Learning as Routing: The Self-Organizing Mixture of Experts (SoME) Framework (Official SoME v2)

Our thesis is that learning can be effectively reframed as discovering optimal computational pathways through a static, random, or pre-trained knowledge base, rather than adapting the weights of the knowledge base itself.

At its core, SoME reframes learning from weight adaptation to information retrieval and composition. It fundamentally decouples a function's knowledge from its address.

- The Knowledge (The "What"): An immutable, static library of "primitive" functions (the frozen experts). The system's potential is gated by the initial diversity of this library—a "chaotic, high-entropy primordial soup" has proven to be the most effective starting point.
- The Address (The "Where"): A dynamic, evolving "address book" (the Key Store) that maps inputs to the relevant experts.

The Learning: Learning is the process of updating this address book. Instead of using backpropagation to change what the experts know, SoME uses what's called "Knowledge Gravity" to change how the system accesses and composes what it already knows. (This is the core differentiation from standard language modeling.)

Core Claims to Adjudicate

- Claim A (The Reservoir Claim): A model can be initialized with random noise, have 99% of its weights frozen ("Knowledge"), and successfully learn tasks solely by training the routing/addressing mechanism ("Address").
- Claim B (The Compositionality Claim): This architecture decouples knowledge from access, allowing "compute primitives" to be recycled/composed for new skills.
- Claim C (The Lifelong Learning Claim): Freezing the knowledge base solves catastrophic forgetting.

Data / Task Suite

- Task 1: Wikitext-103 (Language Modeling). Standard benchmark to see if the model produces coherent text or garbage.

Metrics:

1. Standard Metrics like (Train Loss, Validation Loss, and Perplexity (PPL)) on Wikitext.

The "Bottleneck-Free" Experimental Plan: We will dissect the SoME architecture into its three fundamental components and test each one in isolation under ideal conditions.

1. Phase 1: Characterizing the Optimal Expert Library (The "What")
2. Phase 2: Mapping the Router's True Capacity (The "How")
3. Phase 3: Tuning the Heuristic Learning Engine (The "Learning")

Phase 1: Characterizing the Optimal Expert Library (The "What")
Guiding Question: What are the quantitative properties of the most effective "primordial soup" of static, frozen experts?

- Experiment 1.1: Heads vs Layers Scaling
    - Objective: Run the standard scaling ablation with (number of heads) and (number of layers) to find the SoME architectural sweet spot.
    - Method: Create a number of runs to test each component individually in a “Bottleneck-Free” environment.
    - Hypothesis: The idea is that there is a sweet spot for every architecture to best scaling, this should hold true with SoME and will take some time to find.
- Experiment 1.2: Initialization Method Revisited
    - Objective: Re-run the default vs. orthogonal vs. sparse test, but this time using the powerful MLP router.
    - Hypothesis: The performance gap between sparse and the others will likely increase. A more intelligent router should be better at exploiting the specialized functions provided by sparse initialization.
    - Metrics: Validation Perplexity, Gini Coefficient, Shannon Entropy.
- Experiment 1.3: Quantifying Sparsity
    - Objective: Test the degree of sparsity in the expert weights. Is a 90% sparse expert better than a 50% sparse one?
    - Method: Create several expert libraries with varying levels of init_method="sparse" (e.g., 0.5, 0.7, 0.9, 0.95 sparsity) and train the model.
    - Hypothesis: There is likely a sweet spot. Too little sparsity creates generalists; too much might create functionally dead experts. This will help define the "diversity" of the library more formally.
- Experiment 1.4: Expert Size (d_ffn) vs. Expert Count (num_experts)
    - Objective: At a fixed total parameter count, is it better to have many small experts or fewer large experts?
    - Method: Compare configurations like (num_experts=128, d_ffn=1024) vs. (num_experts=128, d_ffn=1536), and vs (num_experts=256, d_ffn=512).
    - Hypothesis: Given the "Distinct Primitives" theory, a higher number of smaller, more specialized experts will likely outperform fewer, more general experts, and tells us what the ideal expert parameters are.

Phase 2: Mapping the Router's True Capacity (The "How")
Guiding Question: What is the relationship between a router's architecture, the complexity of the expert library, and the onset of "Router Collapse"?

- Experiment 2.1: The Collapse Point of a Linear Router
    - Objective: Precisely map the failure boundary of the original nn.Linear Query Network.
    - Method: Using the optimal expert library from Phase 1, systematically increase num_experts (e.g., 64, 128, 256, 512, 1024) and d_model (e.g., 384, 512, 768)

and record the Gini/Perplexity. This is a deliberate stress test to chart the exact point of collapse.
    - Outcome: A 2D "phase diagram" showing the "Stable" vs. "Collapsed" regions for the linear router. This turns a "bug" into a formal characterization of the component's limits.
- Experiment 2.2: The Collapse Point of an MLP Router
    - Objective: Quantify the increase in capacity gained by an MLP router.
    - Method: Repeat the exact same stress test from 2.1, but with a small MLP router.
    - Hypothesis: The MLP router's "Stable" region on the phase diagram will be significantly larger than the linear router's, proving its ability to manage a more complex expert library.
- Experiment 2.3: The ANN Search Fork
    - Objective: Determine the crossover point where an Approximate Nearest Neighbor (ANN) search becomes more efficient than exact matrix multiplication.
    - Method: Implement a FAISS/HNSW index for the Key Store. For a very large number of experts (e.g., 4096+), benchmark the wall-clock time and perplexity of the exact search (matrix multiply) vs. the ANN search.
    - Outcome: A clear data point: "At N experts, the speed/memory benefits of an ANN index outweigh the accuracy cost of an approximate search." This justifies a key architectural decision for scaling.

Phase 3: Tuning the Heuristic Learning Engine (The "Learning")
Guiding Question: How do the Knowledge Gravity heuristics (α, β, δ) truly interact, and how sensitive is the system's self-organization to their balance?

- Experiment 3.1: Alpha-Beta Sensitivity Analysis
    - Objective: Move beyond empirical tuning to a systematic analysis of the core specialization (α) and clustering (β) forces.
    - Method: Using the non-collapsing MLP router and the optimal expert library, run a grid search over a range of alpha and beta values.
    - Metrics: This is crucial. Don't just look at perplexity. For each run, generate UMAP visualizations of the final Key Store. You are looking for the (α, β) combinations that produce the tightest, most well-defined "Knowledge Galaxies." You can even quantify this with clustering metrics like Silhouette Score or a Mutual Information score between clusters and token types (e.g., part-of-speech tags).
- Experiment 3.2: The Role of Inertia and Forgetting (δ)
    - Objective: Formalize the role of the stabilizing/pruning forces.
    - Method: With the "optimal" α/β balance found in 3.1, conduct an ablation study where you vary the ema_decay and the forgetting_threshold (δ).
    - Hypothesis: This will reveal the importance of the "Adaptive Equilibrium." Too little inertia and popular experts will dominate (high Gini). Too much forgetting and the Key Store will be unstable.

Phase 4: Determining the Modes of Operation within our SoME architecture

Guiding Question: is SoME's performance a function of its operational mode (Composition vs. Retrieval), which is determined by the interplay between router capacity (top_k, MLP size) and the richness of the expert library (num_experts).

- Experiment 4.1 is all about forcing the mode (The top_k Experiment) This study directly tests the importance of composition vs. retrieval.
- 4.1.1: Force Retrieval (top_k=1):
    - Configuration: Use your best compositional model, (num_experts=64).
    - Change: Set top_k = 1. This completely disables the Beta heuristic and forces a pure retrieval strategy.
    - Question: How much does performance degrade when a model optimized for composition is forced into retrieval?
    - Prediction: The perplexity will get significantly worse than 2.04. This would prove that composition was essential for the success of the 64-expert model.
- 4.1.2: Enhance Composition (top_k=8):
    - Configuration: Use your retrieval-mode model, (num_experts=256).
    - Change: Set top_k = 8. This gives the saturated router more "bandwidth" to select and combine multiple experts.
    - Question: Can we improve the performance of a high-expert-count model by giving it more compositional ability?
    - Prediction: The perplexity will improve, dropping below 2.05. The Gini coefficient should also decrease slightly as the routing load is spread across more experts. This would prove that even in retrieval mode, the model still benefits from compositional capacity.
- 4.2: The Role of Beta in Retrieval Mode; This study tests what the Beta heuristic is doing when the model is no longer primarily forming "squads."
- 4.2.1: No Beta at Scale:
    - Configuration: Use a model of (num_experts=256).
    - Change: In ablation_flags, set use_beta = False.
    - Question: Is the Beta heuristic still important when the model is in Retrieval Mode?
    - Prediction: Performance will degrade, but perhaps not as catastrophically as in a low-expert model. In Retrieval Mode, Beta's job likely shifts from "squad formation" to creating "neighborhoods of similarity" in the key store. It pulls functionally similar specialists together, making the search problem easier for the saturated router. Removing it will hurt, but Alpha (specialization) will still allow the model to function to some degree. The UMAP plot for this run will be a completely unstructured, amorphous cloud, proving Beta was the sole organizing force.

Phase 5: Optimizing the Standard Transformer Backbone
Guiding Question: To find a stable and efficient set of standard training hyperparameters for the SoME architecture. During this phase, we will keep the SoME-specific parameters fixed to reasonable defaults.

5.1 The Search Space (What Optuna will test)

- Experiment 1: Learning Rate (The Most Important)
  - lr = trial.suggest_float("lr", 1e-5, 9e-4, log=True)
  - Rationale: This covers the typical range for training Transformers. A log scale is crucial as LR's effect is multiplicative.
- Experiment 2: Optimizer
  - optimizer = trial.suggest_categorical("optimizer", ["AdamW", "Adafactor"])
  - Rationale: AdamW is the standard. Adafactor is a memory-efficient alternative from Google that can sometimes be more stable for large models, which is relevant to our single-GPU constraint.
- Experiment 3: Scheduler Warmup Steps
  - warmup_steps = trial.suggest_int("warmup_steps", 100, 500)
  - Rationale: A proper warmup is critical for training stability. Finding the right duration is key.
- Experiment 4: Weight Decay
  - weight_decay = trial.suggest_float("weight_decay", 0.01, 0.1)
  - Rationale: This is a key regularization parameter to prevent overfitting.