International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

MoSLM: Sparse Orchestration of Specialized Small Language Models

Damodhar Meshram[1], Manas Tarare[2] , Samruddhi Tonghale[3]

[1,2,3]Morningmind technolab private limited ,India

Abstract

This proposal presents MoSLM (Mixture of Small Language Models), a task-level orchestration system inspired by Mixture-of-Experts routing. A small orchestrator (1–3B parameters) decomposes requests into typed subtasks and routes each to suitable small expert models for math, code, retrieval QA, tool calling, and general chat; low-confidence subtasks can be escalated to a larger model.

The proposed orchestrator training has two stages: supervised routing from oracle labels, followed by reinforcement learning (GRPO) with a reward that combines correctness, serving cost, latency, and a regularizer intended to discourage expert collapse. Capability embeddings will be used to test whether new experts can be added without retraining the orchestrator.

The study will test the hypothesis that MoSLM can match or beat a monolithic 8–14B model on mixed reasoning, coding, and tool-use benchmarks while reducing measured serving cost per successful query by 40–60%. This is a target, not a result; cost will include orchestration, retrieval, model loading, and fallback. If completed, planned artifacts include the code, router weights, and MixBench.

Keywords: Small language models; LLM orchestration; task-level routing; expert models; cost-aware inference; reinforcement learning; tool use; retrieval-augmented question answering.

Introduction and motivation

Most agent workloads are many small, narrow calls, and sending every one to a large general model wastes compute. Belcak et al. (NVIDIA, 2025) argue that SLMs are powerful enough, better suited and cheaper for most agentic invocations, with heterogeneous multi-model systems as the natural design (arXiv:2506.02153). Specialised small models already rival much larger ones on the tasks they were tuned for (HierRouter).

Mixture-of-Experts (Switch, Mixtral, DeepSeekMoE) gets efficiency by activating a few expert sub-networks per token. That routing is buried inside one model: experts cannot be swapped, inspected, served on different hardware or trained separately. Request-level routers do the opposite but stay shallow: most make one upfront choice between two models, which limits them on complex queries that need several steps (HierRouter).

The gap this proposal targets is a router that behaves like an MoE gate but works on whole subtasks:

- **Sparse and learned**, like an MoE gate (top-k experts, not all of them)
- **Compositional**: it decomposes a request, so one query can use the math expert and then the code expert

- Balanced: a load-balancing term is intended to discourage router collapse onto one expert; its effect on task quality will be measured
- Open-pool: the study tests whether capability embeddings allow experts to be added without retraining the orchestrator
- Single-GPU deployment target: inference on one 24 GB GPU under the quantization and model-swapping plan below; actual fit and latency will be measured

Related work

This proposal investigates whether combining subtask decomposition, sparse top-k routing, a load-balancing regularizer, and an expert registry improves the quality–cost trade-off. The related-work comparison is preliminary; novelty is a question to test against published papers and current preprints, not an established "first" claim.

Table 1.1 Related work

Work	Routing unit	Expert pool	How the router learns	Potential distinction to test
Token-level MoE (Switch, Mixtral, DeepSeekMoE)	Each token, inside one model	FFN sub-networks	Jointly with the model + balance loss	Experts not swappable or separately trainable
RouteLLM (ICLR 2025)	Whole query	One strong + one weak LLM	Preference data (matrix factorisation, BERT, causal-LM classifiers)	Binary choice; no decomposition; >2x cost saving
FrugalGPT, AutoMix (cascades)	Whole query, escalating	Small → large LLMs	Thresholds / self-verification	Later models redo the query from scratch
HuggingGPT	Subtasks	Hugging Face models	Prompted LLM controller, no training	Large controller; not cost-aware
HierRouter (2025)	Fixed number of hops	3 SLMs (1.5–3.8B)	PPO; reward = quality − α·cost	No decomposition or balance term; 300 examples per task; higher cost than single SLMs
ToolOrchestra / Orchestrator-8B (NVIDIA, 2025)	Multi-turn tool calls	Tools + frontier LLMs (GPT-5 etc.)	GRPO with outcome, efficiency and preference rewards	Relies on paid frontier APIs; 8B orchestrator
CALM (2026)	Whole query	Fleet of SLMs with text descriptions	Self-adaptive MAPE-K loop for QoS	Systems focus; no learned decomposition
OrchSLM (2026 preprint)	Cached candidate answers	Heterogeneous SLMs	Non-interactive selection / voting	Every SLM runs on every query

ToolOrchestra reports 37.1% on Humanity's Last Exam versus 35.1% for GPT-5 at about 30% of the cost in its own evaluation. Because its model pool and evaluation setup differ, this is contextual evidence rather than a directly comparable result. This study will test whether a similar orchestration recipe transfers to smaller locally served experts.

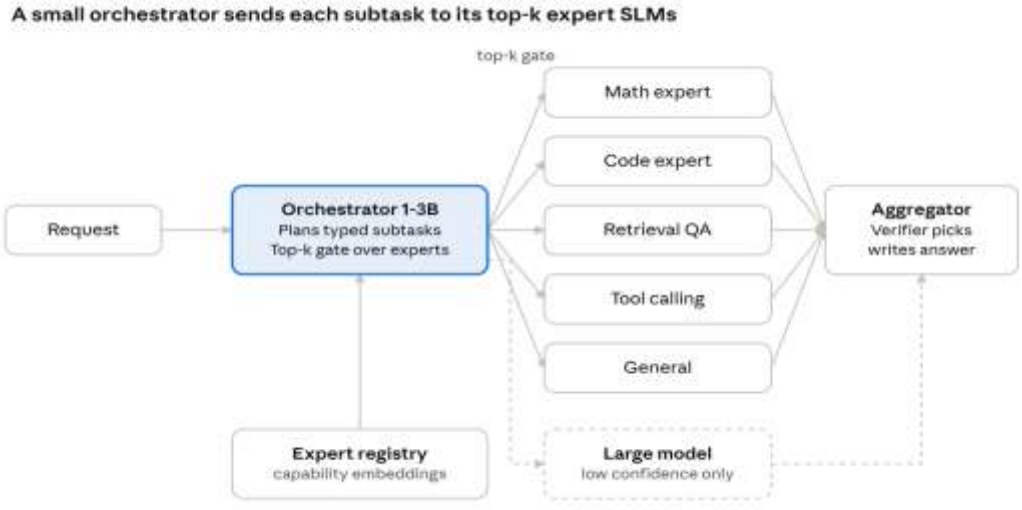
Research questions and hypotheses

#	Research question	Hypothesis (what would confirm it)
RQ1	Can a 1–3B orchestrator over 1–4B experts match a monolithic 8–14B model on mixed tasks?	MoSLM accuracy within 2 points of, or above, Qwen2.5-14B on the mixed suite at ≤ 50% of its FLOPs
RQ2	Does task decomposition beat whole-query routing?	Decomposition adds ≥ 5 points on multi-skill queries (math + code, retrieve + reason) and does not hurt single-skill ones
RQ3	Does an MoE-style load-balancing term prevent router collapse?	Without it, one expert takes > 60% of calls and accuracy drops; with it, expert-usage entropy stays near the oracle's
RQ4	Can a new expert join without retraining the orchestrator?	Adding an unseen expert via its capability embedding recovers ≥ 80% of the gain of full retraining
RQ5	When should the system escalate to a large model?	Confidence-gated escalation on ≤ 10% of queries closes most of the remaining gap to the large model

Proposed architecture

MoSLM has four parts: an orchestrator that plans and gates, a registry of expert SLMs, an aggregator that merges results, and a large-model fallback used only on low confidence.

Fig 1.1 Architecture of the proposed methodology



MoSLM architecture · orchestrator, 5 experts, aggregator, fallback

Solid arrows are the normal path; dashed arrows run only when confidence is low and a subtask escalates to the large model.

1. **Orchestrator (1–3B, e.g. Qwen2.5-1.5B or Llama-3.2-3B).** It writes a short plan of typed subtasks, each with a skill tag and its inputs. For each subtask it scores every expert with a gate over learned capability embeddings, like an MoE gate but over models:
   $p(e \mid s) = \operatorname{softmax}_e\!\left(\frac{\mathbf{h}_s^{\top} W \mathbf{c}_e}{\tau} \right), \qquad \mathcal{E}_s = \operatorname{TopK}_e\, p(e \mid s)$
2. Here h_s is the orchestrator's hidden state for subtask s, c_e is expert e's capability embedding, and k is 1 by default (2 for low-margin decisions).
3. **Expert registry.** Each expert has a capability card: a short text description, a learned embedding c_e, its cost per token and its latency. Experts are either separate SLMs (heterogeneous pool) or LoRA adapters on one shared base model (served together, as in S-LoRA), which is compared as an ablation. A new expert's embedding is initialised from its card and a few hundred probe questions.
4. **Aggregator.** Subtask outputs are passed forward as context to later subtasks; a final step by the orchestrator writes the answer. With k = 2, the aggregator picks between candidates using the orchestrator's own verifier head.
5. **Escalation.** If the gate's top probability or the verifier score falls below a threshold, the subtask goes to a large model (local 14B or an API). The threshold sets the cost–quality trade-off, as α does in RouteLLM.

Training objective. Stage 1 trains the planner and gate with supervised labels: for each training subtask, which experts actually solve it (an oracle built by running every expert). Stage 2 fine-tunes with GRPO on the episode reward

$$R = \mathbb{1}[\text{correct}] \; - \; \lambda_c \, C_{\text{tokens}} \; - \; \lambda_\ell \, L_{\text{latency}} \; - \; \lambda_b \, N \sum_{i=1}^{N} f_i \, P_i$$

The last term is the Switch Transformer load-balancing loss, with f_i the share of subtasks sent to expert i and P_i its mean gate probability over a batch.

Methodology

This proposed study evaluates five expert roles—math, code, retrieval QA, tool calling, and general chat—using the fixed checkpoints in the expert table. To construct the routing oracle, it runs every expert on each training subtask and scores outputs with task-specific criteria: exact match or F1 for math and QA, executable tests for code, function selection, schema validity, and successful execution for tool calling, and benchmark accuracy or a fixed rubric for general-chat items. Oracle labels record every expert that passes its criterion; the top-1 target is the least-cost successful expert, with latency breaking ties. Test questions and labels are excluded from oracle construction and router training.

The orchestrator is trained in two stages: supervised learning fits the planner and gate to the oracle labels, then GRPO optimizes correctness, serving cost, latency, and load balancing. Evaluation compares the routed system with six baseline groups: each expert alone and the oracle; monolithic 8B and 14B models; random and k-nearest-neighbor routers; a RouteLLM-style N-way classifier; HierRouter; and a prompted orchestrator. Expert checkpoints stay frozen in the primary experiment to isolate routing gains. Any expert LoRA fine-tuning is a separate ablation, uses training data only, and reports the same task-specific metrics.

Expert pool (all ≤ 4B, open weights)

Table 1.2 Model selection and usage

Expert	Candidate model checkpoint	Skill
Math	Qwen2.5-Math-1.5B-Instruct	Arithmetic and symbolic reasoning
Code	Qwen2.5-Coder-3B-Instruct	Program synthesis and repair
Retrieval QA	Llama-3.2-3B-Instruct + Qwen3-Embedding-0.6B + Qwen3-Reranker-0.6B	Dense retrieval, passage reranking, grounded factual and multi-hop QA
Tool calling	Qwen3-4B-Instruct-2507 (zero-shot primary candidate)	Function selection, schema-valid JSON arguments, and tool execution
General	Phi-3.5-mini-instruct	Instructions, writing, fallback

Retrieval pipeline. Use Llama-3.2-3B-Instruct as the answer generator, Qwen3-Embedding-0.6B for dense retrieval, and Qwen3-Reranker-0.6B for reranking. Retrieve the 20 highest cosine-similarity passages, rerank them, and pass the top 5 with source IDs to the generator. For HotpotQA, use the official distractor paragraphs as the candidate corpus. For MixBench, freeze and release a dated source-corpus snapshot; do not index test answers or labels. Split longer MixBench documents into 512-token chunks with 64-token overlap. Keep both retrieval models frozen in the primary experiment. Record corpus checksums, checkpoint revisions, tokenizer versions, chunking, vector-index implementation, and search settings.

Tool-calling model. Use Qwen3-4B-Instruct-2507 as the fixed tool-calling candidate. The primary experiment evaluates the checkpoint without fine-tuning, using a frozen system prompt and JSON tool schemas. Keep tool descriptions and outputs consistent across models. Report function-selection accuracy, JSON/schema validity before repair, argument match, and end-to-end execution success on BFCL v3. Any LoRA tuning is a separate ablation using training data only; release the data identifiers, split, prompt template, and adapter settings.

24 GB single-GPU inference plan. This plan applies to inference; expert training and GRPO use the larger resources listed in the timeline. Quantize the orchestrator and autoregressive experts to 4-bit AWQ, cap input context at 8,192 tokens, cap generation at 4,096 tokens, and use batch size 1 for the primary latency and memory runs. Keep the orchestrator and only the selected generative expert resident on the GPU. Store other checkpoints on local SSD, load an expert when selected, and evict it before loading another. Run the embedding model and reranker sequentially; offload them to CPU if GPU headroom is insufficient. Record exact checkpoint revisions and quantization settings, peak VRAM, and warm and cold model-loading latency. Compare quantized outputs with BF16 on a fixed validation slice; include the quantized configuration on the main cost–quality curve only if quality stays within two absolute points of BF16. If a checkpoint lacks compatible 4-bit support or exceeds that quality margin, use a shared base with LoRA adapters or report it separately rather than claiming it fits the 24 GB configuration.

Data. Training and test sets mix single-skill and multi-skill queries: GSM8K and MATH-500 (math), HumanEval+ and MBPP (code), HotpotQA (multi-hop QA), BFCL (function calling), MMLU and ARC (general). We also build **MixBench**, about 2,000 queries that need two or more skills (for example, "compute X, then write code that checks it"), to test decomposition. Splits are fixed, and no test question is used to build the oracle.

Baselines

1. Each expert alone, and an oracle router (upper bound: best expert per query)
2. Monolithic Llama-3.1-8B-Instruct and Qwen2.5-14B-Instruct
3. Random router and a k-nearest-neighbour embedding router
4. RouteLLM-style classifier router, extended to N experts
5. HierRouter (PPO, multi-hop)
6. Prompted orchestrator with no training (HuggingGPT-style)

Metrics. Report per-task quality (exact match/F1 for math and QA, pass@1 using executable tests for code, BFCL v3 tool-call and execution scores, and answer-support accuracy for retrieval), retrieval Recall@20 and MRR@5, total serving cost per successful query (including router, embedder, reranker, model loading, retries, and fallback) as raw GPU seconds/tokens and currency under stated rates, p50 and p95 latency, peak GPU memory, routing accuracy against the oracle, expert-usage entropy, and area under the cost–quality curve (APGR, as in RouteLLM).

Ablations. No decomposition; top-1 vs top-2 gating; no load-balancing term; separate SLMs vs LoRA experts on one base; orchestrator at 0.5B, 1.5B and 3B; adding an unseen expert with and without retraining; escalation thresholds from 0% to 20% of queries.

Statistics. Three seeds for every trained component; 95% bootstrap confidence intervals on all accuracy numbers.

Expected contributions and provisional novelty

1. Potential contribution — task-level sparse gating: the study will test whether top-k routing over expert models with a load-balancing regularizer improves task-level orchestration. It does not claim to be the first such system; novelty will be assessed after a complete literature review.
2. Potential contribution — decompose then route: evaluate whether one request can use several experts in sequence and whether this improves multi-skill tasks over whole-query routing.
3. Potential contribution — open expert pool: test whether capability embeddings support adding or removing experts without retraining the orchestrator.
4. Deployment objective: test whether the orchestrator and expert pool can meet the one 24 GB GPU inference target under the specified quantization and swapping configuration.
5. Planned artifact: if completed, release MixBench, a multi-skill benchmark with per-subtask oracle labels, together with the code and router weights.

The novelty claim remains provisional. The comparison table is an initial map of related work, organized around each study's routing unit, expert pool, learning method, decomposition strategy, load-balancing approach, and support for adding experts. It is intended to identify questions for further review, not to establish that prior work lacks any of these features.

MoSLM's provisional contribution is to study these components as one task-level orchestration system: request decomposition, sparse routing to separately served specialists, cost and latency awareness, load balancing, and expert addition through capability descriptions

Timeline, resources and risks

The project runs 26 weeks in six phases, with a go/no-go check at week 8.



Project timeline and resources

The proposed study will run for 26 weeks across six phases. **Phase 1 (Weeks 1–3):** finalize datasets, evaluation metrics, data splits, and the inference and logging setup. **Phase 2 (Weeks 4–8):** implement the five expert roles, evaluate each expert on the training queries, and build the oracle-routing data. **Phase 3 (Weeks 9–13):** train the orchestrator in two stages, first using supervised oracle labels and then with GRPO. **Phase 4 (Weeks 14–17):** integrate routing, retrieval, and tool execution, and optimize inference for the available GPUs. **Phase 5 (Weeks 18–23):** compare against the six baseline groups and run planned ablations. **Phase 6 (Weeks 24–26):** analyze results, document limitations, and complete the manuscript and reproducibility materials.

Week-8 gate. The oracle router must score at least five points above the best single expert on the preselected validation metric. Define the scoring scale in advance; on a 0–100 scale, this means five percentage points. If it does not pass, revise the expert pool and reevaluate before training the learned router. The 26-week schedule assumes the gate is passed.

Compute and software. Budget approximately 600–900 GPU-hours, mainly for oracle construction and GRPO rollouts. The target is one A100 80 GB GPU or two 24 GB GPUs. Since two cards do not automatically combine their memory, use quantization, small batches, limited context, and model swapping as needed. Free Kaggle or Colab tiers are suitable for prototyping the 0.5B orchestrator, but not assumed for the full run. Use vLLM for expert serving, TRL or verl for GRPO, PEFT/LoRA for the separate fine-tuning ablation, lm-evaluation-harness for benchmarks, and BFCL for tool-calling evaluation.

For reproducibility, record the model checkpoints, software versions, prompts, decoding settings, and hardware configuration used in each phase. Track GPU-hours, peak memory, latency, token use, and any model-swapping overhead alongside task scores. This will make it possible to compare quality and operating cost across MoSLM, the single-expert systems, and the planned baselines. The primary evaluation will use frozen expert checkpoints, with any LoRA-tuned experts reported separately as an ablation. A small pilot will confirm that the planned workload fits the available GPU memory and compute budget before the full run begins.

Risks and fallbacks

Table 1.3 Risks and fallbacks

Risk	Likelihood	Fallback
Experts too similar, so routing gains are small	Medium	Week-8 check: if the oracle router is not ≥ 5 points above the best single expert, swap in more specialised experts
GRPO unstable with a small orchestrator	Medium	Keep the Stage 1 supervised router as the main result; report RL as an ablation
Decomposition errors cascade	Medium	Cap plans at 3 subtasks; let the verifier trigger re-planning once
Cost of building the oracle	Low	Sub-sample to 300–500 queries per task, as HierRouter did
Overlap with new 2026–27 papers	High	Re-check arXiv monthly; sharpen the claim around the balance loss and open pool

References

The literature review is provisional. Publication status and model-card revisions will be checked and pinned before the experiments.

1. Belcak, P. et al. (2025). Small Language Models are the Future of Agentic AI. arXiv:2506.02153.
2. Su, H. et al. (2025). ToolOrchestra: Elevating Intelligence via Efficient Model and Tool Orchestration. NVIDIA & HKU, arXiv:2511.21689.
3. Gupta, N., Guo, B., Kannan, R., Prasanna, V. (2025). HierRouter: Coordinated Routing of Specialized LLMs via Reinforcement Learning. arXiv:2511.09873.
4. Ong, I. et al. (2025). RouteLLM: Learning to Route LLMs with Preference Data. ICLR 2025.
5. Zhang, C., and Yao, Y. (2026). OrchSLM: Probing the Dynamics of Small Language Model Orchestration. arXiv:2609.13470. https://arxiv.org/abs/2609.13470.
6. Jain, H., Pandey, D., and Vaidhyanathan, K. (2026). CALM: A Self-Adaptive Orchestration Approach for QoS-Aware Routing in Small Language Model-based Systems. SEAMS 2026. DOI: 10.1145/3788550.3794880. https://arxiv.org/abs/2602.03632.
7. Saini, M. (2025). Towards Resource-Efficient Multimodal Intelligence: Learned Routing among Specialized Expert Models. arXiv:2511.06441.
8. Shazeer, N. et al. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. ICLR.
9. Fedus, W., Zoph, B., Shazeer, N. (2022). Switch Transformers. JMLR.
10. Jiang, A. et al. (2024). Mixtral of Experts. arXiv:2401.04088.
11. Dai, D. et al. (2024). DeepSeekMoE. ACL.
12. Chen, L., Zaharia, M., Zou, J. (2024). FrugalGPT. TMLR.
13. Aggarwal, P. et al. (2024). AutoMix: Automatically Mixing Language Models. NeurIPS.
14. Shen, Y. et al. (2023). HuggingGPT. NeurIPS.
15. Sheng, Y. et al. (2024). S-LoRA: Serving Thousands of Concurrent LoRA Adapters. MLSys.
16. Shao, Z. et al. (2024). DeepSeekMath (introduces GRPO). arXiv:2402.03300.

17. Qwen Team (2025). Qwen3: Think Deeper, Act Faster. arXiv:2505.09388. Checkpoint: Qwen/Qwen3-4B-Instruct-2507. Model card: https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507 (accessed 2 November 2026).
18. Zhang, Y. et al. (2025). Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. arXiv:2506.05176. Checkpoints: Qwen/Qwen3-Embedding-0.6B and Qwen/Qwen3-Reranker-0.6B. Model cards: https://huggingface.co/Qwen/Qwen3-Embedding-0.6B and https://huggingface.co/Qwen/Qwen3-Reranker-0.6B (accessed 2 November 2026).
19. Qwen Team (2024). Qwen2.5-Math-1.5B-Instruct model card. https://huggingface.co/Qwen/Qwen2.5-Math-1.5B-Instruct (accessed 2 November 2026).
20. Qwen Team (2024). Qwen2.5-Coder-3B-Instruct model card. https://huggingface.co/Qwen/Qwen2.5-Coder-3B-Instruct (accessed 2 November 2026).
21. Meta AI (2024). Llama-3.2-3B-Instruct model card. https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct (accessed 2 November 2026).