Neural Capsules: A Storage-Aware Framework for Hot-Swappable, Incrementally Composable Language-Model Experts

A Mathematical and Systems Research Proposal for Portable Quantized Expert Modules

Ramón Guillamón

learntouseai@gmail.com

Independent Research Concept Manuscript – July 2026

Research status. This manuscript presents an unvalidated research idea and a proposed experimental program. No empirical results, benchmark improvements, speedups, or claims of demonstrated superiority are reported. All performance statements are hypotheses to be tested. The contribution is a formalization of a possible architecture and a falsifiable evaluation plan.

Abstract

Large language models are commonly deployed as monolithic checkpoints or as sparsely activated Mixture-of-Experts (MoE) systems whose experts are designed and trained within a shared architecture. This paper proposes *Neural Capsules*, a research framework in which specialized neural capabilities are represented as independently deployable, quantized expert modules connected to a common backbone through learned input/output interfaces. Capsules are intended to be added, removed, cached, updated, and stored independently, with a router selecting a small active subset under both quality and hardware constraints. The proposal deliberately treats GGUF as an implementation and distribution container rather than as a mathematical primitive. The mathematical object is the expert capsule: a parameterized transformation plus learned projections, routing metadata, provenance, and a compatibility contract. We formalize capsule composition, storage-aware sparse routing, memory and I/O costs, and a criterion for *non-destructive capability expansion*: adding a new expert should improve a target domain while preserving performance on previously supported domains. We distinguish three increasingly difficult settings: same-backbone adapters, distilled experts, and capsules derived from heterogeneous source models. We then define a staged experimental program designed to falsify the central hypotheses on consumer hardware. The proposal builds on, rather than replaces, prior work in sparse MoE, modular post-training, LoRA expert routing, and expert offloading. Its open research question is whether portable, independently maintained, quantized expert modules can become a practical unit of capability composition across model families without requiring full-system retraining.

Keywords: modular language models, mixture of experts, GGUF, quantization, expert routing, model composition, continual learning, edge inference, model offloading, modular post-training.

1 Introduction

Modern language models concentrate broad capabilities inside large parameter sets. Sparse Mixture-of-Experts architectures show that total parameter capacity can be much larger than the parameters activated for a token. Switch Transformers demonstrated trillion-parameter sparse models with conditional computation [1]. Mixtral routes each token to two of eight feed-forward experts, exposing

substantially more total capacity than is active at inference [2]. DeepSeek-V3 further scales this pattern to 671B total parameters with 37B activated per token [3].

This scaling pattern solves one problem but leaves another: model capabilities are still typically packaged, trained, versioned, and distributed as a tightly coupled system. Replacing one expert, importing a capability from a separately trained model, or installing a new domain capability after deployment is not generally equivalent to installing a software plugin. Recent work has moved toward modularity. Branch-Adapt-Route (BAR) trains domain experts independently and composes them with lightweight routing, explicitly targeting cheaper updates and reduced catastrophic forgetting [7]. LoRA-Mixer routes among modular LoRA experts and can deploy pre-trained frozen adapters from external repositories [5]. These works substantially narrow the novelty space for any new proposal based only on "independent experts" or "dynamic routing."

At the systems level, expert placement has become equally important. MoE-Infinity studies expert tracing, prefetching, and caching for offloaded MoE inference [4]. FlashMoE places inactive experts on SSD and uses learned cache replacement to reduce I/O on memory-constrained devices [6]. ReMoE fine-tunes routing to increase short-horizon expert reuse and improve cache locality [8]. Consequently, simply proposing "experts on SSD selected by a router" is also insufficient as a standalone contribution.

The research question considered here is narrower and more compositional:

> *Can a language-model system expose a stable neural interface through which independently maintained, quantized expert modules can be installed or removed after the fact, potentially after being derived from heterogeneous source models, while preserving existing capabilities and while accounting explicitly for memory and storage costs at routing time?*

We call such a module a **Neural Capsule**. A capsule is not a GGUF file mathematically. It is a neural transformation plus interface maps and metadata. GGUF is proposed as one practical deployment representation because the format is designed for model inference, metadata extensibility, quantization, fast loading, and memory mapping [10]. Current `llama.cpp` tooling also supports separately loaded GGUF LoRA adapters, demonstrating that sidecar neural artifacts in GGUF are already operationally plausible [11].

The intended contribution of this paper is therefore not a claim to invent MoE, modular post-training, LoRA routing, or offloading. It is a *research architecture and evaluation program* for treating a capability-bearing neural module as a portable deployment object with a standardized latent contract and a measurable installation cost.

2 Scope and Claims

This proposal makes four scoped claims of interest, all currently unproven.

1. **Capsule abstraction.** A useful unit of model capability may be represented as an independently packaged expert transformation with explicit input/output interface maps rather than as arbitrary raw weights from a full model.
2. **Incremental composition.** A new capsule may be installable with substantially less training than re-training or jointly fine-tuning all existing capabilities, while retaining previous task performance.

3. **Heterogeneous derivation.** Some useful capabilities from a source model with a different internal architecture may be transferable into a capsule compatible with a target backbone through distillation and representation alignment, even though arbitrary cross-model weight splicing is not valid.
4. **Storage-aware execution.** Routing should optimize not only predicted expert utility but also expert residency, transfer latency, and cache behavior when capsules exceed fast-memory capacity.

The paper does **not** claim that arbitrary GGUF models can be directly connected, merged, summed, or treated as mutually compatible experts. It does not claim that aggregate stored parameters are equivalent to a dense model of the same size. It does not claim that quantization or modularity is intrinsically intelligence-enhancing.

3 Related Work

3.1 Sparse Mixture-of-Experts

MoE architectures conditionally activate subsets of model parameters. Switch Transformers simplified routing and demonstrated sparse scaling to the trillion-parameter regime [1]. Mixtral uses eight feed-forward experts per layer and routes tokens to two experts [2]. DeepSeek-V3 uses a large-scale MoE architecture with a small active fraction relative to total parameters [3]. Neural Capsules inherit the principle

$$P_{\text{total}} \gg P_{\text{active}}, \tag{1}$$

but do not treat this inequality as a novel contribution.

3.2 Modular and Independently Trained Experts

BAR is especially relevant because it trains domain experts separately and composes them using an MoE architecture and lightweight router training, targeting modular updates without degrading prior domains [7]. This directly overlaps with the goal of incremental capability extension. Neural Capsules therefore focus on a deployment contract, post-hoc capsule installation, heterogeneous source derivation, quantized packaging, and hardware-aware execution rather than claiming novelty for independent expert training itself.

LoRA-Mixer combines LoRA with expert routing and supports pre-trained frozen LoRA modules obtained externally [5]. This provides an important baseline for the easiest form of capsules: modules attached to a common backbone. The harder proposed setting is to derive a compatible capsule from a source model that does not share the same hidden dimensions, tokenizer, or parameterization.

3.3 Expert Offloading, Caching, and Storage Locality

MoE-Infinity exploits expert activation patterns for request-level tracing, caching, and prefetching [4]. FlashMoE explicitly stores inactive experts on SSD and learns cache replacement behavior for memory-constrained inference [6]. ReMoE fine-tunes the router to favor recently selected experts, increasing reuse and reducing storage transfers [8]. Neural Capsules should be evaluated against these approaches where applicable. The proposed routing objective is thus a generalization for a capsule runtime, not a claim that storage-aware routing is new in isolation.

3.4 GGUF as a Deployment Substrate

GGUF is an extensible binary model format designed for inference with GGML-based executors. Its specification includes typed metadata, tensor descriptors, tensor data, quantization metadata, and mmap-oriented layout [10]. The format supports model files, shards, and auxiliary artifact types; `llama.cpp` also supports GGUF LoRA adapters that are loaded separately from the base model [11]. These properties motivate GGUF as a practical transport for capsules, but the proposed architecture is format-independent.

4 Problem Formulation

Let a deployed language system be

$$\mathcal{M} = (B, R, \mathcal{C}, \Pi), \tag{2}$$

where:

- B is a backbone model that defines a stable working representation;
- R is a router;
- $\mathcal{C} = \{C_1, \ldots, C_N\}$ is the installed capsule set;
- Π is the runtime policy governing memory placement, caching, loading, and eviction.

For a hidden state $h \in \mathbb{R}^d$ produced by the backbone, capsule i is defined as

$$C_i = (E_i, P_i, Q_i, \theta_i, m_i), \tag{3}$$

where E_i is the expert computation, θ_i its parameters, P_i maps the backbone representation into the expert's working space, Q_i maps back to the shared space, and m_i contains routing and deployment metadata.

If expert i uses internal dimension d_i, then

$$P_i : \mathbb{R}^d \to \mathbb{R}^{d_i}, \tag{4}$$

$$E_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}, \tag{5}$$

$$Q_i : \mathbb{R}^{d_i} \to \mathbb{R}^d. \tag{6}$$

The capsule response is

$$c_i(h) = Q_i E_i(P_i h; \theta_i). \tag{7}$$

A simple residual composition rule is

$$h' = F_B(h) + \sum_{i \in S(h)} \alpha_i(h) c_i(h), \tag{8}$$

where F_B is the backbone transformation at the insertion point, $S(h)$ is a sparse set selected by the router, and α_i are normalized gates.

Equation 8 should be interpreted as a design family rather than a commitment to layer-wise insertion.

A capsule may instead operate at a block boundary, on a pooled sequence representation, as an adapter branch, or at a task-level controller. The experimental program should determine which granularity is viable.

5 System Architecture

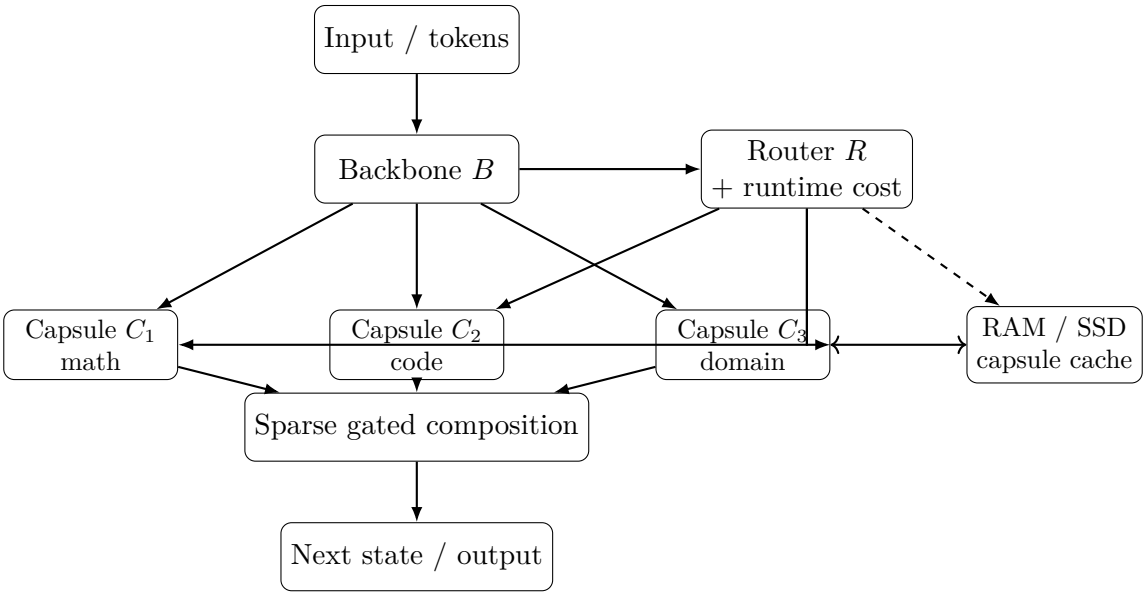

Figure 1: Conceptual Neural Capsule runtime. The router considers both predicted capability utility and deployment cost. Capsules may be resident, memory-mapped, prefetched, loaded, or evicted independently.

The architecture separates four concerns that are often coupled:

1. **Capability representation**: what the expert computes.
2. **Compatibility**: how its representation connects to the backbone.
3. **Selection**: when the expert should be used.
4. **Placement**: where its parameters reside and how costly activation will be.

This separation is central to the proposal. A capsule is not considered "installed" merely because a file is present. Installation requires a validated compatibility contract and routing behavior.

6 Capsule Compatibility Contract

A portable capsule requires more than tensors. We propose a logical compatibility record

$$m_i = (v, d, d_i, \mathcal{I}, q_i, \ell_i, p_i, s_i, \rho_i), \tag{9}$$

where:

- v is the capsule interface version;

- d is the required backbone interface dimension;
- d_i is the capsule internal dimension;
- $\mathcal{I}$ identifies the insertion point or interface type;
- q_i describes quantization;
- ℓ_i describes supported languages or domains;
- p_i records provenance and source-model lineage;
- s_i is serialized size;
- ρ_i contains validation statistics and routing priors.

When GGUF is used, these fields could be encoded in a research-specific metadata namespace such as `neural_capsule.*`. This namespace is proposed here and is **not** asserted to be an existing GGUF standard. GGUF is attractive because its specification explicitly supports extensible key-value metadata and tensor storage [10].

7 Three Levels of Capsule Difficulty

The proposal should be tested in increasing order of difficulty.

7.1 Level I: Same-Backbone Adapter Capsules

The simplest capsule is a LoRA or adapter trained on the same backbone. Compatibility is largely guaranteed by construction. The capsule may be represented as low-rank matrices

$$\Delta W_i = B_i A_i, \tag{10}$$

where $A_i \in \mathbb{R}^{r \times d_{in}}$ and $B_i \in \mathbb{R}^{d_{out} \times r}$. Current `llama.cpp` tooling can load multiple GGUF LoRA adapters separately, making this level a realistic systems prototype [11].

This level is not expected to establish strong scientific novelty; it validates the runtime, packaging, routing, cache accounting, and non-destructive installation protocol.

7.2 Level II: Distilled Expert Capsules

A source model T_i acts as a teacher for a compact expert E_i that is explicitly trained to operate inside the target backbone interface. Given teacher distribution $q_i(y|x)$ and capsule-augmented student distribution $p_i(y|x)$, a distillation objective may be

$$\mathcal{L}_{KD} = \tau^2 \, \mathrm{KL}\left(q_i^{(\tau)}(\cdot|x) \,\|\, p_i^{(\tau)}(\cdot|x)\right), \tag{11}$$

combined with supervised task loss

$$\mathcal{L}_{cap} = \mathcal{L}_{task} + \lambda_{KD}\mathcal{L}_{KD} + \lambda_{reg}\mathcal{L}_{reg}. \tag{12}$$

This setting asks whether a capability can be exported into a small target-compatible module without copying arbitrary internal weights.

7.3 Level III: Heterogeneous Source-Derived Capsules

The strongest version begins with source model T_i whose tokenizer, hidden dimension, attention structure, normalization, and training history differ from the backbone. Direct tensor addition or layer splicing is generally undefined or semantically unjustified. Instead, the proposed pipeline is

$$T_i \xrightarrow{\text{behavior/representation transfer}} C_i \xrightarrow{\text{alignment}} B. \tag{13}$$

One possible alignment loss uses paired prompts and a learned representation target $z_i(x)$:

$$\mathcal{L}_{align} = \mathbb{E}_x \left[\| Q_i E_i(P_i h_B(x)) - z_i(x) \|_2^2 \right]. \tag{14}$$

However, the definition of z_i is itself a research problem. It could be a distilled residual direction, a projected teacher representation, or a task-conditioned target learned through output distillation. The feasibility of Level III is therefore the central high-risk hypothesis of this paper.

8 Routing with Capability and Hardware Costs

Let $u_i(h)$ be the router's estimate of the task utility of capsule i. A conventional sparse router may choose

$$S(h) = \text{TopK}_i \; u_i(h). \tag{15}$$

For memory-constrained deployment, we instead define a cost-adjusted score

$$r_i(h,t) = u_i(h) - \beta \widehat{T}_i^{load}(t) - \gamma \mathbb{1}[i \notin K_t] - \eta \, \sigma_i(t), \tag{16}$$

where K_t is the resident capsule cache, $\widehat{T}_i^{load}$ is estimated load time, and $\sigma_i(t)$ is an optional switching or eviction penalty. Then

$$S_t = \text{TopK}_i \; r_i(h_t, t). \tag{17}$$

The router may be trained with

$$\mathcal{L}_{router} = \mathcal{L}_{quality} + \lambda_{io}\mathcal{L}_{io} + \lambda_{bal}\mathcal{L}_{balance} + \lambda_{switch}\mathcal{L}_{switch}. \tag{18}$$

Equation 16 should not be interpreted as a novelty claim for cache-aware routing by itself. ReMoE and related systems already demonstrate that routing can be modified to improve expert reuse under memory constraints [8]. The purpose here is to make storage cost part of the capsule interface and evaluation contract.

9 Memory, Storage, and I/O Model

Let:

- M_B be backbone resident memory;
- M_{KV} be KV-cache memory;

- M_R be router memory;
- M_{PQ} be interface-projection memory;
- S_i be the serialized/resident size of capsule i;
- K_t be the set of resident capsules at time t.

Fast-memory use is approximated by

$$M_{fast}(t) = M_B + M_{KV}(t) + M_R + M_{PQ} + \sum_{i \in K_t} S_i. \tag{19}$$

Total model storage is

$$M_{storage} = M_B^{disk} + \sum_{i=1}^{N} S_i + M_{meta}. \tag{20}$$

This distinction is essential. A system may store trillions of aggregate parameters while keeping only a small fraction resident, but this does not make its compute or behavior equivalent to a dense trillion-parameter model.

For an expert absent from fast memory, a first-order loading model is

$$T_i^{load} \approx L_{storage} + \frac{S_i}{B_{eff}}, \tag{21}$$

where $L_{storage}$ is fixed access/dispatch latency and B_{eff} is effective transfer bandwidth after software and contention overhead.

The miss cost for a routing step is

$$T_{miss}(t) \approx \sum_{i \in S_t \setminus K_t} \left(L_{storage} + \frac{S_i}{B_{eff}} \right), \tag{22}$$

subject to overlap from prefetching, parallel transfers, and compute. A more realistic implementation should measure rather than assume these quantities.

10 Incremental Capability Expansion

The central behavioral property is **non-destructive capability expansion**. Let $\mathcal{M}_N$ be a system with N installed capsules and C_{N+1} a new capsule. Define

$$\mathcal{M}_{N+1} = \text{Install}(\mathcal{M}_N, C_{N+1}). \tag{23}$$

Let D_{new} be a benchmark for the new capability and D_{old} the union of previously supported domains. Define capability gain

$$G_{new} = \text{Perf}(\mathcal{M}_{N+1}, D_{new}) - \text{Perf}(\mathcal{M}_N, D_{new}), \tag{24}$$

and retention change

$$R_{old} = \text{Perf}(\mathcal{M}_{N+1}, D_{old}) - \text{Perf}(\mathcal{M}_N, D_{old}). \tag{25}$$

A successful installation should satisfy, for chosen practical thresholds $\epsilon > 0$ and $\delta \geq 0$,

$$G_{new} \geq \epsilon, \tag{26}$$

$$R_{old} \geq -\delta. \tag{27}$$

The choice of ϵ and δ must be declared before evaluation. This makes the claim falsifiable and prevents post-hoc relabeling of small gains as success.

A stronger metric can normalize gain by installation cost:

$$\Gamma = \frac{G_{new}}{C_{train} + \lambda C_{storage} + \mu C_{latency}}, \tag{28}$$

where C_{train} is additional training compute, $C_{storage}$ is deployment footprint, and $C_{latency}$ is measured runtime penalty.

11 Capsule Installation Protocol

A proposed installation process is:

1. **Acquire or train source capability.** Select a source model, adapter, or domain dataset.
2. **Derive capsule.** Train or distill E_i, P_i, and Q_i while freezing the deployed backbone where possible.
3. **Validate interface.** Verify shape, insertion point, numerical range, tokenizer assumptions, and output stability.
4. **Quantize.** Produce one or more quantization variants and measure quality loss.
5. **Package.** Store tensors plus provenance and compatibility metadata, optionally in GGUF.
6. **Router adaptation.** Train only the router or a small routing head to recognize the new capsule.
7. **Regression evaluation.** Test both D_{new} and all declared D_{old} domains.
8. **Runtime qualification.** Measure memory, I/O, cache behavior, and latency under realistic request sequences.

A capsule should not be admitted to the active registry until all steps pass predetermined thresholds.

12 Proposed Experimental Program

12.1 Research Questions

RQ1 Can dynamically routed same-backbone capsules match or exceed static adapter selection while keeping inactive modules outside fast memory?

RQ2 Can a new domain capsule be installed with negligible regression on prior domains without retraining existing capsules?

RQ3 Can useful capability be distilled from a heterogeneous source model into a target-compatible capsule with a favorable quality-to-size ratio?

RQ4 Does storage-aware routing improve end-to-end latency and throughput relative to quality-only routing at matched task quality?

RQ5 How does quantization affect capsule routing, specialization, and cross-domain interference?

12.2 Phase A: Runtime Proof of Concept

Use one compact open backbone and three same-backbone LoRA/adapters, for example general, mathematics, and code. Keep the backbone fixed. Package each adapter independently. Compare:

1. base model only;
2. manually selected single adapter;
3. quality-only router;
4. storage-aware router;
5. all adapters resident vs constrained cache vs SSD-backed loading.

The purpose of Phase A is not to prove heterogeneous composition. It tests the runtime abstraction and instrumentation.

12.3 Phase B: Non-Destructive Expansion

Start with capsules $\{C_{math}, C_{code}\}$, evaluate, then install C_{legal} or another clearly distinct domain capsule. Freeze all previous expert weights. Only the new capsule and minimal router parameters may be trained. Report G_{new} and R_{old} exactly as defined above.

12.4 Phase C: Heterogeneous Source Derivation

Choose a teacher model from a different family and a smaller target backbone. Distill a domain capability into a capsule that plugs into the target system. Compare against:

- target backbone without capsule;
- sequential fine-tuning;
- target-specific LoRA;
- knowledge distillation into the full target model;
- modular LoRA routing where applicable;
- a BAR-inspired independent-expert baseline where feasible.

The decisive question is whether the capsule retains enough of the teacher's domain advantage to justify its modular deployment cost.

12.5 Phase D: Storage-Constrained Execution

Construct a capsule library whose total footprint intentionally exceeds available RAM/unified memory. Replay mixed-domain request traces and measure:

- time-to-first-token;
- time per output token;
- tokens per second;
- cache hit rate;
- expert/capsule load count;
- bytes read from storage;
- peak fast-memory usage;
- routing quality;
- task accuracy under latency-aware routing.

The relevant baselines include LRU/LFU-style caching, always-resident subsets, quality-only routing, and prior offloading strategies where implementable. FlashMoE and MoE-Infinity provide important systems reference points [6, 4].

13 Suggested Benchmarks and Metrics

A minimal cross-domain suite may include mathematics (e.g., GSM8K/MATH-style tasks), code generation (e.g., HumanEval/MBPP-style tasks), factual or professional knowledge subsets, and one specialized domain benchmark. The exact datasets must be selected with licensing and contamination analysis before experiments.

The core metrics should include:

Metric	**Purpose**
Task score	Measures domain capability.
Old-domain retention R_{old}	Detects destructive interference after installation.
New-domain gain G_{new}	Quantifies value of the added capsule.
Trainable parameters per installation	Measures update efficiency.
Training FLOPs / wall-clock	Measures adaptation cost.
Serialized capsule size	Measures distribution/storage cost.
Peak fast memory	Tests constrained deployment.
Cache hit rate	Measures placement effectiveness.
Storage bytes read	Directly measures I/O pressure.
TTFT / TPOT / throughput	Captures user-visible inference cost.
Routing accuracy / calibration	Measures expert selection quality.
Quantization degradation	Separates packaging efficiency from model quality.

14 Ablation Studies

At minimum, the following ablations are needed:

1. remove P_i and Q_i when dimensions match;
2. freeze vs train the backbone;
3. top-1 vs top-2 vs soft composition;
4. route per request, per sequence segment, per token, or per layer;
5. quality-only vs cache-aware vs latency-aware routing;
6. FP16/BF16 vs several quantization levels;
7. shared-backbone LoRA capsule vs distilled capsule;
8. single teacher vs multiple teacher sources;
9. router retraining from scratch vs incremental router update;
10. varying capsule library size while holding active compute fixed.

These ablations are important because an apparently successful result may come from ordinary ensembling, increased active compute, or hidden backbone fine-tuning rather than capsule modularity.

15 Failure Criteria

The proposal should be considered unsupported, or at least substantially weakened, if one or more of the following occurs consistently:

1. heterogeneous capsule distillation provides no advantage over a same-size target-specific adapter;
2. installing a new capsule requires retraining most existing experts or the backbone;
3. routing overhead and storage misses dominate the compute saved by sparse activation;
4. quantization destroys the specialized capability or destabilizes routing;
5. router changes cause significant regressions on previously supported domains;
6. the system only works when all capsules share the same original backbone, reducing the proposal to a conventional adapter-routing system;
7. aggregate storage capacity grows but usable capability saturates or degrades because experts are redundant or incompatible.

Explicit failure criteria are necessary because the most ambitious claim – cross-family portable neural capabilities – may simply be false at useful efficiency levels.

16 Why Arbitrary GGUF Fusion Is Not the Proposal

A major conceptual distinction is required. Suppose two full models have weight matrices

$$W_A \in \mathbb{R}^{m \times n}, \qquad W_B \in \mathbb{R}^{p \times q}. \tag{29}$$

If $(m, n) \neq (p, q)$, direct addition is undefined. Even if dimensions match, the models may use different learned bases, normalization conventions, tokenizers, positional encodings, and training objectives. Therefore

$$W_A + W_B \tag{30}$$

need not correspond to a meaningful semantic combination. Quantization further complicates arithmetic on serialized weights.

The Neural Capsule proposal instead defines a learned compatibility operation

$$\Phi_i(h) = Q_i E_i(P_i h), \tag{31}$$

so that composition occurs in a declared shared representation. The file format stores the parameters; it does not provide semantic compatibility by itself.

17 GGUF Packaging Proposal

A capsule could be distributed as a GGUF sidecar-like artifact containing:

- expert tensors θ_i;
- projection tensors P_i, Q_i;
- capsule interface version;
- required backbone family or interface hash;
- insertion point identifier;
- quantization scheme;
- expected hidden dimension;
- supported domains/languages;
- source-model provenance and license metadata;
- validation checksum and optional benchmark card;
- routing prior or embedding used for capsule discovery.

Because GGUF supports extensible metadata and mmap-compatible tensor storage, this is structurally plausible [10]. However, executor changes would be required to interpret the proposed capsule semantics. Existing GGUF compatibility must not be assumed.

18 Scaling Thought Experiment

Suppose a library contains N capsules with parameter counts P_i. Aggregate stored parameters are

$$P_{library} = P_B + \sum_{i=1}^{N} P_i. \tag{32}$$

If only $k \ll N$ capsules are active,

$$P_{active}(t) = P_B^{active} + \sum_{i \in S_t} P_i^{active}. \tag{33}$$

For example, 10,000 capsules averaging 500M parameters would represent 5T capsule parameters in storage. Activating four capsules would expose roughly 2B capsule parameters at a time, plus the backbone. This example illustrates a systems scaling property, not an intelligence equivalence. A library of 5T specialized parameters is not interchangeable with a jointly trained dense 5T model.

The research value lies in testing whether capability breadth can increase with $P_{library}$ while compute remains closer to P_{active} and while new capabilities can be installed independently.

19 Security, Provenance, and Licensing

Portable neural modules create supply-chain concerns analogous to software plugins. A practical capsule ecosystem would require:

- cryptographic hashes and signed provenance;
- explicit source-model and dataset lineage where available;
- license compatibility checks for derived capsules;
- adversarial evaluation for hidden triggers or malicious behavior;
- sandboxed loading and strict metadata parsing;
- resource limits to prevent denial-of-service through oversized or malformed artifacts;
- privacy review when capsules are trained from organizational or personal data.

These issues are not secondary: making neural capabilities installable increases both composability and attack surface.

20 Limitations and Open Problems

20.1 Representation Alignment May Be the Hard Problem

Linear projections P_i and Q_i are mathematically convenient but may be insufficient to bridge representational geometries learned by unrelated models. More expressive adapters could improve alignment but may erase the deployment advantages of small capsules.

20.2 Tokenizer and Sequence Semantics

A hidden-state capsule attached to the target backbone can avoid running the source tokenizer at inference, but the training process must still transfer behavior across different tokenizations. Distillation may work at the output distribution level while representation matching may require careful sequence alignment.

20.3 Capability Isolation Is Imperfect

Skills such as mathematics, code, factual knowledge, and instruction following are entangled. A supposedly specialized capsule may rely on broad base capabilities or alter general behavior unexpectedly.

20.4 Routing Can Become a New Bottleneck

As the capsule registry grows, selecting among thousands of modules may itself require hierarchical retrieval, approximate nearest-neighbor search, or learned indexing. Router mistakes may outweigh expert specialization gains.

20.5 Storage Bandwidth Is Not Free Compute

Sparse activation reduces arithmetic but can replace compute cost with data-movement cost. Modern SSD bandwidth is high relative to historical storage, but repeatedly transferring gigabytes of weights remains expensive. FlashMoE and ReMoE demonstrate that cache behavior materially affects practical performance [6, 8].

20.6 Aggregate Parameter Count Can Be Misleading

Total installed parameters should never be presented as if they were jointly optimized or simultaneously usable. Reporting must always separate $P_{library}$, $P_{resident}$, and P_{active}.

21 Expected Scientific Value if the Hypothesis Holds

If experiments support the proposal, the most important result would not be "a larger model on a small computer." It would be a new deployment property:

> *A language-model system can acquire a useful specialized capability by installing a bounded neural artifact, updating only a small routing/interface component, and preserving previously qualified capabilities within declared tolerances.*

Such a property could support long-lived local models, enterprise-specific expert libraries, modular updates, specialized edge deployments, and capability marketplaces. It could also enable model maintenance practices closer to software package management, where components have versions, compatibility constraints, provenance, and regression tests.

The strongest extension would be successful Level III transfer: deriving portable capsules from heterogeneous source models. That result would suggest that some model capabilities can be exported across architectural boundaries without full-model merging. The negative result would also be valuable, because it would clarify the extent to which learned capabilities are inseparable from the representational geometry of their original model.

22 Reproducibility Plan

A credible first publication should release:

1. all training scripts for capsule derivation;

2. router and cache policy code;
3. capsule metadata schema;
4. exact base/source model revisions and licenses;
5. benchmark prompts and evaluation harnesses where licensing permits;
6. memory and I/O instrumentation;
7. raw per-task and per-request measurements;
8. random seeds and hardware/software versions;
9. negative results, not only successful domains;
10. at least one reference implementation on consumer hardware.

No benchmark number should appear in a final empirical version of this paper unless it is reproduced from logged experiments or clearly attributed to cited prior work.

23 Conclusion

This paper proposes Neural Capsules, an unvalidated architecture for incrementally composable language-model capabilities. The proposal deliberately moves away from the idea that GGUF files themselves form mathematical weight matrices. Instead, it defines a capsule as a neural expert with a compatibility interface, learned projections, routing metadata, provenance, quantization, and an independently manageable deployment lifecycle. GGUF is one candidate container because it already supports quantized tensor storage, metadata extensibility, memory mapping, and separately loaded adapter artifacts.

The framework draws heavily on established ideas in sparse MoE, modular post-training, adapter routing, and expert offloading. Its research opportunity lies in the intersection: portable quantized expert artifacts, post-hoc installation, explicit compatibility contracts, heterogeneous source derivation, and hardware-aware routing under a non-destructive capability-expansion criterion.

The proposal is intentionally falsifiable. It will be supported only if a new capsule can deliver meaningful target-domain gain, preserve prior capabilities within predeclared tolerances, require substantially less update effort than monolithic retraining, and remain operationally useful under realistic memory and storage constraints. Until those experiments are performed, Neural Capsules should be treated as a research hypothesis rather than a demonstrated architecture.

Acknowledgment of Research Status

This manuscript is an independent conceptual research proposal. It synthesizes an original line of inquiry with established and recent literature. No affiliation with the cited projects or authors is implied. The architecture has not yet been experimentally validated.

References

[1] W. Fedus, B. Zoph, and N. Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," *arXiv:2101.03961*, 2021. https://arxiv.org/abs/2101.03961

[2] A. Q. Jiang et al., "Mixtral of Experts," *arXiv:2401.04088*, 2024. https://arxiv.org/abs/2401.04088

[3] DeepSeek-AI et al., "DeepSeek-V3 Technical Report," *arXiv:2412.19437*, 2024. https://arxiv.org/abs/2412.19437

[4] L. Xue, Y. Fu, Z. Lu, L. Mai, and M. Marina, "MoE-Infinity: Offloading-Efficient MoE Model Serving," *arXiv:2401.14361*, 2024. https://arxiv.org/abs/2401.14361

[5] W. Li, Z. Song, H. Zhou, Y. Zhang, J. Yu, and W. Yang, "LoRA-Mixer: Coordinate Modular LoRA Experts Through Serial Attention Routing," *arXiv:2507.00029*, 2025. https://arxiv.org/abs/2507.00029

[6] B. Kim, J. Lee, D. Han, H.-J. Yoo, and S. Kim, "FlashMoE: Reducing SSD I/O Bottlenecks via ML-Based Cache Replacement for Mixture-of-Experts Inference on Edge Devices," *arXiv:2601.17063*, 2026. https://arxiv.org/abs/2601.17063

[7] J. Morrison, S. Adhikesaven, A. Bhagia, M. Zaharia, N. A. Smith, and S. Min, "Train Separately, Merge Together: Modular Post-Training with Mixture-of-Experts," *arXiv:2604.18473*, 2026. https://arxiv.org/abs/2604.18473

[8] X. Zhu, X. Liao, T. Jiang, Y. Zhang, L. Wang, and L. Xiao, "ReMoE: Boosting Expert Reuse through Router Fine-Tuning in Memory-Constrained MoE LLM Inference," *arXiv:2605.27081*, 2026. https://arxiv.org/abs/2605.27081

[9] E. Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models," *arXiv:2106.09685*, 2021. https://arxiv.org/abs/2106.09685

[10] GGML Project, "GGUF File Format Specification," GitHub, accessed July 2026. https://github.com/ggml-org/ggml/blob/master/docs/gguf.md

[11] GGML Project, "llama.cpp: LLM inference in C/C++," GitHub, accessed July 2026. The project documents GGUF model loading and separately loaded GGUF LoRA adapters. https://github.com/ggml-org/llama.cpp