, while falling back to the full router softmax when sparse dispatch tensors are unavailable.
Router disagreement	Symmetric KL divergence	Penalizes probability-mass disagreement and expert-support mismatch between neighbouring router states.
Confidence weighting	Entropy-based edge reliability	Down-weights uncertain router states so ambiguous neighbourhoods do not dominate the geometry objective.
Global load balancing	Optional Switch-style global-batch selected-expert balancing	Tests whether explicit aggregate balancing provides additional protection against collapse without forcing per-sequence uniformity.
Router z-loss	Optional squared router log-normalizer penalty	Stabilizes router logit scale during post-training.
Importance uniformity	Optional soft probability-mass uniformity	Tests whether controlling average router mass improves stability when combined with geometry regularization.
Sequence guard	Optional thresholded sequence-level concentration penalty	Prevents one expert from dominating too much of an individual sequence without forcing every sequence to use all experts uniformly.

Reporting

The appendices use the following reporting conventions.

- For lower-is-better metrics such as perplexity, Load CV, router-neighbour distance, PSS, jitter, and drift, negative changes indicate improvement.
- For higher-is-better metrics such as routing entropy and downstream score, positive changes indicate improvement.
- Confidence intervals are reported where raw evaluation samples are available.
- For single-run comparisons, confidence intervals reflect evaluation-sample uncertainty, not full seed-to-seed uncertainty.
- Internal run identifiers are preserved in raw appendix tables when needed for traceability, but the main Testing chapter uses readable configuration names.

Table A.4 summarizes how signs should be interpreted.

Table A.4: Signed-change conventions.

Metric group	Favourable sign	Interpretation
Perplexity, loss	Negative	The geometry-enabled run has better quality.
Load CV, load variance	Negative	Expert usage is more balanced.
Router-neighbour distance	Negative	Local router-neighbour disagreement is reduced.
PSS	Negative	Prompt-induced variation reduced.
Jitter	Negative	Adjacent-token routing variation is reduced.
Router drift / anchor KL	Negative	The router stays closer to the pretrained anchor.
Router entropy	Positive	The router distribution is less concentrated.
Downstream score	Positive	Task performance improves.

Table A.5: Configuration names.

Readable name	Meaning
Baseline	Reference setting with the geometric loss inactive.
Geometry only	Geometry-enabled router finetuning with confidence weighting.
Geometry with router-logit stabilization	Geometry with confidence weighting and router-logit stabilization.
Geometry with router-logit stabilization and importance uniformity	Geometry with confidence weighting, router-logit stabilization, and importance uniformity. This is the score-oriented configuration family.
Geometry with router-logit stabilization and sequence guard	Geometry with confidence weighting, router-logit stabilization, and sequence guard. This is the prompt-sensitivity-oriented configuration family.
Full controlled geometry configuration	Geometry with confidence weighting, router-logit stabilization, importance uniformity, and sequence guard. This is the main controlled ablation setting.
Geometry without confidence weighting	Full controlled geometry configuration with confidence weighting removed. This is interpreted as a loss-scale ablation.
Global load balancing only	Explicit global load balancing and router-logit stabilization, but no geometry.
Global load balancing with geometry	Explicit global load balancing with the geometric objective active.

Configurations

The main Testing chapter avoids internal run identifiers where possible. The appendices sometimes retain these identifiers to preserve traceability. Table A.5 gives the readable names used in the main text.

Appendix B

Extended Testing Results

This appendix chapter reports the extended evidence behind the main router-only OLMoE testing results in Chapter 5. It contains the detailed quantitative support for the primary geometry comparison, the global load-balancing control, the geometric-coefficient sweep, the ProSA task-level results, matched-seed comparisons, component ablations, per-layer diagnostics, and exploratory early experiments.

Unless otherwise stated, the experiments in this appendix use `allenai/OLMoE-1B-7B-0125-Instruct` with router-only post-training on `Felladrin/ChatML-ultrachat_200k`. Experts, attention layers, embeddings, and the language-model head remain frozen. The results are therefore interpreted as router-only post-training evidence. Sensitivity checks that change the trainable scope, adaptation data, or model backend are reported separately in Appendix C.

The purpose of this appendix is traceability. The main Testing chapter presents the compact argument and the central conclusions; this appendix preserves the more detailed evidence needed to audit those conclusions.

Pretrained Baseline

Table B.1 reports the pretrained OLMoE checkpoint before router post-training. This baseline is not directly comparable to every post-training row because later evaluations use specific held-out splits and downstream evaluation artifacts. It is included to document the starting point of the experimental campaign.

The pretrained checkpoint provides the initial router state from which the router-

Table B.1: Pretrained baseline.

Model	Loss	PPL	Jitter	Entropy	Load CV
OLMoE-1B-7B-0125-Instruct	1.8390	6.2904	0.6185	3.7133	0.8542

Table B.2: Geometry activation check.

Diagnostic	Baseline	Geometry enabled
λ_{geom}	0	100
Matched comparisons	9	9
Weighted geometric loss	0.0000 [0.0000, 0.0000]	0.4286 [0.4258, 0.4313]
Active neighbours	0.0000 [0.0000, 0.0000]	12.0000 [12.0000, 12.0000]
Active token fraction	0.0000 [0.0000, 0.0000]	1.0000 [1.0000, 1.0000]

only post-training runs begin. Later tables focus on matched post-training comparisons, because those comparisons share training data, training length, and evaluation setup.

Primary Geometry

The primary comparison evaluates the marginal effect of enabling the geometric loss. The baseline has $\lambda_{\text{geom}} = 0$, while the geometry-enabled setting uses $\lambda_{\text{geom}} = 100$. This is the core comparison behind Section 5.3.

Activation

Table B.2 verifies that the geometric objective is inactive in the baseline and active in the geometry-enabled setting.

The control rows have zero weighted geometry loss and zero active geometry neighbours. Geometry-enabled rows use all $k = 12$ neighbours on average and have a non-zero weighted geometry loss. This confirms that the geometric objective was active only in the intended setting.

Table B.3: Primary router-structure metrics.

Metric	Baseline	Geometry enabled	Change
Router entropy	3.8327 [3.8325, 3.8329]	3.9193 [3.9191, 3.9194]	+0.0866 [0.0863, 0.0869]
Load coefficient of variation	0.9439 [0.9426, 0.9451]	0.8223 [0.8212, 0.8232]	−0.1216 [-0.1236, -0.1196]
Router-neighbour logit distance	15.4716 [15.4449, 15.5007]	14.0462 [14.0308, 14.0604]	−1.4254 [-1.4688, -1.3859]

Router Structure

Table B.3 reports the main router-structure diagnostics. These are the most direct tests of whether the geometric objective reshapes the router map.

The router-structure diagnostics show the intended pattern: router entropy increases, Load CV decreases, and router-neighbour logit distance decreases. The reduction in router-neighbour distance is the most direct evidence that neighbouring hidden-state representations receive more similar router logits after geometry-aware training.

Quality and ProSA

Table B.4 reports model-quality and downstream metrics for the same comparison.

The quality metrics show preservation rather than clear improvement. Validation loss, perplexity, and mean downstream score remain close to the baseline. Prompt sensitivity decreases, while jitter increases. This supports the interpretation used in Chapter 5: geometry improves local router structure and modestly reduces prompt sensitivity, but it does not improve every sensitivity-related diagnostic and does not reliably improve raw downstream accuracy.

Table B.4: Primary quality and downstream metrics.

Metric	Baseline	Geometry enabled	Change
Validation loss	3.2466 [3.2401, 3.2533]	3.2464 [3.2404, 3.2523]	−0.0003 [-0.0096, 0.0097]
Perplexity	25.7051 [25.5371, 25.8770]	25.6976 [25.5456, 25.8491]	−0.0074 [-0.2459, 0.2476]
Jitter	0.6424 [0.6418, 0.6429]	0.6635 [0.6628, 0.6640]	+0.0211 [0.0202, 0.0219]
Prompt sensitivity score	0.2511 [0.2505, 0.2518]	0.2472 [0.2464, 0.2479]	−0.0040 [-0.0047, -0.0033]
Mean downstream score	0.2667 [0.2661, 0.2672]	0.2669 [0.2663, 0.2675]	+0.0002 [-0.0007, 0.0010]
ARC-Challenge score	0.4002 [0.3988, 0.4014]	0.3969 [0.3957, 0.3982]	−0.0032 [-0.0051, -0.0013]
CommonsenseQA score	0.3771 [0.3758, 0.3783]	0.3792 [0.3783, 0.3801]	+0.0021 [0.0010, 0.0033]
HumanEval score	0.0227 [0.0215, 0.0238]	0.0244 [0.0236, 0.0252]	+0.0017 [0.0008, 0.0026]

Global-Balance Control

The global load-balancing control tests whether the geometry effect can be explained by ordinary global expert-load regularization. The comparison adds geometry on top of explicit global load balancing.

Table B.5: Global-balance control.

Metric	Global balance only	Global balance + geometry	Relative change
Router entropy	3.8334	3.9192	+2.24%
Load coefficient of variation	0.9420	0.8197	−12.98%
Router-neighbour logit distance	15.3997	14.0421	−8.82%
Prompt sensitivity score	0.2513	0.2450	−2.52%
Mean downstream score	0.2657	0.2666	+0.32%
Perplexity	25.5577	25.9741	+1.63%
Base-router centered L_2	3.3607	4.7111	+40.18%

Table B.6 reports the benchmark-level ProSA results for the same comparison.

The paired ProSA test is summarized in Table B.7. The score change is small and statistically unclear, while the PSS gain is supported by the interval and p-value.

Table B.6: Global-balance ProSA results.

Metric	Global balance only	Global balance + geometry	Change
Mean downstream score	0.2657	0.2666	+0.0009
Prompt sensitivity score	0.2513	0.2450	−0.0063
ARC-Challenge score	0.3960	0.3973	+0.0013
ARC-Challenge PSS	0.3066	0.2951	−0.0115
CommonsenseQA score	0.3788	0.3790	+0.0003
CommonsenseQA PSS	0.4041	0.3946	−0.0095
HumanEval score	0.0224	0.0234	+0.0010
HumanEval PSS	0.0432	0.0453	+0.0020

Table B.7: Global-balance paired ProSA test.

Metric	Change / gain	95% CI	p
Mean score change	+0.0009	$[-0.0031, +0.0047]$	0.6764
PSS gain	+0.0063	$[+0.0008, +0.0115]$	0.0188

The control supports the conclusion that geometry is not redundant with global load balancing. Global balance controls aggregate expert usage. Geometry adds a local hidden-state-neighbourhood constraint that changes router-neighbour behaviour and prompt sensitivity.

Geometry-Weight Sweep

The geometric-coefficient sweep evaluates how stronger geometry affects router structure, prompt sensitivity, validation quality, downstream score, and base-router drift. The main Testing chapter reports the compact version of this sweep. This appendix records the detailed sweep summary.

ProSA Summary

Table B.8: Geometry-weight sweep: ProSA summary.

λ_{geom}	Runs	Mean downstream score	Prompt sensitivity score
0	3	0.2661 [0.2602, 0.2726]	0.2518 [0.2462, 0.2572]
50	1	0.2670 [0.2560, 0.2776]	0.2486 [0.2387, 0.2580]
75	1	0.2674 [0.2557, 0.2782]	0.2480 [0.2387, 0.2577]
100	3	0.2676 [0.2607, 0.2741]	0.2484 [0.2424, 0.2544]
125	1	0.2679 [0.2568, 0.2790]	0.2466 [0.2374, 0.2560]
150	1	0.2650 [0.2541, 0.2764]	0.2434 [0.2345, 0.2530]

Table B.9: Geometry-weight sweep: signed relative changes.

λ_{geom}	PPL	Load CV	Neighbour L_2	Drift L_2	PSS	Mean score
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
50	+0.01%	−9.08%	−5.99%	+10.53%	−1.27%	+0.34%
75	+1.69%	−11.33%	−7.57%	+24.51%	−1.51%	+0.49%
100	−0.05%	−12.88%	−9.18%	+39.10%	−1.36%	+0.60%
125	+0.48%	−14.47%	−10.29%	+52.93%	−2.04%	+0.69%
150	+1.63%	−15.62%	−11.37%	+65.68%	−3.32%	−0.39%

Prompt sensitivity generally decreases as the geometry coefficient increases. Mean downstream score remains close to the baseline and does not show the same monotonic pattern.

Relative Changes

Table B.9 reports signed relative changes from the $\lambda_{\text{geom}} = 0$ full controlled setting. Negative values are favourable for lower-is-better metrics such as Load CV, neighbour L_2, PSS, and perplexity. Positive values are favourable for mean score, while positive drift is a cost.

The sweep shows a clear structural trend: stronger geometry reduces Load CV and router-neighbour logit distance. It also increases base-router drift. This confirms that the router is being actively reshaped rather than passively regularized.

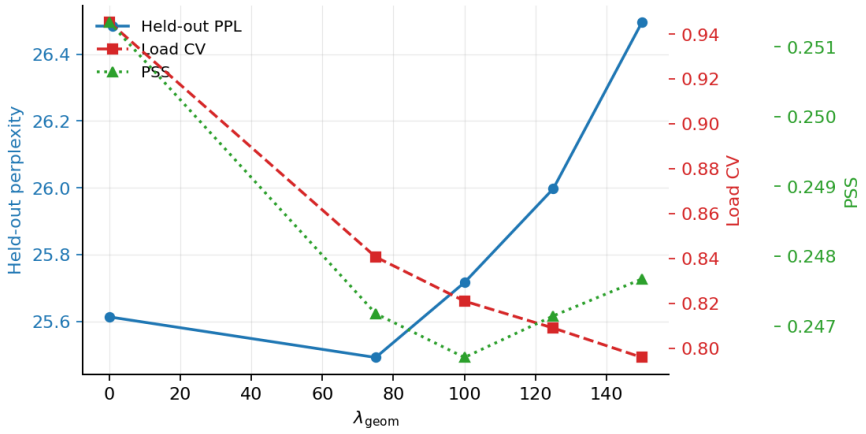


Figure B.1: Dense lambda-sweep trade-off. Larger geometry weights improve load balance and local router-neighbour smoothness, but may increase language-modeling cost or router drift.

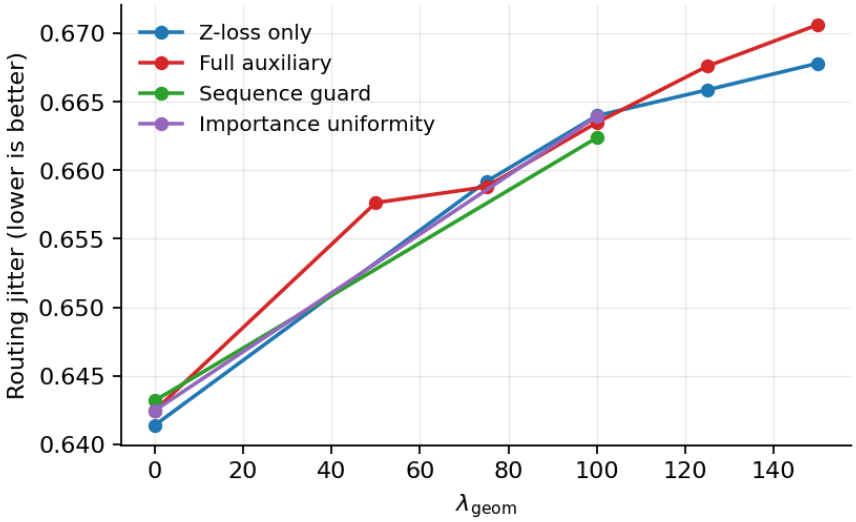


Figure B.2: Routing jitter across geometry weights. Lower jitter is better under the reported metric; the geometry-enabled variants do not consistently improve this diagnostic.

Sweep Figures

The figures below visually confirm the trends reported in the tables.

The operating-point interpretation is that $\lambda_{\text{geom}} = 100$ is a stable and interpretable comparison point. It is not claimed to be universally optimal. It gives a useful balance between router-structure improvement, prompt-sensitivity reduction, validation-quality preservation, and controlled drift.

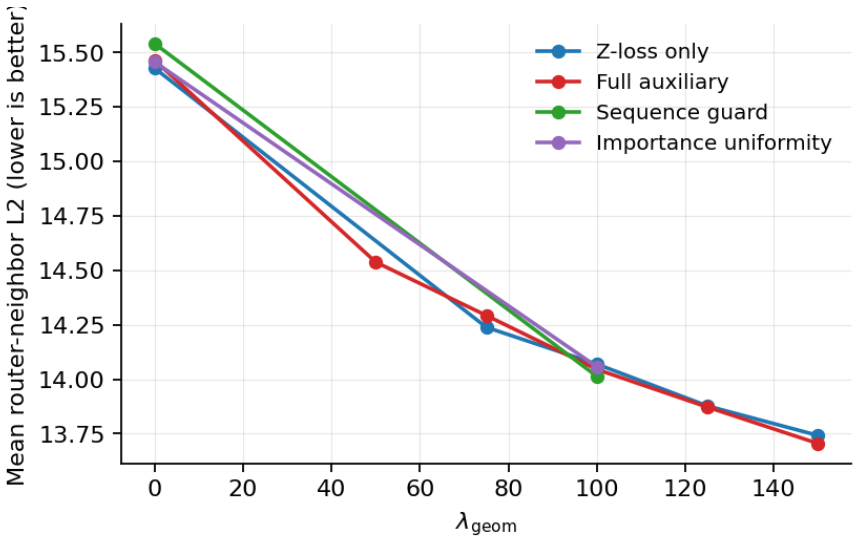


Figure B.3: Router-neighbour L_2 across matched candidates. Lower values indicate reduced local router-logit variation among hidden-state neighbours.

Best Families

Table B.10 summarizes which configuration families are most relevant under different selection criteria.

Table B.10: Configuration-family interpretation.

Selection criterion	Relevant family	Interpretation
Score-oriented result	Geometry with router-logit stabilization and importance uniformity	This family provides the strongest score-oriented configuration among the evaluated candidates.
Prompt-sensitivity-oriented result	Geometry with router-logit stabilization and sequence guard	This family provides some of the clearest prompt-sensitivity reductions.
Strongest router-structure effect	Larger geometry coefficients in the full controlled geometry configuration	Higher geometry weights reduce Load CV and router-neighbour distance more strongly
Most interpretable controlled setting	Full controlled geometry configuration at a moderate geometry coefficient	This setting includes all control components and is therefore the clearest controlled reference for ablation.

This distinction is important because an ablation and a configuration-family comparison answer different questions. The ablation asks what happens when one component is removed from a fixed full setting. The family comparison asks which complete recipe gives the most useful empirical result under a selection criterion.

Per-Task ProSA

This section reports detailed ProSA values for the full controlled geometry configuration. The main Testing chapter reports aggregate values; this appendix records the benchmark-level values.

Table B.11: Full controlled configuration: task-level ProSA.

Metric	$\lambda_{\text{geom}} = 0$	$\lambda_{\text{geom}} = 100$
Mean downstream score	0.2661 [0.2602, 0.2726]	0.2676 [0.2607, 0.2741]
Prompt sensitivity score	0.2518 [0.2462, 0.2572]	0.2484 [0.2424, 0.2544]
ARC-Challenge score	0.3984 [0.3833, 0.4132]	0.3986 [0.3839, 0.4143]
ARC-Challenge PSS	0.3047 [0.2953, 0.3141]	0.3000 [0.2902, 0.3087]
CommonsenseQA score	0.3750 [0.3649, 0.3858]	0.3793 [0.3690, 0.3896]
CommonsenseQA PSS	0.4026 [0.3945, 0.4107]	0.3966 [0.3880, 0.4050]
HumanEval score	0.0247 [0.0190, 0.0305]	0.0251 [0.0200, 0.0308]
HumanEval PSS	0.0480 [0.0376, 0.0583]	0.0485 [0.0375, 0.0599]

The full controlled configuration shows modest aggregate PSS reduction and small score changes. ARC-Challenge and CommonsenseQA show lower PSS under geometry, while HumanEval remains noisy and close across settings.

Importance-Uniformity Family

The importance-uniformity family is included because it gives a strong score-oriented result and was one of the important candidate families during model selection.

Table B.12: Importance-uniformity family: ProSA 50%.

Candidate	λ_{geom}	Avg score	Overall PSS	Benchmark-level note
Control	0	0.2625	0.2500	Reference for this family.
Geometry, moderate	50	0.2623	0.2479	PSS decreases slightly.
Geometry, main point	100	0.2642	0.2481	Highest score in this family

This family illustrates why component ablations and family-level selection should be interpreted separately. Importance uniformity is not the main mechanism in the controlled ablation, but configurations that include it can still be practically useful under score-oriented selection.

Matched Seeds

The matched-seed check tests whether the geometry-associated pattern appears in more than one training seed. The complete seed-level deltas are shown in Table B.13. Score delta is geometry-enabled minus control. PSS gain, Load CV reduction, and router-neighbour L_2 reduction are positive when the corresponding metric improves.

Table B.13: Matched-seed deltas.

Configuration family	Seed	Score Δ	PSS gain	Load CV reduction	L_2 reduction
Geometry with router-logit stabilization	43	-0.0011	$+0.0054$	$+0.1281$	$+1.333$
Geometry with router-logit stabilization	44	-0.0011	$+0.0042$	$+0.1207$	$+1.384$
Full controlled geometry configuration	43	$+0.0017$	$+0.0033$	$+0.1213$	$+1.388$
Full controlled geometry configuration	44	$+0.0012$	$+0.0036$	$+0.1230$	$+1.470$
Geometry with router-logit stabilization and sequence guard	43	$+0.0000$	$+0.0060$	$+0.1189$	$+1.529$
Geometry with router-logit stabilization and sequence guard	44	$+0.0003$	$+0.0025$	$+0.1161$	$+1.526$
Geometry with router-logit stabilization and importance uniformity	43	$+0.0009$	$+0.0033$	$+0.1233$	$+1.408$
Geometry with router-logit stabilization and importance uniformity	44	-0.0021	$+0.0040$	$+0.1222$	$+1.391$

The matched-seed results support a consistency claim for the router diagnostics

and prompt sensitivity. Every matched pair improves Load CV, router-neighbour L_2, and PSS. Score changes are small and mixed, matching the main conclusion of Chapter 5.

Component Ablations

The component-ablation study tests the role of confidence weighting, router-logit stabilization, sequence guard, and importance uniformity. The no-confidence ablation is qualitatively different from the other removals because it changes the scale of the geometric objective and causes severe degradation.

Table B.14: Confidence-weighting ablation.

Variant	Loss	PPL	Load CV	Entropy	Router L_2	PSS
Full controlled reference	3.232	25.34	0.823	3.919	6.8	0.244
No confidence weighting	9.035	8389	1.683	2.396	56.0	0.244

The confidence-weighting ablation shows that a similar PSS value can be misleading when the model has collapsed. The no-confidence run has unusable perplexity, worse load balance, lower entropy, and much larger router drift. It is therefore interpreted as a failure case rather than a valid alternative configuration.

Table B.15 summarizes the controlled component removals.

Table B.15: Controlled component removals.

Ablation	Main effect	Interpretation
Geometry without confidence weighting	Geom. loss: 0.4287 $\rightarrow$ 26.5227; PPL: 8388.	Loss-scale failure.
No router-logit stabilization	PPL $+0.739$.	Protects validation quality.
No sequence guard	PSS $+0.0037$.	Supports prompt sensitivity.
No importance uniformity	Slight changes.	Small marginal effect.
No sequence guard / no importance uniformity	PPL worsens more than either alone.	Possible joint support.

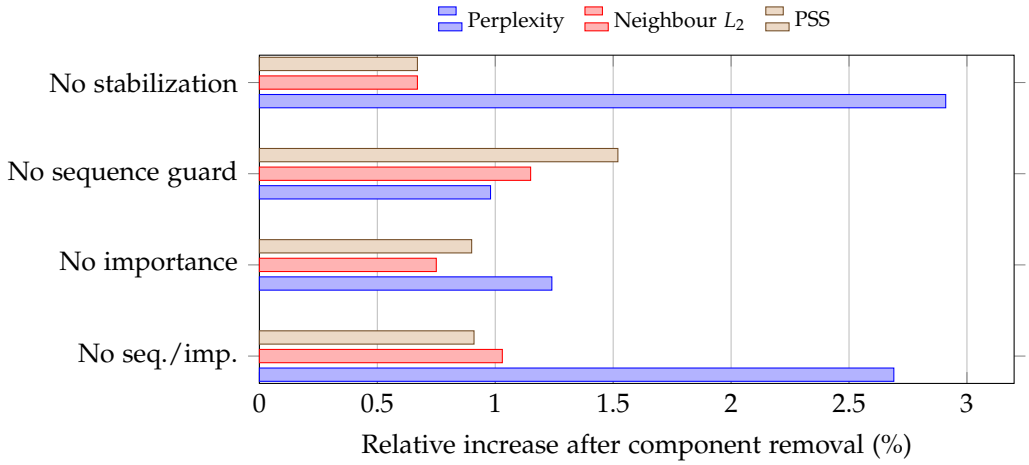


Figure B.4: Relative increase in lower-is-better metrics after removing controlled components from the full controlled geometry configuration. The no-confidence-weighting run is omitted because it is a loss-scale failure case.

The ablation clarifies the component roles. Geometry is the main router-structure mechanism. Confidence weighting prevents loss-scale failure. Router-logit stabilization protects validation quality. The sequence guard contributes most clearly to prompt sensitivity reduction. Importance uniformity is a secondary component in the fixed ablation but remains useful in score-oriented configuration selection.

Layer Routing

Layer-averaged metrics can hide whether the geometry objective affects the whole MoE stack or only a small number of layers. The per-layer diagnostics inspect Load CV, entropy, router-neighbour distance, and router drift across the 16 OLMoE router layers.

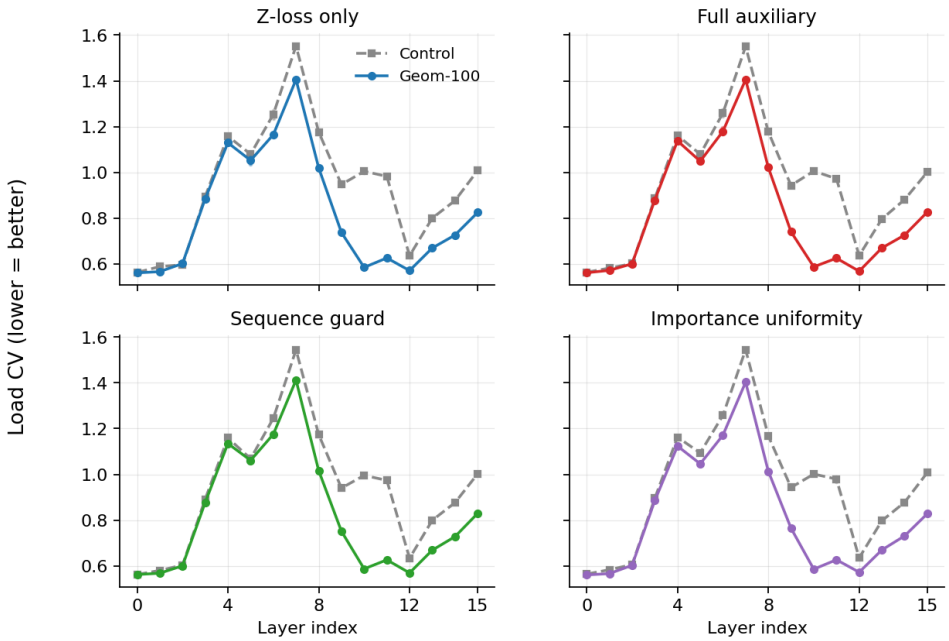


Figure B.5: Per-layer Load CV. Geometry-aware regularization reduces Load CV across the MoE stack, with the largest reductions in the middle layers.

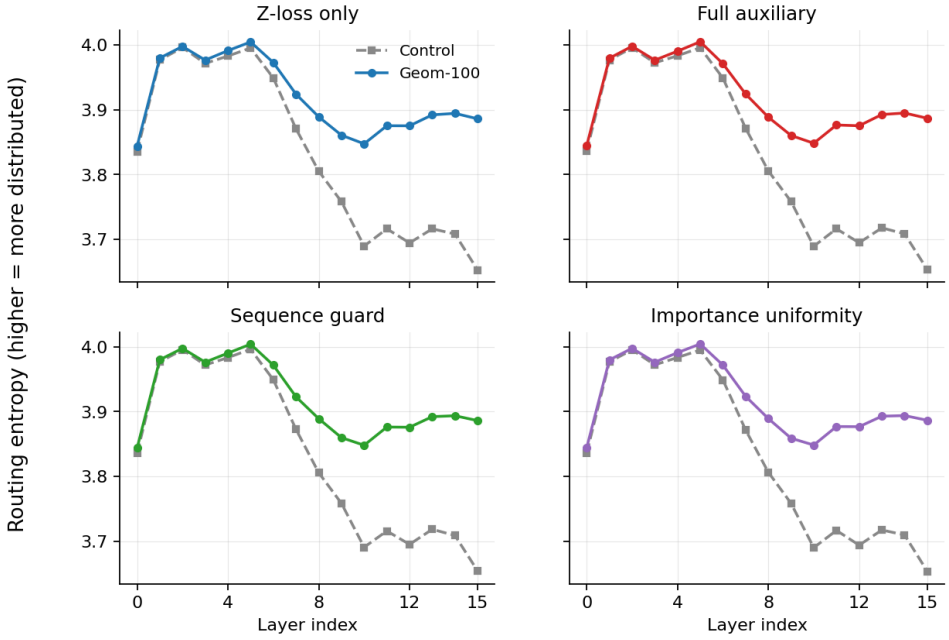


Figure B.6: Per-layer routing entropy. Geometry-aware regularization raises entropy across the MoE stack, indicating more distributed routing probabilities.

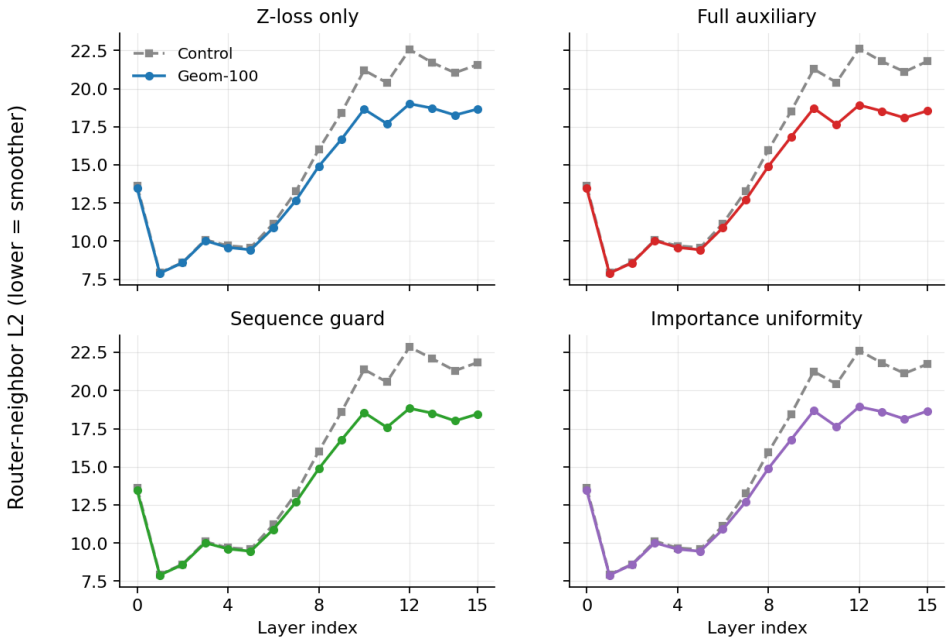


Figure B.7: Per-layer router-neighbour L_2. Geometry-aware regularization lowers local router-neighbour L_2 across layers, supporting the local-smoothness interpretation.

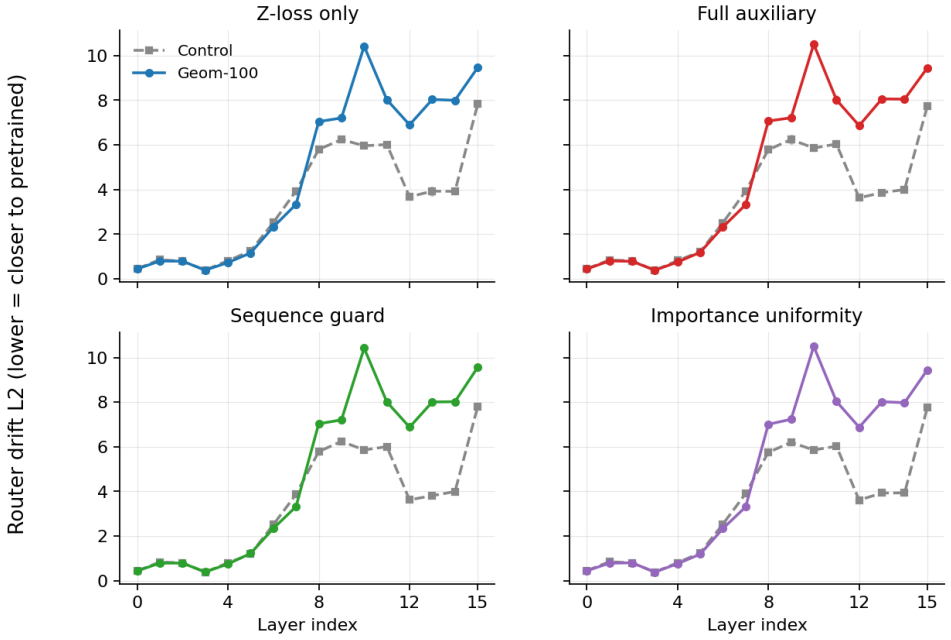


Figure B.8: Per-layer router drift from the pretrained router. Geometry-aware regularization increases drift, which should be interpreted as a cost of reshaping the router.

Table B.16: Layer-group interpretation.

Layer group	Main pattern	Interpretation
Early layers, 0–3	Moderate Load CV and entropy changes.	Geometry affects early routing, but the largest redistributions occur later.
Middle layers, 4–10	Largest Load CV and expert-distribution changes.	Control routing is most concentrated here, so redistribution is strongest.
Late layers, 11–15	Clear router-neighbour L_2 reduction with increased but bounded drift.	Local smoothing remains active near the top of the MoE stack.

The per-layer diagnostics support a stack-wide interpretation. The geometry objective does not merely perturb one router layer; it changes routing statistics across the MoE stack. At the same time, the drift plots show that these changes are not cost-free.

Manifold Diagnostics

The manifold diagnostics examine whether local hidden-state neighbourhoods align with router behaviour. These diagnostics are not used as direct optimization targets in the final evaluation; they are used to interpret the geometry of the learned router map.

Table B.17 reports routing-overlap diagnostics across several configuration variants. The configurations shown include the main geometry families at seed 43, their matched controls, and a half-auxiliary variant. The *half-auxiliary* configuration uses importance-uniformity and sequence-guard weights at half the strength of the full controlled setting ($\lambda_{\text{imp}} = 0.0025$ versus 0.005; $\lambda_{\text{seq}} = 0.0005$ versus 0.001). It is included here to show that reducing the secondary auxiliary term weights does not change the manifold diagnostic pattern.

Table B.17: Routing-overlap and neighbourhood diagnostics. Top-1 agreement is the fraction of kNN neighbour pairs assigned the same top-1 expert. Hidden cosine mean is the mean cosine similarity between neighbour hidden-state pairs. Hidden-router correlation is the correlation between hidden-state cosine similarity and routing probability agreement across neighbour pairs. The half-auxiliary row uses halved importance-uniformity and sequence-guard weights relative to the full controlled setting.

Candidate	Top-1 agreement	Hidden cosine mean	Hidden-router correlation
Sequence guard, Geom-100, seed 43	0.4462	0.5025	−0.4247
Importance uniformity, Geom-100, seed 43	0.4448	0.5018	−0.4232
Geometry with router-logit stabilization, Geom-100, seed 43	0.4456	0.5008	−0.4260
Geometry with router-logit stabilization, control, seed 43	0.4667	0.5050	−0.4386
Full controlled geometry configuration, Geom-100, seed 43	0.4459	0.5029	−0.4248
Full controlled geometry configuration, control, seed 43	0.4660	0.5045	−0.4393
Half auxiliary, Geom-100	0.4469	0.5027	−0.4243
Half auxiliary, control	0.4663	0.5030	−0.4361

Top-1 agreement is consistently lower under Geom-100 across all configurations shown, including the half-auxiliary variant. This pattern holds regardless of whether the secondary auxiliary term weights are at full or half strength. It shows that the manifold diagnostic result is not sensitive to the exact weight of the smaller auxiliary components.

The decrease in top-1 agreement does not contradict the router-neighbour L_2 result. It shows that continuous smoothing of router logits should not be equated with forcing neighbouring tokens to choose the same top-1 expert more often. The method regularizes router distributions and logits, not hard top-1 identity.

Summary

This appendix chapter provides the detailed evidence behind the main router-only OLMoE testing results. The strongest and most consistent evidence is structural: geometry reduces Load CV, increases routing entropy, and lowers router-neighbour logit distance. Prompt sensitivity also decreases in the primary and matched comparisons, although raw downstream score changes remain small and mixed.

The extended results also clarify the trade-offs. Jitter and base-router drift increase when geometry is enabled or strengthened. Confidence weighting is necessary to

prevent the geometric objective from becoming numerically uncontrolled. Router-logit stabilization protects validation quality, and the sequence guard contributes most clearly to prompt sensitivity reduction.

The detailed results therefore support the same conclusion as Chapter 5: geometry-aware routing is best understood as a router-structure and prompt-sensitivity intervention, not as a general raw-accuracy improvement method.

Appendix C

Sensitivity and External Validity

This appendix chapter reports the sensitivity and external-validity evidence behind Section 5.7. The main Testing chapter focuses on the cleanest router-only OLMoE post-training setting. This appendix broadens the evidence base by asking whether the geometry-associated router-structure pattern remains visible when the trainable scope, adaptation data, or MoE backend changes.

The evidence in this appendix should be interpreted carefully. These sweeps are not the primary causal evidence for the thesis claim. The primary evidence remains the router-only experiments in Appendix B. The sweeps reported here are used to test whether the structural effect is fragile or whether it appears across reasonable changes in experimental setting.

Interpretation Rules

The sensitivity and external-validity analyses answer three questions:

- Does the geometry-associated router-structure pattern disappear when the trainable scope is widened beyond routers?
- Does the structural effect appear across different adaptation datasets within the same OLMoE model family?
- Does the structural effect remain visible across additional MoE model families?

These questions are related but not equivalent. The trainable-scope sweep changes

which parameters are allowed to move. It therefore weakens attribution to the router, because attention layers, normalization parameters, experts, or full transformer blocks may also change. The model/data sweep changes either the adaptation data or the MoE backend. It therefore supports external validity, but most comparisons are single-seed and should not be interpreted as seed-level reproducibility evidence.

Table C.1 summarizes how the results in this appendix are interpreted.

Table C.1: Interpretation rules.

Analysis	Tests	Limit
Trainable-scope sensitivity	Effect under wider trainable boundaries.	Weaker router-only attribution.
OLMoE dataset sweep	Effect across adaptation datasets.	Mostly single-seed; not seed-level reproducibility.
Additional MoE backend sweep	Effect across MoE backends.	Backend results are less directly comparable.
Incomplete or missing evaluations	Run traceability.	Not full evidence for all metrics.

The main quantity tracked across all sweeps is router-neighbour logit distance, because it is the most direct diagnostic of the local router-neighbour effect. Load CV is also tracked because expert-load balance is a central practical concern in sparse MoE routing. Downstream score and PSS are interpreted as secondary effects: useful when available, but not required for establishing that the geometry objective changes router structure.

Trainable Scope

The main thesis experiments use router-only finetuning because this setting gives the cleanest attribution. If only router parameters are trainable, then changes in routing diagnostics can be attributed to changes in the router policy rather than changes in expert functions, attention representations, or full-block dynamics.

The trainable-scope sweep tests whether the geometry-associated structural pattern remains visible when this boundary is relaxed. This is useful as a sensitivity analysis, but it is not cleaner evidence than the router-only setting. When attention layers, normalization parameters, experts, or full blocks are trainable, downstream changes may be caused by interactions between the geometric objective and other moving parameters.

Summary

Table C.2 reports the main trainable-scope comparison. Values are computed as geometry-enabled minus the corresponding control setting, except for perplexity where both values are shown.

Table C.2: Trainable-scope summary.

Trainable scope	PPL control → geometry	Δ Load CV	Δ L_2	Δ score	Δ PSS
Router only	25.355 → 25.348	−0.1244	−1.352	−0.0005	−0.0024
Router + norms	32.460 → 33.925	−0.1137	−1.191	+0.0022	−0.0040
Router + attention + norms	4.307 → 4.930	−0.2393	−2.447	−0.0517	−0.0374
Router + experts	7.227 → 7.173	−0.1778	−1.759	+0.0010	−0.0065
Full block	4.221 → 4.860	−0.2383	−2.553	−0.0675	−0.0490

The structural geometry-associated pattern appears in every trainable scope. Load CV decreases in all five comparisons, and router-neighbour logit distance also decreases in all five comparisons. This supports the interpretation that the geometry objective acts on a real router-structure signal rather than on a single fragile router-only run.

However, the downstream interpretation changes when the trainable scope becomes broader. The router + attention + norms and full-block settings show large downstream score drops, even though they also show large PSS reductions. These PSS reductions are therefore partly confounded by weaker task performance. The router + experts setting is more promising because it preserves score and improves PSS, but it is still less clean than router-only finetuning because expert functions are trainable.

Interpretation

The trainable-scope sweep supports three conclusions.

First, the structural effect is not limited to the router-only setting. Geometry reduces router-neighbour logit distance and usually improves expert-load balance even when the trainable parameter set is widened.

Second, broader trainable scopes are not automatically better. A larger structural change can coincide with downstream score degradation. This is particularly visible in the router + attention + norms and full-block settings. Therefore, stronger PSS reductions in broad-scope settings should not be interpreted as unqualified improvements.

Third, the cleanest thesis evidence remains the router-only setting. Router-only finetuning isolates the change to the routing policy and therefore provides the clearest attribution for the method. Broader trainable scopes are useful as sensitivity evidence and future-work directions, but they do not replace the router-only result.

Table C.3: Trainable-scope interpretation.

Observation	Interpretation
All scopes reduce router-neighbour L_2.	The geometry objective consistently affects the local router-neighbour diagnostic.
All scopes reduce Load CV.	The structural change usually coincides with improved aggregate expert-load balance.
Broad scopes can reduce PSS strongly while reducing score.	Prompt-sensitivity reductions can be confounded by weaker task performance.
Router + experts preserves score and reduces PSS.	This is a promising extension, but attribution is weaker because expert functions also move.
Router-only remains the central setting.	It gives the cleanest causal interpretation of the geometry-aware router intervention.

Model/Data Sweep

The model/data sweep tests whether the geometry-associated structural effect is specific to one adaptation dataset or one MoE backend. The sweep has two parts. The dataset sweep keeps the OLMoE model family fixed and changes the adaptation data. The model sweep keeps UltraChat-style adaptation fixed and changes the MoE backend.

The dataset sweep is the stronger external-validity analysis because it changes the data while keeping the model family fixed. The additional model sweep is more heterogeneous because backend architectures, routing details, and pretrained states differ across model families.

Dataset Sweep

The OLMoE dataset sweep keeps the model family fixed and changes the adaptation corpus. This makes it the clearest external-validity test in this appendix. Table C.4 reports the compact sweep summary.

Table C.4: OLMoE dataset sweep.

Adaptation data	Perplexity control → geometry	Router-structure change	Downstream / prompt-sensitivity change
UltraChat anchor	26.070 → 25.919	Load CV −0.1198; neighbour L_2 −1.463.	Score +0.0019; PSS −0.0019.
C4	129.672 → 127.319	Load CV −0.1510; neighbour L_2 −1.878.	Score +0.0025; PSS −0.0007.
FineWeb-Edu	118.326 → 115.969	Load CV −0.1658; neighbour L_2 −2.310.	Score −0.0026; PSS −0.0101.
WikiText-103	85.945 → 81.528	Load CV −0.1470; neighbour L_2 −1.553.	Score +0.0024; PSS −0.0046.

The OLMoE dataset sweep shows a consistent structural result. Across UltraChat, C4, FineWeb-Edu, and WikiText-103, geometry reduces Load CV and router-neighbour logit distance. This indicates that the router-structure effect is not specific to the original UltraChat adaptation data.

The downstream and prompt-sensitivity results are more mixed. Score improves slightly for UltraChat, C4, and WikiText-103, but decreases slightly for FineWeb-Edu. PSS improves in all four OLMoE dataset comparisons, with the clearest reductions on FineWeb-Edu and WikiText-103. This supports a cautious external-validity claim: the structural effect transfers across adaptation data, while downstream score remains setting-dependent.

Table C.5: OLMoE dataset-sweep interpretation.

Observation	Interpretation
Load CV decreases for all completed datasets.	Geometry improves aggregate expert-load balance across adaptation corpora.
Router-neighbour L_2 decreases for all completed datasets.	The local router-neighbour effect is not specific to UltraChat.
PSS decreases for all completed datasets.	Prompt sensitivity improves directionally across the OLMoE data sweep.
Score changes are small and mixed.	Geometry should not be interpreted as a reliable raw-accuracy improvement method.
GitHub Python is incomplete.	It is excluded from quantitative interpretation.

Backend Sweep

The additional model sweep changes the MoE backend while using UltraChat-style adaptation. This is a harder external-validity test because the underlying pretrained models, router implementations, expert layouts, and baseline routing behaviours differ.

Table C.6 summarizes the available model-family results. The Phi-3.5 MoE row is included only for language-modeling and router diagnostics because ProSA was not evaluated.

Table C.6: Additional MoE backend sweep.

MoE backend	Perplexity control $\rightarrow$ geometry	Router-structure change	Downstream / prompt-sensitivity change
Granite 1B / UltraChat	3.403 $\rightarrow$ 3.404	Load CV -0.0123; neighbour L_2 -0.310.	Score $+0.0025$; PSS $+0.0009$.
Granite 3B / UltraChat	3.102 $\rightarrow$ 3.098	Load CV $+0.0000$; neighbour L_2 -0.068.	Score -0.0016; PSS $+0.0002$.
Phi-3.5 MoE / UltraChat	2.942 $\rightarrow$ 2.898	Load CV -0.0218; neighbour L_2 -0.170.	ProSA not evaluated.
Phi tiny MoE / UltraChat	3.325 $\rightarrow$ 3.316	Load CV -0.0034; neighbour L_2 -0.154.	Score -0.0061; PSS -0.0023.

Across the additional MoE backends, router-neighbour logit distance decreases in every completed comparison. This supports the structural interpretation of geometry-aware regularization: the method continues to affect local router-neighbour behaviour outside the original OLMoE setting.

However, the effect is weaker and less uniform than in the OLMoE dataset sweep. Load CV changes are smaller, and downstream/PSS effects are not consistently favourable. Granite 1B improves score but slightly worsens PSS. Granite 3B shows nearly unchanged Load CV and a slight PSS worsening. Phi tiny reduces PSS but decreases score. Phi-3.5 MoE cannot be used for ProSA conclusions because ProSA was not evaluated.

Table C.7: Backend-sweep interpretation.

Observation	Interpretation
Router-neighbour L_2 decreases for all listed backends.	The local structural effect remains visible across model families.
Load CV changes are small or inconsistent outside OLMoE.	Aggregate expert-load effects are more backend-dependent.
Score and PSS effects are mixed.	Downstream and prompt-sensitivity benefits do not transfer uniformly across MoE backends.
Phi-3.5 MoE lacks ProSA evaluation.	It supports only routing and language-modeling interpretation.
OLMoE data sweep is stronger evidence.	Keeping the model family fixed gives a cleaner external-validity test than changing both architecture and routing implementation.

Sweep Comparison

The dataset and model sweeps support different levels of evidence. The OLMoE dataset sweep is stronger because it changes the adaptation data while keeping the model family fixed. The additional MoE backend sweep is broader, but harder to interpret because each model family may have different router architecture, expert layout, pretraining distribution, and baseline load behavior.

Table C.8 compares the two sweep types.

The overall interpretation is that geometry-aware regularization generalizes most clearly as a structural router regularizer. The strongest evidence is within OLMoE across adaptation datasets. Across other MoE backends, the local router-neighbour effect remains visible, but downstream and prompt-sensitivity benefits are more model-dependent.

Table C.8: Dataset sweep versus backend sweep.

Dimension	OLMoE dataset sweep	Additional MoE backend sweep
What changes?	Adaptation data changes; model family remains fixed.	MoE backend changes; adaptation data remains broadly comparable.
Attribution clarity	Stronger, because architecture and routing implementation are fixed.	Weaker, because architecture and router design differ.
Router-neighbour effect	Consistently decreases across completed datasets.	Decreases across listed backends, but magnitudes are smaller.
Load CV effect	Consistently decreases across completed datasets.	Smaller and less consistent.
Downstream/PSS effect	PSS improves across completed datasets; score remains mixed.	Score and PSS effects are model-dependent.
Main conclusion	Strong external-validity evidence for the structural effect across adaptation data.	Broader but more cautious evidence that the local router-neighbour effect can appear across MoE families.

Summary

The sensitivity and external-validity results support the main Testing chapter in three ways.

First, the trainable-scope sweep shows that the geometry-associated structural pattern does not disappear when the trainable boundary is widened. Load CV and router-neighbour logit distance improve in every trainable-scope comparison. However, broad trainable scopes can confound PSS reductions with downstream score degradation, so they are not cleaner evidence than router-only finetuning.

Second, the OLMoE dataset sweep shows that the structural effect is not tied to the original UltraChat adaptation data. Across UltraChat, C4, FineWeb-Edu, and WikiText-103, geometry reduces Load CV and router-neighbour logit distance. PSS also decreases across the completed OLMoE dataset comparisons, although raw downstream score remains mixed.

Third, the additional MoE backend sweep shows that the local router-neighbour effect remains visible outside OLMoE, but downstream and prompt-sensitivity effects are more model-dependent. This supports a careful external-validity claim: geometry-aware routing generalizes most strongly as a router-structure regularizer,

not as a universal downstream-accuracy improvement method.

The conclusion of this appendix is therefore aligned with Chapter 5: the method is best understood as a router-behaviour intervention. The evidence supports a structural claim about local router-neighbour behaviour and expert-load balance. It does not support a broad claim that geometry-aware regularization gives model-independent downstream accuracy gains.

Table C.9 summarizes the evidence sources and their limitations.

Table C.9: Sensitivity and external-validity summary.

Evidence source	Supported conclusion	Main limitation
Trainable-scope sensitivity	Geometry reduces Load CV and router-neighbour distance across trainable scopes.	Wider scopes weaken attribution and can reduce downstream score.
OLMoE dataset sweep	Structural effect transfers across adaptation datasets.	Mostly single-seed and still OLMoE-specific.
Additional MoE backend sweep	Router-neighbour effect is visible across several MoE backends.	Load CV, score, and PSS effects are model-dependent.

Appendix D

Compute Cost

This appendix reports the computational cost of the router-only geometry-aware post-training experiments. The purpose is to support the computational-feasibility discussion in Section 5.8, while keeping the main Testing chapter focused on the experimental findings.

The main question addressed here is not the monetary cost of the experiments. Instead, this appendix documents runtime, throughput, peak memory, trainable parameter count, and the additional overhead introduced by the geometry objective. The results show that the standard OLMoE router-only setting is computationally controlled, while the broader model/data and trainable-scope sweeps are more expensive.

Source and Hardware

The compute analysis is based on saved training-summary files produced by the local training runs. These summaries record wall-clock runtime, mean training throughput, token throughput, peak GPU memory, trainable parameter count, total training steps, and maximum sequence length.

Unless otherwise stated, the local post-training runs reported in this appendix were executed on a single RTX 6000 Blackwell-series GPU. Therefore, reported GPU-hours are interpreted as single-GPU runtime hours for that hardware setting. Monetary cost is not estimated because GPU pricing was not logged.

The pretraining pilot in Appendix E used a different experimental setup and is not part of the post-training compute-overhead comparison in this appendix. The com-

pute values reported here therefore refer to the router post-training experiments used in the Testing chapter.

Table D.1 summarizes the main compute fields used throughout this appendix.

Table D.1: Compute fields.

Field	Meaning
Wall time	Elapsed training time for a run, reported in minutes.
GPU-hours	Wall-clock runtime expressed in hours on one RTX 6000 Blackwell-series GPU.
Steps/s	Mean number of optimization steps completed per second.
Tokens/s	Approximate mean number of training tokens processed per second.
Peak memory	Peak GPU memory allocated during the run, reported in GiB.
Trainable parameters	Number of parameters updated during training. For standard OLMoE router-only runs, this is approximately 2.10M parameters.
Approximate train tokens	Maximum-token estimate based on training steps and maximum sequence length. This is an upper-bound style accounting measure, not a statement that every sequence was fully packed.

Compute Audit

Table D.2 summarizes the scanned training outputs. The local training outputs are the primary reference for the Testing chapter. The quick outputs are retained as an audit trail because they come from earlier or separate run folders with different throughput behaviour.

Table D.2: Scanned training outputs.

Output group	Runs	Approx. train tokens	GPU-hours
Local training outputs	61	312.32M	22.07
Quick outputs	7	35.84M	8.49
Total scanned outputs	68	348.16M	30.55

The main Testing chapter uses the local training outputs as the computational reference. These include the ablation, global-balance, half-auxiliary, lambda-sweep, model/data-sweep, multiseed, multiseed-priority, and trainable-scope runs. The quick outputs are not used as the central cost estimate because they have different runtime characteristics.

Router-Only Cost

The standard OLMoE router-only runs are the most relevant compute reference for the main thesis claim. In these runs, the model backbone, experts, attention layers, embeddings, and language-model head are frozen. Only the router parameters are updated.

Table D.3 summarizes the typical standard router-only cost.

Table D.3: Typical router-only cost.

Quantity	Typical value
Training steps	5000
Maximum sequence length	1024
Approximate maximum-token budget per run	5.12M tokens
Trainable parameters	2,097,152
Peak GPU memory	≈14.93 GiB
Wall-clock time	≈18–20 min
Runtime per run	≈0.30–0.33 GPU-hours
Throughput	≈4.3–4.8 steps/s
Token throughput	≈4.4–4.9k tokens/s

The trainable parameter count is small relative to the full sparse model. The cost is therefore dominated by forward and backward execution through the frozen model, plus the additional local-neighbour computation introduced by the geometry objective. This distinction is important: router-only geometry-aware post-training is not free, but it remains much cheaper and more controlled than broad trainable-scope experiments.

Family Summary

Table D.4 summarizes runtime and memory by local training experiment family. The table is intentionally compact; detailed raw timing values are kept in the run-summary files.

The standard OLMoE router-only experiment families remain close to 14.93 GiB peak memory. This includes the ablation, global-balance, half-auxiliary, lambda-sweep, multi-seed, and multi-seed-priority runs. The model/data sweep and trainable-scope sweep have higher mean memory because they include larger MoE backends or broader trainable parameter sets.

Table D.5 reports throughput for the same experiment families.

Table D.4: Runtime and memory by experiment family.

Experiment family	Runs	Train tokens	GPU-hours	Mean wall time	Mean peak memory
Ablation	6	30.72M	1.89	18.90 min	14.93 GiB
Global balance	2	10.24M	0.61	18.44 min	14.93 GiB
Half auxiliary	2	10.24M	0.60	17.88 min	14.93 GiB
Lambda sweep	8	40.96M	2.68	20.07 min	14.93 GiB
Model/data sweep	16	81.92M	6.86	25.74 min	21.03 GiB
Multi-seed	9	46.08M	2.74	18.27 min	14.93 GiB
Multi-seed priority 2	8	40.96M	2.41	18.07 min	14.93 GiB
Scope sweep	10	51.20M	4.28	25.67 min	34.76 GiB

Table D.5: Throughput by experiment family.

Experiment family	Mean steps/s	Median steps/s	Mean tokens/s	Median tokens/s
Ablation	4.41	4.42	4513.93	4526.39
Global balance	4.53	4.53	4639.29	4639.29
Half auxiliary	4.68	4.68	4787.87	4787.87
Lambda sweep	4.25	4.42	4344.22	4526.23
Model/data sweep	3.69	4.16	3778.14	4261.24
Multi-seed	4.69	5.03	4797.83	5143.94
Multi-seed priority 2	4.63	4.65	4734.58	4758.07
Scope sweep	3.50	2.84	3576.04	2911.66

The throughput table shows the same pattern as the memory table. Standard router-only OLMoE runs cluster around 4.3–4.8 steps/s. The model/data and trainable-scope sweeps are slower because they include different backends or broader trainable parameter sets.

The compute distribution supports the design of the Testing chapter. The main router-only experiments are relatively inexpensive and controlled. The broader sweeps are valuable for sensitivity and external-validity analysis, but they should not define the practical cost of the core method.

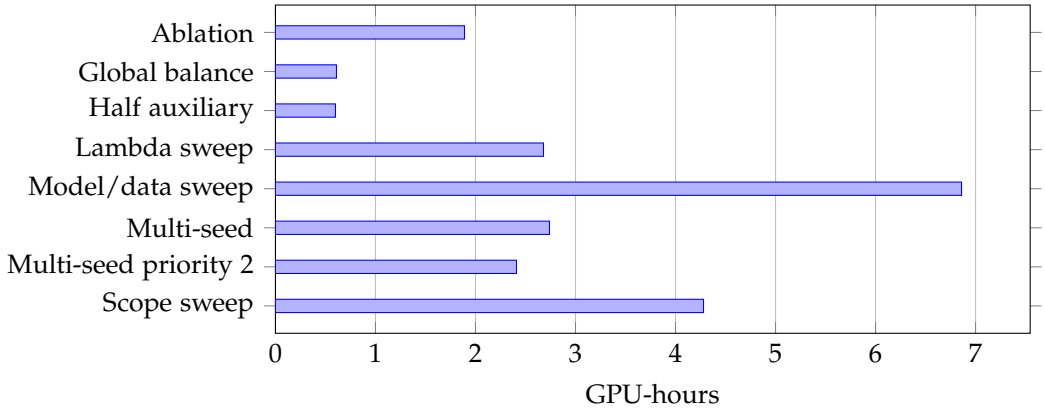


Figure D.1: Single-GPU runtime by local experiment family. The model/data and trainable-scope sweeps are the most expensive groups because they include different backends or broader trainable scopes.

Geometry Overhead

The most relevant overhead question is how much additional runtime is introduced when the geometry objective is added to an otherwise matched training setup. Matched comparisons show that the additional cost is moderate in the standard router-only setting.

The overhead arises primarily from constructing local hidden-state neighbourhoods and computing the router-neighbour divergence. The value of λ_{geom} scales the auxiliary loss after this computation and therefore is not itself the dominant runtime factor.

Table D.6 summarizes representative matched-overhead estimates.

Table D.6: Matched geometry overhead.

Matched comparison family	Wall-clock overhead	Token-throughput change
Global balance pair	+12.7%	−11.3%
Half-auxiliary pair	+13.1%	−11.6%
Model/data sweep pairs, median	+6.3%	−5.9%
Scope sweep pairs, median	+6.3%	−5.9%
Multi-seed priority pairs, median	+11.4%	−10.1%

A defensible summary for the main local runs is that geometry-aware routing typically adds approximately 6–13% wall-clock overhead in matched settings. Some individual runs are slower, especially in multi-seed folders, but those outliers are

not used as the central estimate because runtime can be affected by system contention and backend differences.

Lambda Overhead

The lambda sweep changes the strength of the geometry objective while keeping the same overall training structure. In principle, increasing λ_{geom} should not substantially increase compute once the neighbourhood computation is already active. The coefficient scales the computed loss, whereas the expensive part is the neighbour search and divergence calculation.

Table D.7 summarizes the lambda-sweep family-level timing.

Table D.7: Lambda-sweep compute summary.

Quantity	Value
Runs	8
Total single-GPU runtime	2.68 GPU-hours
Mean wall time	20.07 min
Median wall time	18.84 min
Mean steps/s	4.25
Median steps/s	4.42
Mean tokens/s	4344.22
Median tokens/s	4526.23
Mean peak memory	14.93 GiB
Median peak memory	14.93 GiB

The lambda-sweep family has the same peak memory footprint as the standard OLMoE router-only experiments. The mean wall time is slightly higher because some geometry-enabled settings add neighbour-computation overhead, but the memory footprint remains essentially unchanged.

Auxiliary Runtime

The geometry objective is the main additional compute driver. The smaller auxiliary components, such as confidence weighting, router-logit stabilization, sequence guard, and importance uniformity, have comparatively small runtime effects.

Table D.8 summarizes representative runtime effects from the component-ablation family. The values are relative to the full controlled geometry configuration.

Table D.8: Auxiliary component runtime effects.

Removed component	Wall-clock change	Token-throughput change
No confidence weighting	−0.8%	+0.8%
No router-logit stabilization	+4.6%	−4.4%
No sequence guard	−1.6%	+1.6%
No importance uniformity	+1.3%	−1.3%
No sequence guard and no importance uniformity	+2.4%	−2.3%

These values should not be overinterpreted as precise component-level runtime laws. Small differences in run timing can arise from system variability. The important point is that none of the auxiliary components dominate runtime in the way the geometry-neighbour computation does.

The confidence-weighting ablation is especially important methodologically. It does not substantially reduce runtime, but it causes a severe loss-scale failure in model quality. Therefore, it should not be removed for computational reasons.

Memory Footprint

The standard OLMoE router-only experiments have a stable peak-memory footprint of approximately 14.93 GiB. This is true across the ablation, global-balance, half-auxiliary, lambda-sweep, multi-seed, and multi-seed-priority families.

The memory footprint increases mainly when the experiment changes the model family or widens the trainable scope. This is visible in the model/data sweep and trainable-scope sweep summaries.

Table D.9 compares representative memory footprints.

The memory evidence supports the same methodological conclusion as the main Testing chapter. Router-only post-training is the most computationally controlled regime. Broader trainable-scope settings and larger MoE backends are useful for sensitivity and external-validity analysis, but they are not representative of the practical cost of the core router-only method.

Table D.9: Representative peak memory.

Experiment type	Representative peak memory
Standard OLMoE router-only	≈14.93 GiB
OLMoE lambda sweep	≈14.93 GiB
OLMoE multi-seed runs	≈14.93 GiB
Model/data sweep mean	21.03 GiB
Scope sweep mean	34.76 GiB
Router + experts scope setting	≈62.66 GiB
Full-block scope setting	≈64.22 GiB
Phi-3.5 MoE backend setting	≈84.73 GiB

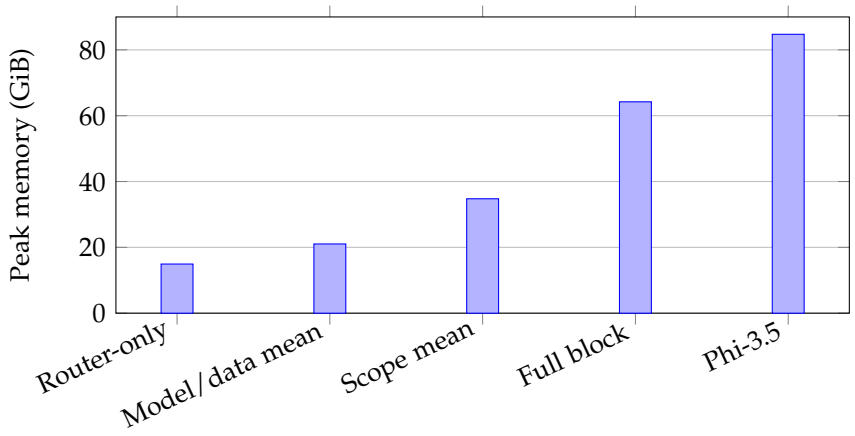


Figure D.2: Representative peak-memory requirements across experiment types. The standard router-only method remains near 14.93 GiB, while broader scopes and larger backends require substantially more memory.

Quick Runs

The quick output runs are retained for traceability but are not used as the central compute reference in Chapter 5. They have different runtime characteristics and should not be mixed directly with the local training runs when estimating standard router-only cost.

Table D.10 summarizes the quick outputs.

Table D.10: Quick output runs for router lambda ladder.

Quick output family	GPU-hours	Mean wall time	Mean peak memory
Router-logit stabilization	1.95	58.52 min	15.01 GB
Full controlled combination	3.22	64.40 min	15.01 GB
Half-auxiliary combination	3.31	99.43 min	15.01 GB

These quick outputs are slower than the main local router-only runs. They are therefore treated as historical or audit outputs rather than as the main runtime reference.

Method Interpretation

The compute results support three points about the main method.

First, the standard OLMoE router-only setting is computationally controlled. A typical run completes in approximately 18–20 minutes on a single RTX 6000 Blackwell-series GPU and uses approximately 14.93 GiB of peak memory. The trainable parameter count is only 2.10M.

Second, the geometry objective adds measurable but moderate overhead. Matched comparisons suggest a typical wall-clock overhead of approximately 6–13%. This overhead is caused by local-neighbour construction and router-neighbour divergence computation. The geometry coefficient itself does not dominate runtime once the geometry computation is active.

Third, the expensive settings are not the standard router-only experiments. The model/data sweep and trainable-scope sweep have higher runtime or memory because they include larger MoE backends or broader trainable parameter sets. These sweeps are important for sensitivity and external validity, but they should not define the practical cost of the main router-only method.

Summary

This appendix documents the computational cost of the post-training experiments used in Chapter 5. Across 61 local training runs, the experiments account for approximately 312.32M maximum training tokens and 22.07 GPU-hours on a single RTX 6000 Blackwell-series GPU. Standard OLMoE router-only runs complete in approximately 18–20 minutes, use about 14.93 GiB of peak memory, and update

about 2.10M trainable router parameters.

The geometry objective is computationally feasible but not cost-free. It typically adds approximately 6–13% wall-clock overhead in matched local comparisons. Peak memory remains essentially unchanged in the standard OLMoE router-only setting. The primary added cost comes from constructing local hidden-state neighbourhoods and computing the router-neighbour divergence.

The broader model/data and trainable-scope sweeps are more expensive, especially in memory. This reinforces the methodological choice made in the thesis: router-only finetuning is not only cleaner for attribution, but also the most computationally controlled regime for studying geometry-aware router regularization.

Appendix E

Pretraining Pilot

This appendix documents the pretraining pilot that was carried out in addition to the router-only post-training experiments reported in Chapter 5. The purpose of the pilot was to test whether the geometry-aware routing objective could also be used during from-scratch MoE pretraining, where token representations, router parameters, and expert parameters are all still being learned.

This appendix should be read as supporting evidence rather than as part of the main router-only thesis claim. The models used here are substantially smaller than `allenai/OLMoE-1B-7B-0125-Instruct`, and the downstream evaluations use small benchmark slices. The main value of the pilot is diagnostic: it shows that geometry-aware routing can be applied during pretraining, but that the balance between geometry pressure and load-balancing pressure is more delicate than in router-only post-training.

The key distinction is that post-training starts from a model with trained representations and trained experts, whereas pretraining starts from random initialization. In the post-training setting, the geometry objective reshapes an existing router. In the pretraining setting, the geometry objective can influence the formation of representations, routing clusters, and expert usage from the beginning of training. This makes the coefficient balance especially important.

Purpose and Scope

The pretraining pilot addresses three questions:

1. Can the geometry-aware routing objective be implemented during from-scratch

sparse MoE language-model pretraining?

2. How does geometry interact with load balancing when both representations and experts are still being learned?
3. Do geometry-trained pretraining checkpoints remain valid under small downstream and prompt-sensitivity checks?

The pilot is not used to claim that geometry-aware pretraining improves downstream accuracy. The model scale is small, the benchmark slices are limited, and the run count is not sufficient for strong statistical conclusions. Instead, the pilot is used to support a narrower claim: geometry-aware routing is technically applicable during pretraining, but it requires careful balancing against expert-load objectives.

Table E.1: Role of the pretraining pilot relative to the main thesis experiments.

Dimension	Main router-only experiments	Pretraining pilot
Model state	Pretrained OLMoE model with frozen experts and backbone.	Fresh randomly initialized TinyMoE-style model.
Trainable parameters	Router parameters only.	Full model trained from scratch.
Primary purpose	Test whether geometry reshapes an existing router map.	Test whether geometry can be active during router and expert formation.
Evidence strength	Main thesis evidence.	Supporting diagnostic evidence.
Main limitation	Router-only setting does not test from-scratch learning.	Small model scale and small downstream slices.

The pretraining pilot therefore extends the thesis argument without replacing the main evidence. It is included because it reveals an important practical point: geometry-aware routing can be used earlier in training, but the coefficient ratio between geometry and load balancing becomes critical.

Pretraining Setup

The pilot uses a fresh OLMoE-style sparse MoE causal language model trained on streaming C4 text. Tables E.2 and E.3 summarize the architecture and training setup.

Table E.2: Model setup used for the TinyMoE pretraining pilot.

Field	Value
Code directory	`pretraining/`
Entrypoint	`pretrain.py`
Model family	Fresh randomly initialized OLMoE-style sparse MoE causal LM
Tokenizer	`allenai/OLMoE-1B-7B-0125-Instruct` tokenizer
Total parameters	189,407,744 parameters
Active parameters	Approximately 76M active parameters per token
Transformer layers	12
Hidden size	512
Attention	8 query heads, 8 key-value heads, head dimension 64
Experts	8 experts per MoE layer
Routing	Top-2 expert routing
Expert FFN width	1024
Vocabulary size	50,304
Maximum sequence length	512 tokens

The effective token budget is calculated as:

4 sequences/GPU $\times$ 16 accumulation steps $\times$ 4 GPUs $\times$ 512 tokens $=$ 131,072 tokens/step.

At 16,000 optimizer steps, this gives:

$$131{,}072 \times 16{,}000 = 2{,}097{,}152{,}000$$

training tokens.

This hardware setup is different from the local post-training runs discussed in Appendix D. The post-training compute appendix reports single-GPU RTX 6000 Blackwell-series runs, while this pretraining pilot used 4 NVIDIA L4 GPUs. The two compute settings should therefore not be mixed when interpreting runtime or memory.

Table E.3: Training setup used for the TinyMoE pretraining pilot.

Field	Value
Dataset	`allenai/c4`, English split
Data format	Streaming text packed into fixed-length chunks with EOS separators
Training hardware	4 NVIDIA L4 GPUs, 24GB VRAM each
Training method	DDP, bf16 autocast, gradient checkpointing
Per-GPU batch size	4 sequences
Gradient accumulation	16 micro-steps
Effective batch	256 sequences per optimizer step
Tokens per optimizer step	131,072
Training budget	16,000 optimizer steps, approximately 2.097B tokens
Optimizer	AdamW
Learning rate schedule	$3 \cdot 10^{-4}$ with cosine decay to $3 \cdot 10^{-5}$
Warmup	2,000 optimizer steps
Weight decay	0.1, excluding biases and normalization parameters
Gradient clipping	1.0
Evaluation frequency	Every 1,000 optimizer steps
Evaluation set	500 packed C4 validation sequences

Loss Function

For each batch, the total pretraining loss is:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda_{\text{geom}}(s)\mathcal{L}_{\text{geom}} + \lambda_{\text{bal}}(s)\mathcal{L}_{\text{bal}} + \lambda_z(s)\mathcal{L}_z.$$

Here, $\mathcal{L}_{\text{task}}$ is the standard causal language-modeling cross-entropy loss. The geometry loss, $\mathcal{L}_{\text{geom}}$, is applied on all 12 MoE layers. For each layer, pre-gate hidden states are L2-normalized, and a local cosine-similarity kNN graph is constructed within a token window. Router probabilities are projected into a top-k renormalized probability space, and neighbouring tokens are encouraged to have similar routing distributions using Jensen–Shannon divergence. The neighbour divergences are weighted by a softmax over cosine similarities with temperature τ.

The load-balancing loss, $\mathcal{L}_{\text{bal}}$, is the global Switch-style load-balancing objective. It uses selected expert counts and mean router probabilities to discourage expert collapse. Counts are synchronized across DDP ranks and accumulated across the gradient-accumulation window. The z-loss, $\mathcal{L}_z$, penalizes large router log-sum-

exp values and is used to control router-logit scale during random-initialization training.

The important difference from router-only post-training is that the geometry loss affects training while representations, routers, and experts are still forming. A coefficient that is safe during post-training may be too strong during pretraining if it encourages premature local routing clusters before the load-balancing term can distribute traffic across experts.

Run Families

Three main pretraining runs are included in the pilot. The baseline run disables geometry and uses load balancing plus z-loss. The first geometry run uses a strong geometry coefficient and a weak load-balancing coefficient. The second geometry run, denoted `geom_v2`, rebalances the coefficients after observing expert-load imbalance in the first geometry run.

Table E.4: Pretraining run families and their purpose.

Run	λ_{geom}	λ_{bal}	Purpose
`tiny_moe_baseline`	0.0	0.2	Control run with load balancing and z-loss only.
`tiny_moe_geom`	5.0	0.2	Initial geometry run using strong geometry pressure.
`tiny_moe_geom_v2`	1.0	1.0	Rebalanced geometry run after observing load imbalance in `tiny_moe_geom`.

The initial geometry run used $\lambda_{\text{geom}} = 5.0$ and $\lambda_{\text{bal}} = 0.2$, giving an effective geometry-to-balance ratio of 25:1. This reduced adjacent-token routing variation, but it made the expert-load distribution substantially less balanced. The geometry term was too strong relative to the balancing term, so nearby tokens were encouraged toward similar experts faster than the balance loss could distribute load.

The rebalanced run corrected this by reducing λ_{geom} to 1.0 and increasing λ_{bal} to 1.0, changing the effective ratio from 25:1 to 1:1. This coefficient change is the main methodological lesson from the pilot.

Training and Validation Results

The validation metrics follow the same interpretation used in the Testing chapter. Loss and perplexity measure next-token prediction quality on held-out C4

sequences. Jitter measures adjacent-token routing variation. Routing entropy and expert-load CV measure how the router distributes tokens across experts. Lower loss, perplexity, jitter, and Load CV are preferred, while higher entropy is preferred when model quality is preserved.

Table E.5: C4 validation metrics from the pretraining runs. The initial geometry run is shown at step 12,000 because it was used diagnostically and was not the final rebalanced comparison.

Run	Step	Eval loss	PPL	Jitter	Load CV	Main reading
`tiny_moe_baseline`	16,000	3.4970	33.02	0.4321	0.1290	Control run; balanced expert usage.
`tiny_moe_geom`	12,000	3.5692	35.49	**0.3508**	0.4561	Low L_{geom}, but severe imbalance.
`tiny_moe_geom_v2`	16,000	**3.4941**	**32.92**	0.4136	0.1404	Rebalanced geometry run

Table E.5 shows the main pretraining pattern. The first geometry setting strongly reduced adjacent-token routing variation, but it increased Load CV from the baseline range of approximately 0.13 to 0.46. This is not a desirable pretraining state because it means that expert usage became substantially less balanced.

The rebalanced `geom_v2` run kept Load CV close to the baseline while preserving a smaller reduction in adjacent-token routing variation. It also achieved slightly better C4 validation loss and perplexity than the baseline at the final 16,000-step checkpoint.

Table E.6: Coefficient-balance interpretation of the pretraining pilot.

Run	Observed effect	Interpretation
`tiny_moe_baseline`	Balanced expert usage and normal validation behaviour.	Provides the control reference for from-scratch TinyMoE training.
`tiny_moe_geom`	Lower adjacent-token routing variation but much higher Load CV.	Geometry pressure was too strong relative to load balancing.
`tiny_moe_geom_v2`	Load CV close to baseline, slight validation improvement, and modest routing-variation reduction.	Rebalancing geometry and load balancing made the geometry objective usable during pretraining.

This behaviour differs from the router-only post-training setting. In post-training, the model already has trained token representations and trained experts, so the

geometry objective mainly nudges an existing router. In pretraining, both representations and experts are still emerging. A strong geometry signal can therefore encourage premature routing clusters before the experts have learned useful specialization. The `geom_v2` result suggests that geometry can be active during pretraining, but it must be balanced against a sufficiently strong load-balancing objective.

Downstream Validity Checks

The final baseline and `geom_v2` checkpoints were exported to HuggingFace-compatible directories and evaluated using small lm-eval slices. The benchmark limit was 0.02, meaning that only 2% of each benchmark was used. These results are therefore validity checks rather than strong downstream conclusions.

Table E.7: Small 2% lm-eval validity checks for the final pretraining checkpoints. Higher values are better.

Benchmark metric	Samples	Baseline	Geom v2
IFEval prompt strict accuracy	11	0.0909	**0.2727**
IFEval instruction strict accuracy	11	0.2632	**0.4211**
HellaSwag accuracy	201	0.3682	0.3682
HellaSwag normalized accuracy	201	**0.4030**	0.3881
ARC-Challenge accuracy	24	**0.1250**	0.0833
ARC-Challenge normalized accuracy	24	**0.2500**	0.0833
BoolQ accuracy	66	**0.6667**	0.6364
WinoGrande accuracy	26	**0.5769**	0.5385

The task-validity results are mixed. The `geom_v2` checkpoint performs better on the small IFEval slice, ties raw HellaSwag accuracy, and performs worse on the small multiple-choice slices. Because the sample sizes are small, these results should not be interpreted as a reliable downstream ranking. Their main role is to verify that the exported checkpoints run correctly in lm-eval and that the rebalanced geometry-trained model has not collapsed into unusable behaviour.

Prompt-Sensitivity Check

A small ProSA-style prompt-sensitivity evaluation was run on ARC-Challenge and CommonsenseQA. Each benchmark instance was evaluated with 12 prompt variants, using a 2% dataset limit. Lower PSS is better, while higher average accuracy is better.

Table E.8: Small 2% ProSA validity check for final pretraining checkpoints.

Dataset	Instances	Prompts	Baseline PSS	Geom v2 PSS	Baseline ACC	Geom v2 ACC
ARC-Challenge	24	12	0.2355	**0.1913**	**0.2674**	0.1979
CommonsenseQA	25	12	0.2073	**0.1406**	0.2067	**0.2267**

The prompt-sensitivity check is the most positive downstream diagnostic for the pretraining pilot. On both evaluated datasets, `geom_v2` has lower PSS than the baseline. On CommonsenseQA it also has slightly higher average accuracy, while on ARC-Challenge it trades lower PSS for lower average accuracy.

This pattern is consistent with the main thesis results. Geometry-aware routing can reduce prompt sensitivity, but this does not automatically imply higher benchmark accuracy. The pretraining result therefore supports the broader interpretation of the thesis: geometry is best understood as a routing-behaviour intervention, not as a guaranteed accuracy-improvement mechanism.

Summary of Pretraining Findings

The pretraining pilot supports four conclusions.

First, the geometry-aware routing objective can be implemented during from-scratch MoE pretraining without changing the main language-model architecture. The same general idea of local hidden-state neighbourhood regularization can be applied while the model is being trained from random initialization.

Second, coefficient balance is more delicate during pretraining than during router-only post-training. The initial geometry run used $\lambda_{\text{geom}} = 5.0$ and $\lambda_{\text{bal}} = 0.2$. This reduced adjacent-token routing variation, but it also produced severe expert-load imbalance. The rebalanced run used $\lambda_{\text{geom}} = 1.0$ and $\lambda_{\text{bal}} = 1.0$, restoring Load CV close to the baseline while retaining a smaller routing-variation reduction.

Third, the rebalanced geometry checkpoint remained valid under small downstream checks. It did not dominate the baseline across benchmark slices, but it also did not collapse. It improved the small IFEval slice, tied raw HellaSwag accuracy, and was weaker on several small multiple-choice slices.

Fourth, the small ProSA check supports the same qualitative pattern as the main post-training experiments. The rebalanced geometry checkpoint had lower PSS on both ARC-Challenge and CommonsenseQA, but the accuracy changes were mixed.

This reinforces the interpretation that geometry-aware routing can reduce prompt sensitivity without guaranteeing raw downstream accuracy gains.

Overall, the pretraining pilot indicates that geometry-aware routing is not limited to router-only post-training. It can also be applied during from-scratch sparse MoE pretraining. However, the pilot also shows that pretraining is more sensitive to the ratio between geometry pressure and load-balancing pressure. For the scope of this thesis, the pretraining pilot is therefore best treated as supporting evidence and as a direction for future large-scale evaluation, not as a replacement for the main router-only post-training results.

Bibliography

[1] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[2] Jordan Hoffmann et al. "Training Compute-Optimal Large Language Models". In: *Advances in Neural Information Processing Systems*. 2022. URL: https://arxiv.org/abs/2203.15556.

[3] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[4] Noam Shazeer et al. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer". In: *International Conference on Learning Representations (ICLR)*. 2017. URL: https://arxiv.org/abs/1701.06538.

[5] Dmitry Lepikhin et al. *GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding*. 2020. arXiv: 2006.16668 [cs.CL]. URL: https://arxiv.org/abs/2006.16668.

[6] William Fedus, Barret Zoph, and Noam Shazeer. "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity". In: *Journal of Machine Learning Research* 23.120 (2022), pp. 1–39. URL: https://jmlr.org/papers/v23/21-0998.html.

[7] Barret Zoph et al. *ST-MoE: Designing Stable and Transferable Sparse Expert Models*. 2022. arXiv: 2202.08906 [cs.CL]. URL: https://arxiv.org/abs/2202.08906.

[8] Yanqi Zhou et al. *Mixture-of-Experts with Expert Choice Routing*. 2022. arXiv: 2202.09368 [cs.LG]. URL: https://arxiv.org/abs/2202.09368.

[9] Zihan Qiu et al. "Demons in the Detail: On Implementing Load Balancing Loss for Training Specialized Mixture-of-Expert Models". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vienna, Austria: Association for Computational Linguistics,

2025, pp. 5005–5018. DOI: 10.18653/v1/2025.acl-long.249. URL: https://aclanthology.org/2025.acl-long.249/.

[10] Nabil Omi, Siddhartha Sen, and Ali Farhadi. *Load Balancing Mixture of Experts with Similarity Preserving Routers*. 2025. DOI: 10.48550/arXiv.2506.14038. arXiv: 2506.14038 [cs.LG]. URL: https://arxiv.org/abs/2506.14038.

[11] Zhongyang Li, Ziyue Li, and Tianyi Zhou. "Routing Manifold Alignment Improves Generalization of Mixture-of-Experts LLMs". In: *International Conference on Learning Representations*. 2026. URL: https://openreview.net/forum?id=3lskwxB653.

[12] Tam Minh Nguyen et al. *Improving Routing in Sparse Mixture of Experts with Graph of Tokens*. 2025. arXiv: 2505.00792 [cs.LG]. URL: https://arxiv.org/abs/2505.00792.

[13] Jingming Zhuo et al. "ProSA: Assessing and Understanding the Prompt Sensitivity of LLMs". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Miami, Florida, USA: Association for Computational Linguistics, 2024, pp. 1950–1976. DOI: 10.18653/v1/2024.findings-emnlp.108. URL: https://aclanthology.org/2024.findings-emnlp.108/.

[14] Fuzhao Xue et al. "OpenMoE: An Early Effort on Open Mixture-of-Experts Language Models". In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by Ruslan Salakhutdinov et al. Vol. 235. Proceedings of Machine Learning Research. PMLR, 2024, pp. 55625–55655. URL: https://proceedings.mlr.press/v235/xue24c.html.

[15] Niklas Muennighoff et al. "OLMoE: Open Mixture-of-Experts Language Models". In: *International Conference on Learning Representations*. 2025. URL: https://arxiv.org/abs/2409.02060.

[16] Matthew Lyle Olson et al. "Probing Semantic Routing in Large Mixture-of-Expert Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2025*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 18263–18278. ISBN: 979-8-89176-335-7. DOI: 10.18653/v1/2025.findings-emnlp.991. URL: https://aclanthology.org/2025.findings-emnlp.991/.

[17] Lucas Bandarkar et al. *Multilingual Routing in Mixture-of-Experts*. 2025. arXiv: 2510.04694 [cs.CL]. URL: https://arxiv.org/abs/2510.04694.

[18] Xin He et al. *ExpertFlow: Optimized Expert Activation and Token Allocation for Efficient Mixture-of-Experts Inference*. 2024. arXiv: 2410.17954 [cs.AI]. URL: https://arxiv.org/abs/2410.17954.

[19] Leyang Xue et al. *MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache*. 2024. arXiv: 2401.14361 [cs.LG]. URL: https://arxiv.org/abs/2401.14361.

[20] Elias Frantar and Dan Alistarh. *QMoE: Practical Sub-1-Bit Compression of Trillion-Parameter Models*. 2023. arXiv: 2310.16795 [cs.LG]. URL: https://arxiv.org/abs/2310.16795.

[21] Jack Cashman and Jiaqi Nie. *Tiny-QMoE*. 2025. arXiv: 2509.22951 [cs.PF]. URL: https://arxiv.org/abs/2509.22951.

[22] Weilin Cai et al. *A Survey on Mixture of Experts*. 2024. arXiv: 2407.06204 [cs.LG]. URL: https://arxiv.org/abs/2407.06204.

[23] Siyuan Mu and Sen Lin. *A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications*. 2025. arXiv: 2503.07137 [cs.LG]. URL: https://arxiv.org/abs/2503.07137.

[24] Nan Du et al. "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 5547–5569. URL: https://proceedings.mlr.press/v162/du22c.html.

[25] Albert Q. Jiang et al. *Mixtral of Experts*. 2024. arXiv: 2401.04088 [cs.LG]. URL: https://arxiv.org/abs/2401.04088.

[26] Aran Komatsuzaki et al. "Sparse Upcycling: Training Mixture-of-Experts from Dense Checkpoints". In: *International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=T5nUQDrM4u.

[27] Shengzhuang Chen, Ying Wei, and Jonathan Richard Schwarz. "Automatic Expert Discovery in LLM Upcycling via Sparse Interpolated Mixture-of-Experts". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Wanxiang Che et al. Vienna, Austria: Association for Computational Linguistics, July 2025, pp. 16703–16717. ISBN: 979-8-89176-251-0. DOI: 10.18653/v1/2025.acl-long.816. URL: https://aclanthology.org/2025.acl-long.816/.

[28] Minxuan Lv et al. "DSMoE: Matrix-Partitioned Experts with Dynamic Routing for Computation-Efficient Dense LLMs". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025. DOI: 10.18653/v1/2025.emnlp-main.997. URL: https://aclanthology.org/2025.emnlp-main.997/.

[29] Huy Nguyen et al. *Statistical Perspective of Top-K Sparse Softmax Gating Mixture of Experts*. 2023. arXiv: 2309.13850 [stat.ML]. URL: https://arxiv.org/abs/2309.13850.

[30] Trevor Gale et al. "MegaBlocks: Efficient Sparse Training with Mixture-of-Experts". In: *Proceedings of Machine Learning and Systems*. Vol. 5. 2023. URL: https://proceedings.mlsys.org/paper_files/paper/2023/hash/5a54f79333768effe7e8927bcccf1Abstract-mlsys2023.html.

[31] Lean Wang et al. *Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts*. 2024. arXiv: 2408.15664 [cs.LG]. URL: https://arxiv.org/abs/2408.15664.

[32] Ashwinee Panda et al. "Dense Backpropagation Improves Training for Sparse Mixture-of-Experts". In: *Advances in Neural Information Processing Systems*. 2025. URL: https://openreview.net/forum?id=jTUIo4Kl6u.

[33] Feng Yao et al. *DenseMixer: Improving MoE Post-Training with Precise Router Gradient*. SPIGM @ NeurIPS. Sept. 2025. URL: https://openreview.net/forum?id=88PI0SpVL3.

[34] Bowen Dong et al. "Maximum Score Routing For Mixture-of-Experts". In: *Findings of the Association for Computational Linguistics: ACL 2025*. Ed. by Wanxiang Che et al. Vienna, Austria: Association for Computational Linguistics, July 2025, pp. 12619–12632. ISBN: 979-8-89176-256-5. DOI: 10.18653/v1/2025.findings-acl.653. URL: https://aclanthology.org/2025.findings-acl.653/.

[35] Mike Lewis et al. *BASE Layers: Simplifying Training of Large, Sparse Models*. 2021. arXiv: 2103.16716 [cs.LG]. URL: https://arxiv.org/abs/2103.16716.

[36] Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. *Hash Layers For Large Sparse Models*. 2021. arXiv: 2106.04426 [cs.LG]. URL: https://arxiv.org/abs/2106.04426.

[37] Yilun Liu et al. *Routing-Free Mixture-of-Experts*. 2026. arXiv: 2604.00801 [cs.LG]. URL: https://arxiv.org/abs/2604.00801.

[38] Hussein Hazimeh et al. "DSelect-k: Differentiable Selection in the Mixture of Experts with Applications to Multi-Task Learning". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021, pp. 29335–29347. URL: https://arxiv.org/abs/2106.03760.

[39] Ziteng Wang, Jun Zhu, and Jianfei Chen. "ReMoE: Fully Differentiable Mixture-of-Experts with ReLU Routing". In: *International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=4D0f16Vwc3.

[40] Amirhossein Vahidi et al. *DirMoE: Dirichlet-routed Mixture of Experts*. 2026. arXiv: 2602.09001 [cs.LG]. URL: https://arxiv.org/abs/2602.09001.

[41] Joan Puigcerver et al. *From Sparse to Soft Mixture of Experts*. 2024. arXiv: 2308.00951 [cs.LG]. URL: https://arxiv.org/abs/2308.00951.

[42] Damai Dai et al. *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*. 2024. arXiv: 2401.06066 [cs.CL]. URL: https://arxiv.org/abs/2401.06066.

[43] Zhenpeng Su et al. "CartesianMoE: Boosting Knowledge Sharing among Experts via Cartesian Product Routing in Mixture-of-Experts". In: *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 10040–10055. DOI: 10.18653/v1/2025.naacl-long.505. URL: https://aclanthology.org/2025.naacl-long.505/.

[44] Sugyeong Eo et al. "Mixture-of-Clustered-Experts: Advancing Expert Specialization and Generalization in Instruction Tuning". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025. DOI: 10.18653/v1/2025.emnlp-main.718. URL: https://aclanthology.org/2025.emnlp-main.718/.

[45] An Wang et al. "HMoE: Heterogeneous Mixture of Experts for Language Modeling". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 21943–21957. ISBN: 979-8-89176-332-6. DOI: 10.18653/v1/2025.emnlp-main.1115. URL: https://aclanthology.org/2025.emnlp-main.1115/.

[46] Giang Do, Hung Le, and Truyen Tran. "SimSMoE: Toward Efficient Training Mixture of Experts via Solving Representational Collapse". In: *Findings of the Association for Computational Linguistics: NAACL 2025*. Association for Computational Linguistics, 2025. URL: https://aclanthology.org/2025.findings-naacl.107/.

[47] Jun Bai et al. "Understanding and Leveraging the Expert Specialization of Context Faithfulness in Mixture-of-Experts LLMs". In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Ed. by Christos Christodoulopoulos et al. Suzhou, China: Association for Computational Linguistics, Nov. 2025. DOI: 10.18653/v1/2025.emnlp-main.1114. URL: https://aclanthology.org/2025.emnlp-main.1114/.

[48] Suchin Gururangan et al. *DEMix Layers: Disentangling Domains for Modular Language Modeling*. 2021. arXiv: 2108.05036 [cs.CL]. URL: https://arxiv.org/abs/2108.05036.

[49] Changho Hwang et al. "Tutel: Adaptive Mixture-of-Experts at Scale". In: *Proceedings of Machine Learning and Systems*. Vol. 5. 2023. URL: https://

proceedings.mlsys.org/paper_files/paper/2023/hash/5616d34cf8ff73942cfd5aa922842556-Abstract-mlsys2023.html.

[50] Xudong Lu et al. *Not All Experts are Equal: Efficient Expert Pruning and Skipping for Mixture-of-Experts Large Language Models*. 2024. arXiv: 2402.14800 [cs.CL]. URL: https://arxiv.org/abs/2402.14800.

[51] Zhanhao Xie et al. "Automated Fine-Grained Mixture-of-Experts Quantization". In: *Findings of the Association for Computational Linguistics: ACL 2025*. Ed. by Wanxiang Che et al. Vienna, Austria: Association for Computational Linguistics, July 2025, pp. 27024–27037. ISBN: 979-8-89176-256-5. DOI: 10.18653/v1/2025.findings-acl.1386. URL: https://aclanthology.org/2025.findings-acl.1386/.

[52] Ahmad Chamma, Omar El Herraoui, and Guokan Shang. *MixtureKit: A General Framework for Composing, Training, and Visualizing Mixture-of-Experts Models*. Submitted to ICLR 2026. 2025. URL: https://openreview.net/forum?id=bVyCzNDNX8.

[53] Sieun Hyeon and Jaeyoung Do. *Is Retraining-Free Enough? The Necessity of Router Calibration for Efficient MoE Compression*. 2026. arXiv: 2603.02217 [cs.LG]. URL: https://arxiv.org/abs/2603.02217.

[54] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

[55] Shih-Yang Liu et al. "DoRA: Weight-Decomposed Low-Rank Adaptation". In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by Ruslan Salakhutdinov et al. Vol. 235. Proceedings of Machine Learning Research. PMLR, 2024, pp. 32100–32121. URL: https://proceedings.mlr.press/v235/liu24bn.html.

[56] Xun Wu, Shaohan Huang, and Furu Wei. *Mixture of LoRA Experts*. 2024. arXiv: 2404.13628 [cs.CL]. URL: https://arxiv.org/abs/2404.13628.

[57] Shaoxiang Chen, Zequn Jie, and Lin Ma. *LLaVA-MoLE: Sparse Mixture of LoRA Experts for Mitigating Data Conflicts in Instruction Finetuning MLLMs*. 2024. arXiv: 2401.16160 [cs.CV]. URL: https://arxiv.org/abs/2401.16160.

[58] Xinrong Chen et al. "GuiLoMo: Allocating Experts and Ranks for LoRA-MoE via Bilevel Optimization with GuidedSelection Vectors". In: *Findings of the Association for Computational Linguistics: EMNLP 2025*. Suzhou, China: Association for Computational Linguistics, 2025, pp. 7552–7567. DOI: 10.18653/v1/2025.findings-emnlp.399. URL: https://aclanthology.org/2025.findings-emnlp.399/.

[59] Yuan Zhuang et al. *LD-MoLE: Learnable Dynamic Routing for Mixture of LoRA Experts*. 2025. arXiv: 2509.25684 [cs.LG]. URL: https://arxiv.org/abs/2509.25684.

[60] Leo Gao et al. *A Framework for Few-Shot Language Model Evaluation*. 2023. arXiv: 2305.14782 [cs.CL]. URL: https://arxiv.org/abs/2305.14782.

[61] Dan Hendrycks et al. "Measuring Massive Multitask Language Understanding". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=d7KBjmI3GmQ.

[62] Peter Clark et al. *Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*. 2018. arXiv: 1803.05457 [cs.AI]. URL: https://arxiv.org/abs/1803.05457.

[63] Rowan Zellers et al. "HellaSwag: Can a Machine Really Finish Your Sentence?" In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 4791–4800. DOI: 10.18653/v1/P19-1472. URL: https://aclanthology.org/P19-1472/.

[64] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG]. URL: https://arxiv.org/abs/2110.14168.

[65] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: https://arxiv.org/abs/2107.03374.

[66] Boxin Wang et al. "Adversarial GLUE: A Multi-Task Benchmark for Robustness Evaluation of Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021. URL: https://openreview.net/forum?id=GF9cSKI3A_q.

[67] Stephanie Lin, Jacob Hilton, and Owain Evans. "TruthfulQA: Measuring How Models Mimic Human Falsehoods". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, 2022, pp. 3214–3252. DOI: 10.18653/v1/2022.acl-long.229. URL: https://aclanthology.org/2022.acl-long.229/.

[68] Melanie Sclar et al. "Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I Learned to Start Worrying about Prompt Formatting". In: *International Conference on Learning Representations*. 2024.