(Focus Labs) A Technical Deep Dive into the Self-Organizing Mixture of Experts (SOME) Architecture v2

1.0 Introduction to the SOME Paradigm

The Self-Organizing Mixture of Experts (SOME) architecture represents a significant evolution in the design of large-scale neural networks. Its core premise is the strategic separation of an expert's internal function (its weights) from its address (its routing key). In the SOME model, an expert's specialized function remains stable and unchanging, while its address is plastic, capable of evolving during the inference process. This dynamic adaptation is driven by a process metaphorically described as "Knowledge Gravity," which continually refines the discoverability of experts based on their usage patterns.

This whitepaper provides a detailed examination of the SOME architecture. We will detail the foundational design choices that enable this paradigm, explore the dual-update mechanisms that power its dynamic key updates, and analyze the crucial stability protocols that ensure the system's long-term viability and prevent architectural collapse.

2.0 Foundational Design: Static Experts and Dynamic Keys

The most critical design decision in the SOME architecture is the separation of a static, functional component from a dynamic, addressable one. This choice is the bedrock upon which the model's capacity for in-inference adaptation is built, allowing it to learn from new data without suffering the catastrophic forgetting that plagues traditional models when their core weights are modified.

2.1 The Role of Static Experts

In the SOME model, a vast pool of experts are first trained using conventional pre-training methods. Following this phase, their internal parameters—specifically the up-projection and down-projection weights (w_up, w_down)—are permanently frozen. These experts function as a collection of stable and reliable specialists. Their core knowledge is fixed, ensuring that the model's foundational capabilities remain intact and predictable. They represent a static library of functions that the model can call upon.

2.2 The Function of Dynamic Keys

While the experts themselves are static, their accessibility is not. Each expert is assigned a plastic routing key (k), which is a vector that exists in the same high-dimensional space as the router's query vector (q). These keys are not part of the expert's frozen parameters. Instead, they are stored in a dynamically updatable key-value store, such as a FAISS index or a custom GPU structure. This allows the model to modify an expert's address without ever touching its internal function.

2.3 The Rationale for Separation

This architectural separation provides a direct solution to the classic stability-plasticity dilemma. It decouples the process of learning new associations from the risk of corrupting core knowledge. Attempting to update the multi-million parameter weights of an expert at test time would lead to immediate catastrophic forgetting and is computationally infeasible for real-time

applications. In contrast, updating a small key vector is computationally trivial, involving a simple vector addition. This operation is orders of magnitude faster and less resource-intensive than a backward pass required to modify expert weights, making in-inference adaptation feasible at production scale. This design elegantly modifies an expert's discoverability rather than its core function. The model isn't re-learning how to perform a task; it's learning how to better organize its internal "address book" to find the right specialist for the task at hand. This foundational design creates a stable yet flexible architecture, setting the stage for the dynamic forces that act upon it.

3.0 The Dual-Update Mechanism: The Forces of Knowledge Gravity

After a batch of data is processed, the SOME model enters a brief "synaptic consolidation" phase. During this phase, the dynamic keys of the activated experts are updated by two distinct but complementary attractive forces. Collectively, these forces are known as "Knowledge Gravity," pulling relevant experts together and toward the problem spaces where they are most effective.

3.1 Force 1: Query Pull (Relevance Attraction)

The first force, Query Pull, operates as a k-means-like drift, pulling an expert's key closer to the queries that successfully activate it. In this analogy, the expert's key acts as a cluster centroid, and the update moves it toward the mean of the data points (the query vectors) it has been assigned to. For each token that generates a query vector q and activates an expert e, the expert's key k_e is nudged toward q. This mechanism ensures that an expert's address migrates toward the conceptual center of the problems it is best suited to solve.

The mathematical formulation for this update is:

$$k_{e,new} = k_e + \alpha * (q - k_e)$$

Here, α represents a small "plasticity rate" that controls the magnitude of the update. For example, an expert that is consistently activated by queries related to Python syntax will see its key gradually drift into the center of the "Python syntax" region of the vector space, making it easier to find for future, similar queries.

3.2 Force 2: Peer Pull (Hebbian Co-activation)

The second force, Peer Pull, forges strong synaptic links between complementary specialists that are co-activated to process a single token (a common occurrence in Top-K routing where K>1). When two experts are selected together, their keys are pulled closer to each other, creating a conceptual and gravitational bond. This concept is inspired by Hebbian learning, where "neurons that fire together, wire together".

The update is applied symmetrically to both experts:

$$k1_{new} = k1 + \beta * (k2 - k1)$$

$$k2_{new} = k2 + \beta * (k1 - k2)$$

The term β is a "bonding rate," typically smaller than the plasticity rate α. The architectural significance of this mechanism is profound: it drives emergent functional composition. For instance, if a "for-loop expert" and a "variable-assignment expert" are frequently used in tandem, Peer Pull draws their keys together. This creates a de facto "meta-expert" for "programming basics," ensuring that these primitive skills are co-located and can be activated

as a single, more complex functional unit in the future. However, a system governed solely by attractive forces is inherently unstable, which necessitates the inclusion of countervailing stability mechanisms.

4.0 Maintaining Architectural Stability: Inertia and Decay

A system governed purely by the attractive forces of Knowledge Gravity would eventually collapse, with all expert keys converging into a single, undifferentiated mass. To prevent this and ensure a stable, useful organization of the expert key space, the SOME architecture incorporates countervailing forces that provide inertia and prevent systemic collapse.

4.1 Gravitational Mass (Usage Inertia)

To introduce stability, each expert is assigned a Gravitational Mass proportional to its overall activation frequency. This mass functions as inertia; more frequently used experts have higher inertia, which in turn scales down their learning rates (α and β). This inertia is critical for model predictability. It ensures that the model's core, generalist functions remain stable and robust, preventing high-frequency, specialist data from degrading the model's overall performance. These high-mass experts become stable "galactic centers" within the vector space, around which smaller, more niche experts can establish stable orbits, creating a structured and resilient knowledge hierarchy.

4.2 Dark Energy (Repulsive Force / Decay)

To prevent the total gravitational collapse of the key space, a repulsive force, or Dark Energy, is introduced. This mechanism ensures that experts maintain a degree of separation. The source material outlines two potential implementations:

1. Applying a small, constant repulsive force to all keys.
2. Implementing a decay mechanism where the keys of inactive experts slowly drift back towards their origin point.

The choice between these implementations presents a trade-off. A constant repulsive force is an active, persistent mechanism for maintaining space, while a decay-to-origin approach is a passive "forgetting" mechanism that purges infrequently used experts. The latter may be more computationally efficient and less prone to creating unstable oscillations in the key space. This force encourages the continuous exploration of the vector space and provides a method for pruning the network, as the keys of "dead" or useless experts can be recycled or ignored. Together, these stability mechanisms ensure that the dynamic organization driven by Knowledge Gravity results in a long-term, dynamic equilibrium rather than a catastrophic collapse.

5.0 Conclusion

The Self-Organizing Mixture of Experts (SOME) architecture provides a tangible architectural blueprint for achieving continual learning without scheduled retraining cycles. Its central innovation—the separation of stable expert function from a plastic expert address—resolves the fundamental conflict between continuous learning and the catastrophic forgetting that has long constrained AI development. By combining the dual attractive forces of Query Pull and Peer Pull with the stabilizing counter-mechanisms of Inertia and Decay, SOME creates a robust,

self-organizing system. This architecture shifts the paradigm from static, monolithic models to dynamic, self-organizing knowledge systems. It represents a critical and pragmatic step toward developing more autonomous, adaptive, and ultimately more efficient large-scale AI.

Formal Definition of the Self-Organizing Mixture of Experts (SOME) Architecture

1.0 Objective

To formally define a large-scale neural network architecture, SOME, that separates an expert's static function (its weights) from its dynamic address (its routing key). This separation enables continuous, in-inference adaptation to new data without catastrophic forgetting, while maintaining a stable knowledge base.

2.0 Core Components

A SOME layer is designed as a sub-module within a larger transformer architecture. Let us define its components for a single transformer layer l.

2.1 Static Components (The Knowledge Base)

These components are frozen after the initial pre-training phase.

- Expert Pool (E): A collection of M independent, dense expert networks.
- $E = \{e_1, e_2, ..., e_M\}$
- Individual Expert (e_i): Each expert e_i is a standard Feed-Forward Network (FFN) with its own unique, frozen weights. It performs a transformation on a hidden state $z \in \mathbb{R}^{d\_model}$.
- Function: $e_i(z) = W\_up_i(\sigma(W\_down_i(z)))$
- Parameters:
- $W\_down_i \in \mathbb{R}^{d\_ffn \times d\_model}$: The down-projection weight matrix.
- $W\_up_i \in \mathbb{R}^{d\_model \times d\_ffn}$: The up-projection weight matrix.
- σ: A non-linear activation function (e.g., GELU, SiLU).
- State: The parameters W_down and W_up are static and frozen post-training.

2.2 Dynamic Components (The Routing System)

These components remain plastic and are updated during the inference phase.

- Router (R): The router's function is to generate a query from an input token and use it to select relevant experts. It consists of two parts:
- Query Network (Q): A small, trainable neural network that maps an input hidden state $x \in \mathbb{R}^{d\_model}$ to a query vector $q \in \mathbb{R}^{d\_key}$.
    - Function: $q = Q(x)$
    - Note: The query space dimension d_key is typically equal to the model dimension d_model.
- Key Store (K): A dynamically updatable key-value store containing the routing keys for all M experts.
    - $K = \{k_1, k_2, ..., k_M\}$ where each $k_i \in \mathbb{R}^{d\_key}$ is the routing key for expert e_i.
- State: The keys k_i are plastic and evolve during inference.

3.0 Architectural Integration: The Forward Pass

The process for routing a single input token with hidden state x through a SOME layer is as follows:

1. Query Generation: The router's Query Network Q computes the query vector for the input token.
    - $q = Q(x)$
1. Expert Scoring: The query vector q is used to compute a similarity score s_i for each expert e_i by comparing q with the expert's key k_i. This is typically a dot-product similarity.
    - $s_i = q \cdot k_i$ for i = 1 to M
1. Top-K Expert Selection: The router selects the set I, which contains the indices of the K experts with the highest scores.
    - I = TopK_indices({s_1, s_2, ..., s_M})
1. Gating Weight Calculation: The scores of the selected experts are passed through a softmax function to compute the gating weights g_i, which determine the contribution of each selected expert.
    - g_i = softmax({s□ | j ∈ I}) for each i ∈ I
1. Sparse Expert Computation: The input x is processed in parallel by only the K selected experts.
    - $y_i = e_i(x)$ for each i ∈ I
1. Output Combination: The final layer output y_SOME is the weighted sum of the outputs from the selected experts.
    - y_SOME = Σ_{i ∈ I} g_i * y_i
1. Final Layer Output: This output is added to the original input via a standard residual connection.
    - y_output = x + y_SOME

4.0 Dynamic Update Mechanism: "Knowledge Gravity"

After a batch of data has been processed, the model enters a "synaptic consolidation" phase where the dynamic keys in the Key Store K are updated. This process is gradient-free (i.e., operates under torch.no_grad()).

4.1 Force 1: Query Pull (Relevance Attraction)

This force adapts an expert's key based on the queries that activated it. For each token-expert pair (q, e_i) where e_i was in the Top-K set for query q, the expert's key k_i is updated:

- k_i_new = k_i_old + α * (q - k_i_old)
- α is a small, positive scalar known as the "plasticity rate."

4.2 Force 2: Peer Pull (Hebbian Co-activation)

This force strengthens the association between experts that are frequently co-activated. For each pair of experts (e_i, e□) that were activated for the same query q:

- k_i_new = k_i_old + β * (k□_old - k_i_old)
- k□_new = k□_old + β * (k_i_old - k□_old)
- β is a small, positive scalar known as the "bonding rate," typically β < α.

5.0 Stability Mechanisms

To ensure the long-term stability of the self-organizing key space, two countervailing forces are introduced.

5.1 Gravitational Mass (Usage Inertia)

The learning rates α and β are modulated by an expert's activation frequency. Let usage(e_i) be a running count or moving average of expert e_i's activation frequency.

- $\alpha\_effective = \alpha / (1 + usage(e_i))$
- $\beta\_effective = \beta / (1 + usage(e_i))$
- This term ensures that frequently used, "generalist" experts have higher inertia and their keys update more slowly, preserving the model's core, stable knowledge.

5.2 Dark Energy (Repulsive Force / Decay)

To prevent the total collapse of the key space into a single point, a repulsive mechanism is implemented. Two primary options are:

1. Constant Repulsive Force: Apply a small, constant force that pushes each key k_i away from the centroid of all other keys.
2. Decay to Origin: Keys of experts with an activation frequency usage(e_i) below a certain threshold θ are slowly decayed towards the origin vector.
    - if $usage(e_i) < \theta$: $k_i\_new = k_i\_old * (1 - \delta)$
    - δ is a small, positive decay rate. This serves as a "forgetting" mechanism that prunes unused or irrelevant experts over time.