SABHA: SPARSE LATENT COORDINATION OF SPECIALIST LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce **Sabha**, a modular language model that coordinates separately pretrained domain specialists through a compact learned interface. Each specialist, or *island*, is a dense transformer. A router selects a subset of islands, which emit pooled hidden-state messages into a shared latent bus; a Council decoder uses these messages to generate the system's output. The architecture separates specialist transformer blocks from the coordination interface and does not require one pretrained island to serve as the output hub. With three 234M-parameter islands and a 25M-parameter Council, the top-2 configuration achieves aggregate multitask cross-entropy of 7.708, compared with 7.721 for a 729M dense baseline and 7.873 for our Branch-Train-Stitch implementation. Its estimated inference FLOPs per token are 74.5% of the dense baseline's. An individual code-island update reduces system-level code loss by 0.12, with a 0.01 increase on reasoning and a 0.29 decrease on general text. These findings support sparse whole-model coordination as a practical design point for modular language modeling. They establish neither a general advantage over dense models nor behavioral isolation under arbitrary updates: the quality margin is small, routing underuses one specialist, and the current sequential implementation is slower despite its lower estimated arithmetic cost.

1 INTRODUCTION

Dense language models organize broad domain knowledge within a single, coupled transformer backbone. Although a task may need only part of that knowledge, execution still traverses the dense backbone rather than selecting a domain-specific model. This motivates a different organization: build capability from smaller specialists that can be trained and maintained separately, and activate only a subset for each query. The aim is not simply to shrink a large model, but to separate total model capacity, per-query computation, and the unit of specialist maintenance.

Sparse mixture-of-experts (MoE) models already separate total parameter count from active computation by selecting expert layers within a shared architecture (Shazeer et al., 2017; Fedus et al., 2022). Separately trained expert language models move the modular boundary to the whole backbone (Li et al., 2022). The remaining challenge is coordination: how can a small interface combine these specialists without executing all of them or relying on another full-sized model as the output hub? We ask whether *sparse whole-model activation and compact latent communication can retain competitive aggregate predictive quality*.

This paper proposes a design using separately pretrained specialist *islands* and a learned *Council* (Sabha). In the evaluated configuration, three islands specialize in general text, mathematical reasoning, and code. The Council routes each query to a subset, projects pooled specialist representations into a shared latent bus, and decodes the output (Figure 1). The islands retain separate transformer blocks; no pretrained island supplies the full output-generating hub. The Council is a central coordinator, and the prototype uses the generalist's first layer for routing features. Thus the design separates specialist backbones without claiming complete symmetry or the absence of all shared computation.

The distinction from prior work is architectural, not the invention of modular experts. Branch-Train-Merge combines independent experts, CALM learns cross-attention between models, and Branch-Train-Stitch (BTS) connects specialists through a pretrained hub (Li et al., 2022; Bansal et al., 2024; Zhang et al., 2025). Sabha instead studies a bounded message interface between selected

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

whole-model specialists and a separate compact decoder. This places the coordination burden on a small learned interface rather than on a privileged specialist backbone.

The top-2 system achieves multitask loss 7.708 versus 7.721 for the dense baseline, with approximately one quarter less estimated inference computation. The supported result is a quality–arithmetic tradeoff: the aggregate margin is small, and dense remains better on general-text and code evaluations. The current sequential implementation is also slower despite its lower FLOPs. These results motivate the modular design without establishing a general quality advantage or a serving-speed improvement.

The study contributes a concrete sparse whole-model coordination architecture, a comparison with dense and BTS-style baselines at similar reported sizes, and analyses of the arbiter, routing behavior, and a specialist update. The update analysis separates *parameter-local maintenance* from *behavioral isolation*: unchanged specialist weights do not guarantee unchanged composed outputs. Together, these results characterize an alternative to monolithic model organization at the tested scale, while leaving larger-scale performance and deployment efficiency open.

2 Related Work

Conditional computation and domain experts. Sparse MoE layers condition computation on the input while retaining shared model components (Shazeer et al., 2017; Fedus et al., 2022). DEMix introduces domain-specific feedforward modules and supports flexible domain mixing (Gururangan et al., 2022). Sabha instead places the expert boundary around a full transformer backbone. This changes the unit of activation and maintenance; it does not imply that layer-level experts are incapable of useful specialization. We do not evaluate a sparse-MoE baseline, and make no empirical claim of superiority over that class.

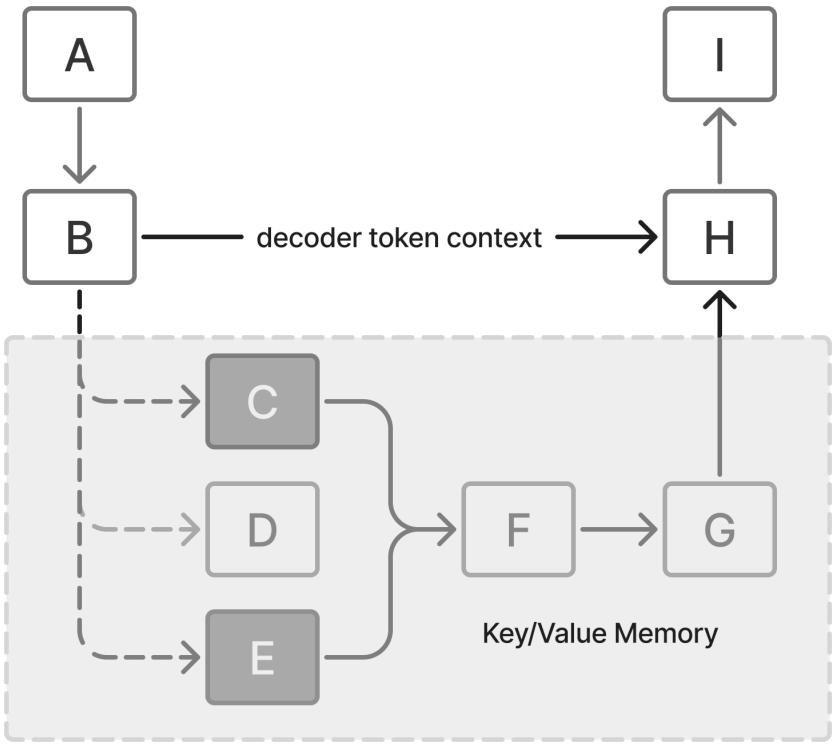
The Branch-Train family. Branch-Train-Merge (BTM) trains separate expert language models that can be combined through ensembling or parameter averaging (Li et al., 2022). Branch-Train-MiX (BTX) uses separately specialized models to initialize MoE feedforward experts, averages other parameters, and then learns token-level routing (Sukhbaatar et al., 2024). BTS retains constituent models and trains stitch layers between experts and a seed language model (Zhang et al., 2025). BTS therefore also supports modular expert addition and removal. Sabha's distinction is not the invention of modular experts, but a separate small output decoder operating on pooled messages from selected whole-model experts. Our BTS comparison concerns the implementation described in Section 4, not the best achievable BTS configuration.

Learned composition interfaces. CALM connects existing language models through cross-attention while keeping their original weights intact (Bansal et al., 2024). More recently, Armstrong et al. (2026) construct feedforward graphs of frozen language models with learned latent projections and a lightweight cross-attention output node. These are close precedents for learned inter-model interfaces. Sabha studies a simpler, single-stage specialist-to-decoder topology with query-level subset activation and domain-specialist pretraining. Its contribution lies in this combination and the observed quality–cost tradeoff, not in latent cross-attention alone.

Latent communication between agents. LatentMAS investigates training-free latent collaboration (Zou et al., 2025), whereas Interlat learns compressed latent communication between agents (Du et al., 2025). Sabha is a single composed prediction system with a trained bus interface, rather than a protocol for exchanging generated natural language between agents. Learning the receiving interface makes representation alignment part of the optimization problem, but does not by itself guarantee lossless communication, semantic composition, or robustness to specialist updates.

3 Architecture and Training

Figure 1 summarizes Sabha-B. The illustrated top-2 route activates the generalist and code islands; the reasoning island is inactive. The single block F denotes pooling and projection applied separately to each active island.



Legend

A Available token context
B Query router
C Generalist island (active)
D Reasoning island (inactive)
E Code island (active)
F Pooling + bus projection
G Shared latent bus
H Council decoder
I Vocabulary head

Figure 1: Sabha-B with top-2 activation. Darker routes select C and E; the lighter route leaves D inactive. F denotes separate per-island pooling and projection, shown as one stage rather than shared projection weights. The Council H cross-attends to bus G and produces vocabulary logits through I. The horizontal context path schematizes delivery of the original token context to H, not a router-generated representation. Routing features and causal masking are described in Sections 3.2–3.3.

3.1 Components and Parameter Boundaries

Sabha contains N decoder-only specialist models M_i, a router $\mathcal{R}$, a message interface $\mathcal{B}$, and an arbiter $\mathcal{A}$. We distinguish K, the number of activated islands, from m, the number of messages emitted by each activated island. The specialist transformer blocks have separate parameters θ_i. All components use a common vocabulary; the prototype description also uses a shared lexical interface with tied input/output embeddings. Independence therefore refers to the transformer blocks, not necessarily to every embedding or output parameter. Reported model sizes and this accounting boundary are discussed in Section 4 and Supplementary Appendix B.

For a conditioning sequence x of length T, island i produces hidden states $H_i = M_i(x) \in \mathbb{R}^{T \times d}$. Our islands use $d = 1024$, 16 layers, 16 attention heads, and feedforward width 4096; each is reported as approximately 234M parameters. The three training streams cover general text (M_G), mathematical reasoning (M_R), and code (M_C). Domain-specific pretraining assigns these roles explicitly; it does not establish statistical independence or prove that the underlying corpora contain no overlapping documents or capabilities.

3.2 Query-Level Island Selection

The router uses the mean-pooled first-layer representation of the generalist as a prompt summary, $p = \mathrm{mean}(M_G^{(1)}(x)) \in \mathbb{R}^d$. It computes

$$s_i = w_i^\top p + b_i, \qquad \alpha = \mathrm{softmax}(s), \qquad \mathcal{S}_K(x) = \mathrm{TopK}(\alpha). \tag{1}$$

The selected weights are renormalized as $\widetilde{\alpha}_i = \alpha_i / \sum_{j \in \mathcal{S}_K} \alpha_j$. Only selected islands execute their full specialist forward pass. Reading the generalist's first layer is a routing overhead even when the generalist is not selected; reusable computation must be counted only once. This choice introduces an asymmetry at the input interface, but does not make the generalist the output hub. The main configuration uses $N = 3$ and $K = 2$.

3.3 Pooled Latent Messages

Learned message queries $Q_{\mathrm{msg}} \in \mathbb{R}^{m \times d}$ summarize the hidden states of each selected island:

$$Z_i = \mathrm{softmax}\left(\frac{Q_{\mathrm{msg}} H_i^\top}{\sqrt{d}}\right) H_i \in \mathbb{R}^{m \times d}, \qquad B_i = Z_i W_i^{\mathrm{bus}} \in \mathbb{R}^{m \times d_b}. \tag{2}$$

Here $W_i^{\mathrm{bus}} \in \mathbb{R}^{d \times d_b}$ maps island-specific representations into a common bus width. The Council receives

$$B = \mathrm{Concat}_{i \in \mathcal{S}_K(x)} B_i \in \mathbb{R}^{Km \times d_b}. \tag{3}$$

Pooling bounds the number of message vectors presented to the decoder by Km, instead of passing every island hidden state at every layer. It does not remove the cost of producing those states, and the messages are a potentially lossy bottleneck. The implemented exchange is specialist-to-Council aggregation: islands do not directly read each other's messages or iterate through a recurrent negotiation protocol.

Causal scope. Equations 1–3 describe an operation on an available conditioning sequence. For autoregressive prediction of token x_t, every routing feature and message used for that prediction must depend only on permitted context, not on x_t or later target tokens. A causal self-attention mask inside the Council is not sufficient if its cross-attention memory has pooled future target states. The same restriction applies to full-sequence summaries used by the router. This is a necessary implementation condition for interpreting the reported losses as causal language-model losses; the present record does not include a prefix-invariance audit. We do not infer causal correctness from the pooling equation alone.

3.4 Two Output Arbiters

Sabha-A: weighted logit fusion. The first variant combines island logits on the common vocabulary:

$$\ell_A(x) = \sum_{i \in \mathcal{S}_K(x)} \widetilde{\alpha}_i \ell_i(x). \tag{4}$$

This adds no decoder parameters beyond the routing interface. It provides an output-fusion comparator, but is not an oracle router or an exhaustive baseline for all possible expert ensembles.

Sabha-B: Council decoder. The second variant uses a four-layer, four-head decoder of width $d_c = 512$, reported as approximately 25M parameters. It self-attends over its token context and cross-attends to B. Writing its hidden output as a_t, the token logits are

$$a_t = \mathcal{A}_{\theta_c}(x_{<t}, B_t) \in \mathbb{R}^{d_c}, \qquad \ell_{B,t} = g_{\mathrm{vocab}}(a_t) \in \mathbb{R}^V. \tag{5}$$

B_t denotes messages restricted to the information available at position t; this notation specifies the causal requirement rather than a particular caching implementation. The vocabulary map g_{vocab} includes any width adaptation needed by the lexical interface. In particular, the 512-dimensional decoder state cannot be multiplied directly by an unadapted 1024-dimensional island output matrix. Cross-attention's query and key/value projections likewise accommodate the different Council and bus widths.

3.5 Objective and Staged Optimization

The training formulation combines prediction loss with compute and load-balancing terms:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda\, \mathbb{E}[C_{\text{activated}}] + \beta\, \mathcal{L}_{\text{balance}}, \qquad \mathcal{L}_{\text{balance}} = N \sum_{i=1}^{N} f_i \bar{\alpha}_i. \tag{6}$$

Here $\bar{\alpha}_i$ is the batch-average router probability and f_i is the fraction of selected routing slots assigned to island i. This auxiliary form follows Switch-style load balancing (Fedus et al., 2022); it encourages use of multiple experts but does not guarantee uniform hard selections. The reported coefficients are $\lambda = 0.1$ and $\beta = 0.01$.

Fixed sparsity is not learned compute allocation. For equal-cost islands and fixed K, the sum of activated island costs is KC, independent of their identities. In that setting a cost term alone cannot learn how many islands to execute. The present efficiency result is therefore attributed to the top-2 architecture, not to evidence that the router has learned a variable per-query budget. Equation 6 also accommodates configurations with variable activation costs, but those are not evaluated here.

Training proceeds in three stages. First, each island is pretrained on its assigned domain stream. Second, island weights are frozen while the Council is trained on a multitask mixture. Third, the top four transformer layers of each island are unfrozen alongside the Council for joint fine-tuning at learning rate 3×10^{-5}. Separate parameter ownership thus persists, although Stage 3 couples the islands through a shared system-level loss. Independence of the pretraining process should not be confused with independence of the final behavior.

3.6 Computation and Specialist Updates

Computation. With shared work counted once, the system's arithmetic cost decomposes into routing, activated specialists, pooling/projection, and decoding:

$$C_{\text{Sabha}} = C_{\mathcal{R}} + \sum_{i \in \mathcal{S}_K} C_i + C_{\mathcal{B}}(K, m, T) + C_{\mathcal{A}}(K, m, T). \tag{7}$$

Reducing K removes specialist computation, but a parameter count is not a FLOPs count. Attention, output projection, sequence length, and coordination overhead matter. Consequently, K/N is only a rough intuition, not a bound relative to an arbitrary parameter-matched dense network. Section 5.2 reports the analytical estimates used in the experiment.

Parameter-local updates and system behavior. Separate specialist backbones make it possible to restrict an update to one island while keeping the other islands, shared lexical parameters, and Council fixed. This leaves the standalone non-target island functions unchanged at fixed input and evaluation state. It does not ensure unchanged system behavior: the updated island may send different messages to the Council on any domain where it is consulted. Parameter locality therefore specifies *where an update is stored*, not *which tasks it can affect*. Section 5.3 measures target and non-target losses after a specialist update rather than assuming behavioral isolation from the architecture.

4 Experimental Setup

Models and comparison scope. Sabha-B contains three reported 234M-parameter islands and a 25M Council (nominal sum 727M). Sabha-A uses the same island configuration without the decoder (nominal sum 702M). Dense-729M is a single transformer with width 1536, 24 layers, 24 heads, and feedforward width 6144. The dense baseline is trained on the concatenation of the three domain corpora. These are similar *reported model sizes*; exact distinct-parameter accounting depends on lexical sharing and connector inclusion, as detailed in Supplementary Appendix B.

Our BTS implementation reuses the three specialist models, takes the generalist as the frozen hub, and inserts gated linear stitch projections at transformer blocks. All three models execute on every forward pass; only stitch parameters are trained. This is a BTS-style comparison at the study's scale, not a claim of reproducing the published BTS paper's full training budget or optimized settings. In addition, Sabha's final stage adapts some specialist layers whereas this baseline does not; the comparison cannot isolate communication topology from that optimization difference.

Table 1: Domain streams used in island pretraining. Counts describe the available tokenized corpora, not an audited total of tokens consumed by each training run.

Island	Role	Corpus	Tokens
M_G	General text	C4 (Raffel et al., 2020)	1,034M
M_R	Reasoning	AutoMathText, OrcaMath, MathInstruct	504M
M_C	Code	CodeSearchNet (Husain et al., 2019)	216M

Table 2: Main model comparison: held-out cross-entropy (↓). Model sizes are the reported nominal counts. Bold indicates the lowest point estimate, not statistical significance. The multitask stream is evaluated separately.

Model	Size	Multitask	General text	Reasoning	Code
Sabha-B	727M	**7.708**	7.880	**7.480**	7.878
Dense	729M	7.721	**7.830**	7.514	**7.806**
BTS-style	∼727M	7.873	7.841	7.870	8.279

Data and optimization. Table 1 summarizes approximately 1.754B corpus tokens. The reasoning stream combines AutoMathText (math-ai, 2024), OrcaMath (Microsoft, 2024), and MathInstruct (TIGER-Lab, 2023). All streams use a common 32K SentencePiece vocabulary (Kudo & Richardson, 2018). The reported training setup uses BF16 on NVIDIA GPUs with 80GB memory, AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1), and a cosine learning-rate schedule (Loshchilov & Hutter, 2017). Island pretraining runs for 50K steps with reported batch size 4 and sequence length 2048. Council training runs for 5K–10K steps, followed by 2K–5K joint fine-tuning steps. The dense baseline uses sequence length 1024 because of memory constraints. Thus the study is not a strictly matched-context or audited matched-training-compute comparison.

Evaluation. We report held-out cross-entropy loss; lower is better. Each reported evaluation uses 100 batches, batch size 2, and sequence length 512. The multitask evaluation and individual-domain evaluations are distinct reported streams: the multitask column is not defined as the arithmetic mean of the other columns. Main model comparisons use the values from the final result table. Earlier arbiter and training-duration diagnostics are kept separate because they refer to different reported checkpoints. Results are point estimates without repeated-seed uncertainty intervals. No downstream benchmark accuracy is claimed.

5 Results

5.1 Aggregate Predictive Quality

Table 2 contains the central comparison. Sabha-B has the lowest multitask point estimate: 0.013 below dense and 0.165 below BTS-style. Against the dense model it improves reasoning loss by 0.034, but general-text and code losses are higher by 0.050 and 0.072. It therefore offers a different domain tradeoff, not uniformly better domain performance.

The aggregate margin over dense is small. We interpret the result as comparable predictive quality in this experiment, rather than an established generalization advantage. Together with the analytical cost estimate below, it identifies a promising lower-arithmetic operating point. The larger gap to our BTS-style baseline is consistent with the usefulness of the separate Council interface in this setup, but does not identify whether sparsity, decoder capacity, training procedure, or topology is responsible.

5.2 Arithmetic Cost and Measured Throughput

The top-2 configuration activates two 234M islands plus the 25M Council under the nominal counting convention. The recorded analytical cost is 1,158M FLOPs/token, compared with 1,556M for dense and 1,601M for BTS-style (Table 3). We retain the reported normalized value of 74.5%; ratios recomputed from the rounded FLOP totals differ slightly. The supported efficiency claim is therefore approximately 25.5% less estimated arithmetic than dense at similar aggregate loss. It is a pointwise

Table 3: Analytical inference cost and reported multitask loss. M denotes millions. Active parameters use the nominal component-count convention. The top-1 entry has no accompanying quality measurement and is not a demonstrated quality–cost point.

Configuration	Active params	FLOPs/token	Dense fraction	Loss
BTS-style (all)	∼727M	1,601M	102.9%	7.873
Dense	729M	1,556M	100.0%	7.721
Sabha-B, top-2	493M	1,158M	74.5%	7.708
Sabha, top-1	259M	624M	40.1%	—

Table 4: Reported specialist-update experiment, separate from the main comparison. Only the code island's top four layers are fine-tuned for 5K steps. Negative changes indicate lower loss. A small positive reasoning change is retained rather than reported as exact invariance.

Evaluation	Before	After	Change
System: code	8.02	7.90	-0.12
System: general text	8.17	7.88	-0.29
System: reasoning	7.47	7.48	$+0.01$
Code island alone	1.84	1.72	-0.12

comparison, not an established Pareto frontier over alternative model sizes, budgets, or training recipes.

The current implementation is not faster. Reported throughput is 31,060 tokens/s for Sabha-B versus 127,883 tokens/s for dense, or approximately $0.243\times$ dense throughput. The prototype dispatches selected islands sequentially in Python, while dense executes a contiguous model. This is a plausible source of overhead, not a measured attribution of the entire gap. Parallel dispatch and kernel optimization may improve performance, but no such speedup has been measured. The throughput figures also should not be interpreted as an optimized autoregressive serving benchmark without an equivalent decoding protocol.

5.3 Updating One Specialist

Updating only the code island improves its standalone code loss and the system's code loss by 0.12 (Table 4). General-text loss decreases by 0.29, while reasoning loss increases by 0.01. This demonstrates a useful targeted update in one observed case, with limited measured adverse change on the reported non-target evaluations.

The non-target island parameters remain fixed by the update procedure. The system's non-target losses do not remain fixed: they change in both directions. This illustrates the distinction between parameter-local updates and system behavior in Section 3.6. The result supports parameter-local maintenance, but neither proves immunity to catastrophic forgetting nor establishes that repeated updates remain benign. We do not attribute the general-text improvement to a particular representation mechanism without a causal ablation, and we do not claim a measured maintenance advantage over dense or BTS because the corresponding update comparisons were not run.

5.4 Arbiter and Routing Diagnostics

A decoder-equipped system improves the reported early comparison. At the earlier 10K-step setting, logit fusion (Sabha-A) has multitask loss 8.47, the best standalone island has 8.13, and Sabha-B has 7.55. Thus the decoder-equipped system improves on the fusion comparator by 0.92 and on the best island by 0.58. This supports the utility of the Sabha-B configuration, but adding the decoder also adds approximately 25M parameters and changes the optimization problem. The comparison does not isolate latent communication from additional capacity or show that no other output-fusion scheme could succeed. Supplementary Appendix A keeps these checkpoints distinct from the main 7.708 result.

Higher soft-score entropy does not imply diversified hard routing. The reported router entropy rises from 0.46 to approximately 1.05 within 100 training steps and remains near that value, compared with $\log 3 \approx 1.10$. However, an inference diagnostic assigns half of selected slots to the generalist and half to the code island, with no reasoning-island selections. The observations are compatible: nearly uniform soft probabilities can repeatedly produce the same top-2 ordering. At fixed top-2, zero reasoning selections mean the diagnostic is consistent with always using the same generalist–code pair. It therefore does not establish successful query-adaptive specialization, even though computation is sparse. The diagnostic's separately recorded multitask loss is 7.718; we do not substitute it for the final-table value or assume the routing trace identifies that exact checkpoint.

A larger island gap is not a scaling law. Earlier training-duration diagnostics report Sabha-B/best-island losses of 7.55/8.13 at 10K steps and 7.72/10.08 at 50K. Although the difference grows from 0.58 to 2.36, the composed system's own loss increases by 0.17 and the comparator worsens substantially. These observations do not establish improved absolute performance with scale. Nor does exceeding the best island's aggregate score establish within-example composition: a specialist selector can outperform every single expert's overall average. We retain the diagnostics as evidence about the observed configurations, not as a scaling or compositionality theorem.

6 Discussion and Limitations

Sabha targets the coupling of capacity, execution, and maintenance within a single model backbone. Separating specialist transformer blocks from a compact coordination interface creates distinct design choices: which specialists to retain, which to execute, and which parameters to update. The reported quality–arithmetic result and specialist update provide initial support for this organization. Its prospective value is greatest when selective execution and separately maintained domains are objectives in their own right, rather than when the sole objective is the lowest loss or latency from a fixed model.

The cost of modularity. Sabha trades a contiguous execution path for routing, message compression, and cross-model coordination. Lower estimated FLOPs need not yield lower wall-clock cost, as the current throughput demonstrates. That penalty remains an obstacle for latency- or throughput-sensitive use, not a speedup that can be presumed from future optimization. Likewise, parameter-local updates still require regression testing of composed outputs. These costs may be acceptable when separately managed expertise matters, but a maintenance or serving advantage over dense adaptation is not established here. Causal evaluation and consistent cost accounting, in contrast, are validity requirements rather than optional architectural tradeoffs.

Predictive quality and experimental scope. All reported quality results are held-out cross-entropies. They do not establish problem-solving accuracy, executable code correctness, or performance at larger model sizes. The dense margin is small and no seed variation is reported. Different training sequence lengths, incompletely specified token exposure, and the extra specialist adaptation used by Sabha constrain causal interpretation of the baseline comparisons. Corpus assignment is not a deduplication or contamination audit.

Interface and accounting scope. The recorded architecture description does not fully resolve lexical-weight sharing, connector-inclusive parameter totals, or the exact causal pooling implementation. The nominal sizes and analytical FLOPs should therefore not be read as an audited unique-parameter or serving-cost ledger. Causal access control must hold for both the router and latent bus, not just the decoder. Supplementary Appendix B states the relevant boundaries rather than inferring missing implementation facts.

Mechanism and generality. The current evidence does not isolate the contributions of routing, communication, and Council capacity. It does not show that all three specialists are needed, and the available hard-routing diagnostic underuses the reasoning island. There is no measured top-1 quality, no multi-budget frontier, and no comparison against sparse MoE or an oracle expert selector. These limits narrow the claim to the reported architecture and operating point; they do not negate the observed quality–FLOPs tradeoff.

7 CONCLUSION

Sabha explores an alternative to treating a language model as one monolithic unit of capacity, computation, and maintenance. Separately pretrained specialist backbones supply domain capacity; a compact Council coordinates a sparse subset through a bounded latent interface. The central idea is to make specialist capacity, active computation, and parameter updates distinct architectural choices, rather than organizing all three around a single dense backbone.

At the evaluated scale, the reported results provide initial support for this design. Top-2 Sabha-B achieves multitask cross-entropy of 7.708, compared with 7.721 for dense and 7.873 for our BTS-style baseline, at 74.5% of the dense model's estimated inference FLOPs (Tables 2–3). The important result is competitive aggregate prediction with sparse whole-model execution, not an established accuracy advantage. Updating the code island reduces system-level code loss by 0.12, with a 0.01 increase in reasoning loss and an improvement on general text (Table 4), providing one measured example of useful parameter-local maintenance.

The tradeoff is coordination: explicit specialist boundaries introduce routing, communication, and execution overhead, while local parameter updates do not guarantee behaviorally isolated changes. The prototype's throughput and routing limitations remain practical constraints; the evaluation and accounting boundaries in Section 6 delimit the present evidence. The design's value must therefore be judged against the needs of a modular system, not FLOPs alone.

The broader direction is to develop LLMs as evolving collections of specialists, with learned interfaces governing how their capacity is used. Can domain specialists be added or refreshed while keeping active computation bounded and preserving previously supported behavior? This is a research objective, not a demonstrated scaling law. Sabha supplies an initial architecture and empirical operating point for pursuing it. The opportunity is not merely to make an LLM smaller, but to make its capacity modular, its execution selective, and its evolution incremental.

AI USE STATEMENT

Generative AI assisted manuscript structure and language, literature identification and reference verification, and critical review of mathematical formulations and result interpretations. All numerical experimental results are taken from pre-existing runs; no new experimental measurements were generated during this revision. The authors remain responsible for the accuracy of the methodology, results, citations, and final text.

ETHICS STATEMENT

The study concerns language-model architecture and does not include a human-subject experiment. Modularity is not a safety guarantee: specialists and their composed outputs may retain errors, biases, or unsafe behavior from training data. Any release or deployment should respect the terms of the source datasets and evaluate the complete system rather than assuming that isolated parameter updates preserve all non-target behaviors.

REPRODUCIBILITY STATEMENT

To support independent reproduction and broader experimentation, we will release the Sabha source code in a public repository, including scripts for specialist pretraining, Council training, joint fine-tuning, and specialist updates, together with the dense and BTS-style baseline implementations used in this study. The release will document data acquisition, preprocessing, tokenization, and split construction; model, routing, pooling, and causal-masking configurations; software dependencies and hardware requirements; and the recorded hyperparameters, random seeds, and checkpoint identifiers for each experiment. Evaluation scripts will cover loss evaluation, routing diagnostics, update comparisons, analytical FLOP accounting, and throughput measurement, with instructions linking configurations and outputs to the paper's tables. Original run artifacts will be distinguished from any reconstructed settings, and unavailable metadata or other barriers to exact reproduction will be explicitly identified. Dataset redistribution will respect the source licenses, with acquisition instructions supplied where data cannot be included. The repository will expose interfaces for specialist backbones, routers, latent-message exchange, and Council decoding so that researchers can substitute domain-specific models and datasets, vary the number of specialists and activation budgets, and examine alternative coordination mechanisms. Our aim is to make Sabha an accessible experimental framework for investigating when language-model capacity can be organized into separately trained, selectively executed, and incrementally maintained components across domains.

REFERENCES

Marcus Armstrong, Navid Ayoobi, and Arjun Mukherjee. Dead weights, live signals: Feedforward graphs of frozen language models. *arXiv preprint arXiv:2604.08335*, 2026. URL https://arxiv.org/abs/2604.08335.

Rachit Bansal, Bidisha Samanta, Siddharth Dalmia, Nitish Gupta, Shikhar Vashishth, Sriram Ganapathy, Abhishek Bapna, Prateek Jain, and Partha Talukdar. LLM augmented LLMs: Expanding capabilities through composition. *arXiv preprint arXiv:2401.02412*, 2024. URL https://arxiv.org/abs/2401.02412.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling agents to communicate entirely in latent space. *arXiv preprint arXiv:2511.09149*, 2025. URL https://arxiv.org/abs/2511.09149.

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

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A. Smith, and Luke Zettlemoyer. DEMix layers: Disentangling domains for modular language modeling. In *Proceedings of NAACL-HLT*, pp. 5557–5576, 2022. doi: 10.18653/v1/2022.naacl-main.407. URL https://aclanthology.org/2022.naacl-main.407/.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019. URL https://arxiv.org/abs/1909.09436.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018. URL https://arxiv.org/abs/1808.06226.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models. *arXiv preprint arXiv:2208.03306*, 2022. URL https://arxiv.org/abs/2208.03306.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. URL https://arxiv.org/abs/1711.05101.

math-ai. AutoMathText. Dataset, 2024. URL https://huggingface.co/datasets/math-ai/AutoMathText.

Microsoft. Orca math word problems 200k. Dataset, 2024. URL https://huggingface.co/datasets/microsoft/orca-math-word-problems-200k.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL https://www.jmlr.org/papers/v21/20-074.html.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017. URL https://arxiv.org/abs/1701.06538.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Daniel Li, Wen-tau Yih, Jason Weston, and Xian Li. Branch-train-MiX: Mixing expert LLMs into a mixture-of-experts LLM. *arXiv preprint arXiv:2403.07816*, 2024. URL https://arxiv.org/abs/2403.07816.

TIGER-Lab. MathInstruct. Dataset, 2023. URL https://huggingface.co/datasets/TIGER-Lab/MathInstruct.

Qizhen Zhang, Prajjwal Bhargava, Chloe Bi, Chris X. Cai, Jakob Foerster, Jeremy Fu, Punit Singh Koura, Ruan Silva, Sheng Shen, Emily Dinan, Suchin Gururangan, and Mike Lewis. BTS: Harmonizing specialized experts into a generalist LLM. *arXiv preprint arXiv:2502.00075*, 2025. URL https://arxiv.org/abs/2502.00075.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*, 2025. URL https://arxiv.org/abs/2511.20639.