Depth Becomes Time: Recurrent Sparse Transformers with Writable Memory, Indexed Expert Retrieval, and Learned Halting

Mohammad Rastegari
Elastix AI
mohammad@elastix.ai

Architecture/theory proposal. This version intentionally reports no new large-scale empirical results; all efficiency claims are stated as testable hypotheses or conditional results.

Abstract

Large language models conventionally bind representational depth, parameter specialization, working state, and inference cost to a fixed stack of distinct Transformer blocks. We study a different factorization in which *depth becomes time*. A compact recurrent core repeatedly attends to immutable token history and a bounded writable scratchpad, routes the contextualized state to a sparse expert library, updates state and memory, and only then decides whether to halt and emit a token. For very large expert banks, we recast routing as maximum-inner-product retrieval: compact hash or quantized codes generate a candidate set, followed by exact re-ranking of only those candidates. The resulting execution loop is

$$\text{attend} \rightarrow \text{retrieve/route} \rightarrow \text{top-}k \text{ experts} \rightarrow \text{state/memory update} \rightarrow \text{halt or recur}.$$

The proposal is deliberately positioned relative to Universal Transformers, Sparse Universal Transformers, MoEUT, Relaxed Recursive Transformers, recurrent-depth language models, Mixture-of-Recursions, PEER, and memory-augmented models: recurrence, sparse experts, adaptive depth, and continuous depth-wise batching are not individually new. The intended contribution is a particular execution contract: an immutable canonical token memory, a bounded writable working state, post-attention retrieval-indexed expert selection, post-update semantic halting, and a training/deployment recipe that treats their interactions as first-class design constraints. We give conditions for bounded memory updates, stable approximate routing, retained-evidence attention, and trajectory agreement; identify repeated full-context attention and cross-token writable state as important efficiency/training bottlenecks; and propose explicit variants for each. Training is presented first as a discrete expected-exit objective and then as an optional continuous-depth/neural-event formulation in which semantic halting is a learned terminal event rather than a numerical ODE tolerance. We also describe checkpoint conversion and the implications for accelerators, batching, and speculative decoding. The central claim is falsifiable: separating stored capacity, latent compute time, and writable state may move the quality–latency–energy frontier, but only if the runtime can exploit conditional execution without losing model quality or pretraining efficiency.

1 Introduction

The dominant Transformer is spatial: a token traverses a fixed sequence of distinct parameterized blocks. Increasing capability therefore tends to increase several coupled quantities at once—unique parameters, layer count, weight traffic, activation state, and latency. Yet the computation demanded by different tokens is heterogeneous. Predictable syntax, retrieval, arithmetic, code, and multi-hop reasoning need not deserve the same amount or kind of computation.

This paper studies a temporal alternative:

> **reuse a powerful core in latent time; retrieve the relevant evidence; invoke only a few expert transformations; preserve bounded writable state; and stop when the token is ready.**

The intended factorization exposes three principal resources and one control policy:

$$\boxed{\text{stored capacity } E} \qquad \boxed{\text{latent computation } T(x)} \qquad \boxed{\text{working state } m} \qquad \boxed{\pi_{\text{halt}}(x,t)}. \tag{1}$$

A large expert library can increase stored capacity without activating every parameter. Recurrent time can increase effective depth without introducing a new copy of the shared core. A bounded scratchpad can preserve intermediate state without rewriting the committed token history.

The high-level idea has substantial precedent. Universal Transformers (UTs) already combine recurrent depth and adaptive computation [7, 8]. Sparse Universal Transformers (SUTs) combine UT-style recurrence with sparse MoE and dynamic halting [10]. MoEUT makes shared-layer Transformers competitive by introducing mixtures of experts in both feed-forward and attention components [11]. Relaxed Recursive Transformers (RRTs) show how pretrained Transformers can be converted into recurrent models, recover depth specialization with loop-specific low-rank adapters, and exploit continuous depth-wise batching [12]. Recurrent-depth language models and Mixture-of-Recursions (MoR) demonstrate that latent recurrent computation and token-adaptive recursive depth remain competitive at contemporary LM scale [13, 14]. Subsequent retrofit work further shows that recurrence can be installed into pretrained language models with a recurrence curriculum [39]. Attractor Models provide a complementary 2026 fixed-point view of iterative refinement and adaptive convergence [40]. Coconut demonstrates a different form of latent-space reasoning by feeding hidden states back as continuous thoughts [38]. PEER independently shows that product-key retrieval can select from more than a million tiny experts [20]. Consequently, the novelty claim here cannot be "recurrence + MoE + halting." The proposal instead asks whether these lines can be made into one coherent autoregressive execution architecture with a stricter memory contract, retrieval-indexed routing, explicit post-update halting semantics, and a systems design that is built around recurrent execution.

The conceptual starting point also connects to *Discovering Neural Wirings* (DNW), which treated architecture as a learned sparse subgraph of a larger computational supergraph rather than a fixed manually layered topology [1]. In the present setting the graph is dynamic: attention selects evidence, indexed routing selects transformations, memory writes modify latent state, recurrent time determines path length, and halting terminates the walk.

The key memory contract. Committed token KVs are immutable evidence. A separate typed scratchpad is writable working state. The *primary baseline is token-local*: the scratchpad persists across recurrent steps used to produce one output token and is reset after that token commits. This keeps the latent loop stateful without introducing an unavoidable recurrence across teacher-forced token positions. A *sequence-scoped* extension carries a small scratchpad across emitted tokens, but it is explicitly treated as a post-training/chunked-memory variant because cross-token writes introduce an additional sequential dependency during pretraining. Neither variant becomes part of the canonical token history. This separation avoids silently rewriting causal evidence while still permitting internal representations and hypotheses to evolve.

The key efficiency caveat. Recurrence is not automatically efficient. Repeating full attention over a long context at every latent step can dominate cost. Large expert banks can make the router itself expensive. Token-dependent halting creates scheduler divergence. The architecture is attractive only if attention reuse/retrieval, indexed routing, expert locality, active-token compaction, and weight residency compensate for the added control flow. We make these conditions explicit rather than treating FLOP counts as sufficient evidence.

Contributions. This paper makes the following architecture-level contributions.

1. It defines an explicit autoregressive recurrent loop with the ordering *attention* $\rightarrow$ *indexed routing* $\rightarrow$ *top-k experts* $\rightarrow$ *state/scratchpad update* $\rightarrow$ *halt*, where one token is emitted only at halt.

2. It separates immutable canonical token KVs from a bounded writable scratchpad, specifies how a newly sampled token is committed exactly once, and distinguishes a training-friendly token-local scratchpad from a sequence-scoped extension.
3. It recasts large-expert routing as retrieval followed by exact candidate re-ranking, and gives route-stability and recurrent-trajectory agreement conditions.
4. It identifies the long-context attention re-scan as a first-class bottleneck and proposes a two-tier evidence-attention variant that keeps the scratchpad always visible while retrieving only a small token-KV subset on later recurrent steps.
5. It gives a training hierarchy: a discrete expected-exit baseline that directly couples halt probabilities to prediction loss, followed by an optional Neural-ODE/Neural-Event formulation for learning continuous latent trajectories and semantic terminal events [3, 4].
6. It gives conversion, hardware, and evaluation recipes that explicitly compare against the closest existing recursive, sparse, adaptive-depth, memory, and retrieval-expert baselines.
7. It maps the architecture onto continuous depth-wise batching—introduced for Recursive Transformers by RRT [12]—and extends that systems abstraction with expert-local sub-batches, indexed routing, paged immutable KVs, and per-request writable state. We do not claim continuous depth-wise batching itself as a new contribution.

2 Closest Prior Art and What Is Actually New

2.1 Learned wiring and recursive computation

DNW learns sparse connectivity inside a large supergraph [1]. Neural ODEs interpret residual depth as continuous dynamics and allow adaptive numerical evaluation [3]; DEQs represent effectively infinite tied depth through a fixed point [6]; and recent Attractor Models revisit implicit/fixed-point refinement for language and reasoning [40]. UTs make Transformer depth recurrent and add per-position adaptive computation [7, 8]; PonderNet provides a probabilistic learned-computation formulation [9]. Looped Transformers show constructively that a constant-depth Transformer placed in a loop can emulate instruction-like operations, program counters, branching, and finite programs [36]. Formal Turing-completeness results for Transformers and UTs are useful expressivity context but do not establish learnability or efficiency [37, 7].

2.2 The closest architectural overlaps: SUT, MoEUT, RRT, and MoR

The strongest novelty challenge comes from prior recursive sparse Transformers. SUT already combines Universal-Transformer recurrence, sparse experts, and dynamic halting [10]. MoEUT combines recurrence with MoE feed-forward and attention modules and introduces grouping/normalization choices that materially improve language-model competitiveness [11]. RRT shows a practical pretrained-model conversion path, demonstrates that loop-specific low-rank adapters recover part of the expressivity lost by strict tying, and already proposes continuous depth-wise batching [12]. MoR combines recursive parameter sharing with token-adaptive depth and selective KV handling [14]. Recent recurrent-depth LMs show that repeated latent computation can improve reasoning without exposing additional chain-of-thought tokens [13], while retrofit work studies installing this behavior into pretrained models [39]. Coconut is related in motivation but distinct architecturally: it feeds latent hidden states back as continuous thoughts rather than repeatedly executing the same recurrent sparse block [38].

Table 1 states the intended differentiation rather than implying those ingredients are absent from prior work.

	Recurrent	Sparse experts	Adaptive depth	Writable state	Indexed route	Immutable/writable split
UT [7]	✓	–	✓	–	–	–
SUT [10]	✓	✓	✓	–	–	–
MoEUT [11]	✓	✓	optional	–	–	–
RRT [12]	✓	–	compatible	–	–	–
MoR [14]	✓	–	✓	–	–	–
PEER [20]	–	✓	–	–	✓	–
This proposal	✓	✓	✓	✓	✓	✓

Table 1: Closest architectural overlap. Checkmarks describe the high-level mechanism, not implementation equivalence. The contribution claimed here is the full execution contract and its training/systems consequences, not the individual ingredients.

2.3 Sparse experts, expert reuse, and retrieval

Sparse MoE decouples stored parameter capacity from active compute [15, 16, 17]. Hash Layers show that discrete hashing can select sparse parameters without a conventional learned router [18]. Product-Key Memory provides efficient retrieval from large learned memories [19]; PEER applies product-key retrieval to more than a million tiny experts [20]. ReXMoE allows expert reuse across neighboring layers, directly challenging the layer-local expert-bank assumption [21]. These works motivate a global reusable expert library, but also imply that "reuse experts across depth" is not itself a novel claim. The new systems question here is how expert retrieval interacts with recurrent trajectories, temporal locality, writable state, and halting.

2.4 Adaptive depth and early exit

CALM, Mixture-of-Depths, LayerSkip, and related early-exit systems demonstrate that computation can be allocated non-uniformly across tokens and depth [26, 27, 28]. Recent post-training systems continue to explore token-level exits and the associated missing-KV problem in causal LMs [29, 44]. The present model differs by making the recurring shared core itself the unit of adaptive depth, rather than exiting from a one-way untied stack.

2.5 Memory as state rather than longer text

Neural Turing Machines and Differentiable Neural Computers establish differentiable read/write external memory [30, 31]. Transformer-XL, Recurrent Memory Transformer, Memorizing Transformers, and Titans provide several forms of recurrence or external memory [32, 33, 34, 35]. The scratchpad here is deliberately narrower: a bounded typed working state separated from the committed autoregressive KV history.

3 Architecture

Let $x_{1:n}$ be the committed token history and

$$\mathcal{M}_{\text{tok}} = \{(k_i^{\text{tok}}, v_i^{\text{tok}})\}_{i=1}^{n} \tag{2}$$

its immutable KV memory. Let

$$\mathcal{M}_{\text{sp}}^{(t)} = \{(k_j^{(t)}, v_j^{(t)})\}_{j=1}^{m} \tag{3}$$

be a bounded writable scratchpad at recurrent step t, and $h_t \in \mathbb{R}^d$ the state of the token currently under computation.

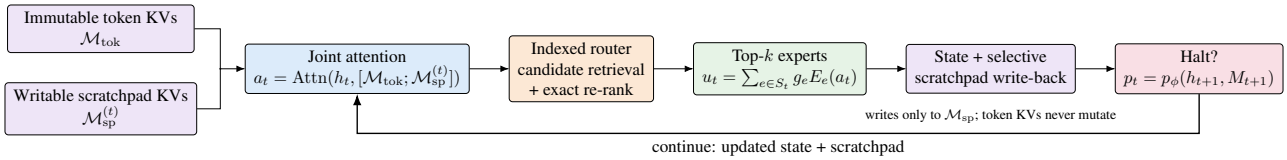

Figure 1: One recurrent step. The halt decision is evaluated only after the expert and memory updates. A token is emitted only when the halt condition fires.

Scratchpad lifecycle. The primary baseline is *token-local*: $\mathcal{M}_{\text{sp}}$ persists across the recurrent steps used to produce one output token and is reset after that token commits. This preserves a clean autoregressive training path and avoids making every teacher-forced position depend on the scratchpad updates of all previous positions. A *sequence-scoped* extension carries a small scratchpad across emitted tokens within one generation episode and resets it only at sequence boundaries. That extension can provide longer-lived hypotheses, but it introduces an outer recurrence across token positions and should therefore be evaluated as a separate post-training or chunk-recurrent regime. Token emission never appends scratchpad state to $\mathcal{M}_{\text{tok}}$.

3.1 Autoregressive commit semantics: what one immutable KV actually stores

A recurrent model with one canonical KV pair per committed token must specify how that pair is created; otherwise "immutable token KVs" hides the very representation-evolution problem the architecture is meant to solve. At generation position n, let the recurrent loop produce terminal state $h_n^\star$ from the already committed prefix $x_{1:n}$. The next-token distribution is

$$p_\theta(x_{n+1} \mid x_{1:n}) = \text{softmax}(W_o h_n^\star). \tag{4}$$

After a token x_{n+1} is selected (ground truth under teacher forcing, sampled/decoded at inference), a lightweight *commit encoder* creates its immutable evidence record once:

$$c_{n+1} = f_{\text{commit}}([e(x_{n+1}); h_n^\star; \rho_{n+1}]), \tag{5}$$

$$k_{n+1}^{\text{tok}} = W_K^{\text{commit}} c_{n+1}, \qquad v_{n+1}^{\text{tok}} = W_V^{\text{commit}} c_{n+1}. \tag{6}$$

The terminal state summarizes the causal prefix before x_{n+1}; the sampled token embedding supplies the newly committed symbol and ρ_{n+1} is a positional code. Thus the record is causal, context-conditioned, and never revised later. The next recurrent token state can be initialized as $h_{n+1,0} = f_{\text{init}}(c_{n+1})$. A cheaper embedding-only commit and a richer small ingest block are important ablations. The key empirical question is whether one canonical evidence representation plus writable working state can replace the layer-specific KV hierarchy of a deep Transformer without losing quality.

For a well-defined autoregressive model, all persistent internal state used at evaluation must be a causal function of the committed prefix. Routing and halting are therefore deterministic (or explicitly treated as model randomness) at deployment; no future token may affect $\mathcal{M}_{\text{tok}}$ or a persistent scratchpad before it is committed.

3.2 Joint attention over immutable history and writable state

Let $\mathcal{M}_t = [\mathcal{M}_{\text{tok}}; \mathcal{M}_{\text{sp}}^{(t)}]$. For head q,

$$q_t^{(q)} = W_Q^{(q)} \,\text{Norm}(h_t), \tag{7}$$

$$\alpha_t^{(q)} = \text{softmax}\left(\frac{q_t^{(q)} (K_t^{(q)})^\top}{\sqrt{d_h}} + b_{\text{type}} \right), \tag{8}$$

$$a_t = h_t + W_O[\alpha_t^{(1)} V_t^{(1)}; \ldots; \alpha_t^{(H)} V_t^{(H)}]. \tag{9}$$

Memory-type embeddings or biases distinguish committed evidence from mutable working state.

3.3 Strict sharing versus phase-conditioned adapters

A serious weakness of fully tied depth is loss of layer-specific transformations, especially in attention. MoEUT and RRT provide direct evidence that grouping or depth-specific adaptation can recover this expressivity [11, 12]. We therefore treat *strict sharing* as the cleanest base model, not a dogma. A practical variant uses $q \ll L$ phase embeddings or low-rank adapters:

$$W_\bullet(t) = W_\bullet^{(0)} + B_{\phi(t)} A_{\phi(t)}, \qquad \phi(t) \in \{1, \dots, q\}. \tag{10}$$

The ablation $q \in \{1, 2, 4, 8\}$ measures how much unique depth is actually needed. This also provides a realistic conversion bridge from an untied teacher.

3.4 Routing and expert update

The router operates on the post-attention state. With low-dimensional query $q_t = U\,\mathrm{Norm}(a_t) \in \mathbb{R}^{d_r}$ and expert keys $k_e \in \mathbb{R}^{d_r}$,

$$s_{t,e} = q_t^\top k_e + b_e, \qquad S_t = \mathrm{TopK}_e(s_{t,e}, k), \qquad g_t = \mathrm{softmax}(s_t|_{S_t}), \tag{11}$$

and

$$u_t = \sum_{e \in S_t} g_{t,e} E_e(\mathrm{Norm}(a_t)). \tag{12}$$

Expert granularity and fair compute normalization. The expert count E alone is not a meaningful capacity claim. Let one expert contain P_e parameters and cost C_e when active. The stored expert capacity is EP_e, while one recurrent step activates approximately kP_e parameters and costs kC_e (plus routing). Comparisons to a dense FFN should therefore match *active* width/compute, e.g. choose $kC_e \approx C_{\mathrm{FFN}}$ at a target step budget. Million-expert routing is plausible only in the fine-grained/tiny-expert regime exemplified by PEER [20]; a million conventional Transformer FFNs is neither a realistic storage claim nor the intended design. For standard-size FFN experts, the practically relevant range may be hundreds to thousands.

A gated residual update is preferable to an unconstrained repeated residual:

$$\tilde{h}_{t+1} = a_t + \gamma_t u_t, \qquad \gamma_t = \gamma_{\max} \sigma(c_t), \tag{13}$$

with small initial $\gamma_{\max}$ or ReZero-like initialization. This reduces recurrent drift and gives the optimizer control over step magnitude.

3.5 Selective scratchpad write-back

For slot j,

$$z_{t,j} = \sigma(f_z(\tilde{h}_{t+1}, v_j^{(t)}, c_{t,j})), \tag{14}$$

$$\hat{v}_{t,j} = f_v(\tilde{h}_{t+1}, c_{t,j}), \tag{15}$$

$$v_j^{(t+1)} = (1 - z_{t,j}) v_j^{(t)} + z_{t,j} \hat{v}_{t,j}. \tag{16}$$

The safest initial design keeps scratchpad *keys fixed or slowly updated and values writable*. Fully writable keys are an ablation because moving addresses and values simultaneously can cause memory thrashing. $c_{t,j}$ may contain attention mass, slot age, write history, and a typed address embedding.

Algorithm 1 Recurrent sparse decoding for one emitted token

Initialize current token state h_0 from the latest committed record; initialize a token-local scratchpad (or load it only in the sequence-scoped extension).
for $t = 0, \ldots, T_{\max} - 1$ **do**
  Attend to immutable token KVs and current scratchpad.
  Retrieve a candidate expert set; exact re-rank; select top-k.
  Execute experts and update h_{t+1}.
  Selectively write scratchpad values (and optionally keys).
  Evaluate post-update halt controller.
  if halt criterion is met **then**
    break
  end if
end for
Emit one next-token distribution; commit one token and create/append its canonical KV once using the commit encoder.
Initialize the next current state from the new committed record; reset the token-local scratchpad, or carry it forward only in the explicit sequence-scoped extension.

3.6 Post-update halting and a corrected differentiable objective

A halt probability is computed *after* the state and scratchpad update:

$$p_{i,t} = \sigma(f_{\text{halt}}(h_{i,t+1}, \mathcal{M}_{\text{sp},i}^{(t+1)})). \tag{17}$$

A common pitfall is to add an expected-step penalty without coupling the halt distribution to prediction quality. Let

$$S_{i,t} = \prod_{u<t}(1 - p_{i,u}), \qquad \pi_{i,t} = p_{i,t} S_{i,t}, \tag{18}$$

with $p_{i,T_{\max}} = 1$ to force termination. If $\ell_{i,t}$ is the token prediction loss from the state after step t, the simplest fully differentiable objective is

$$\mathcal{L}_{\text{exit}} = \frac{1}{N} \sum_i \sum_{t=1}^{T_{\max}} \pi_{i,t} \ell_{i,t} + \lambda_T \frac{1}{N} \sum_i \sum_{t=1}^{T_{\max}} S_{i,t}. \tag{19}$$

The first term teaches the halting head *which exits are accurate*; the second prices expected deployed steps. PonderNet-style priors or ACT-style ponder costs can replace this parameterization [8, 9]. At deployment, a deterministic threshold or calibrated margin converts the learned distribution into discrete halting. Because soft exit training and hard deployment differ, an optional detached-final-state self-distillation term

$$\mathcal{L}_{\text{exitKD}} = \sum_{t<T_{\max}} \omega_t \, \text{KL}\Big(\text{sg}[p_{T_{\max}}^{\text{tok}}] \,\|\, p_t^{\text{tok}}\Big) \tag{20}$$

can make intermediate exits predictive without allowing their losses to dominate the final trajectory. The compute price λ_T can also be randomized or supplied as a budget embedding during training so one checkpoint learns several quality/compute operating points rather than one brittle threshold.

4 Indexed Expert Retrieval: Routing as Search

A sparse MoE avoids evaluating every expert but a conventional router may still compute a dense score for every expert. The retrieval formulation is a *design constraint*, not a property of arbitrary routers: we intentionally use

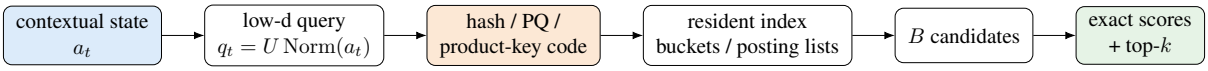

Figure 2: Indexed expert routing. Compact retrieval identifies a small candidate set; exact scoring preserves the original router whenever the true top-k is recalled.

an indexable inner-product scoring function. For this linear/bi-encoder router, the rows of the router matrix are expert keys, so top-k expert selection is a maximum-inner-product search (MIPS) problem. Exact dense routing costs $\Theta(Ed_r)$ per routed state. This is negligible for tens of experts and potentially dominant for thousands or millions.

4.1 Retrieve candidates, then re-rank exactly

Let an index $\mathcal{I}$ return a candidate set C_t of size $B \ll E$:

$$C_t = \mathrm{Retrieve}(q_t; \mathcal{I}), \qquad S_t = \mathrm{TopK}_{e \in C_t}(q_t^\top k_e + b_e, k). \tag{21}$$

Approximation is confined to candidate generation; final router scores are exact on the candidate set.

Proposition 1 (Exact route conditioned on candidate recall)**.** *Let $S_t^\star$ be the exact global top-k set. If $S_t^\star \subseteq C_t$, exact re-ranking inside C_t returns $S_t = S_t^\star$.*

Proof. All globally top-k experts are present and receive the same full-precision scores as in the dense router; no excluded expert can outrank the kth global expert. □

4.2 Hashing and quantized retrieval

For normalized query and expert keys, random-hyperplane LSH uses

$$c(q) = \mathrm{sign}(Pq) \in \{-1, +1\}^b, \qquad c_e = \mathrm{sign}(Pk_e), \tag{22}$$

with collision probability $1 - \theta(q, k_e)/\pi$ for one bit [22]. Multi-table and multi-probe search expose explicit latency/recall knobs. For general unnormalized MIPS, asymmetric LSH can be used [23]. Alternatives include product-key retrieval [19, 20], inverted/PQ search and anisotropic quantization [24], and binary quantization with explicit error control [25]. The right choice is hardware-dependent; the architectural interface is simply *candidate retrieval → exact re-rank*.

4.3 Ranking stability and recurrent compounding

Let $s_{(k)}$ and $s_{(k+1)}$ be the exact boundary scores and $\Delta_k = s_{(k)} - s_{(k+1)}$.

Proposition 2 (Top-k stability under bounded score error)**.** *If $|\hat{s}_e - s_e| \le \epsilon$ for all experts relevant to the top-k boundary and $\Delta_k > 2\epsilon$, the boundary selected/excluded ordering cannot flip.*

Proof. The kth selected score is at least $s_{(k)} - \epsilon$ and the best excluded score is at most $s_{(k+1)} + \epsilon$. □

Approximation errors can compound across recurrent time.

Proposition 3 (Trajectory agreement under per-step candidate recall)**.** *Suppose the indexed router contains the exact global top-k at recurrent step t with conditional probability at least $1 - \delta_t$. Then for a T-step trajectory,*

$$\Pr[\textit{all indexed routes match dense routes through } T] \ge 1 - \sum_{t=1}^{T} \delta_t. \tag{23}$$

Proof. Apply the union bound to route-miss events across recurrent steps. □

Thus hard tokens that recur many times require stricter per-step route recall or confidence-triggered fallback. This is a distinctive requirement of recurrent retrieval: a 99% single-step recall may be inadequate for a long trajectory.

4.4 Training the index without destabilizing the model

A practical schedule is:

1. train an exact dense router and expert bank first;
2. learn U, expert keys, and compact codes to preserve teacher top-k and top-k margins;
3. enable candidate retrieval but keep exact candidate re-ranking;
4. mine hard negatives near the teacher routing boundary;
5. refresh expert codes on a slower clock than model weights;
6. add a confidence fallback that broadens probes or uses the dense router when route margin or candidate coverage is poor;
7. only then optimize index locality and expert placement for hardware.

Load balancing must be measured *after* candidate generation. Approximate retrieval can otherwise starve experts even when the dense teacher router is balanced.

5 Efficiency: Where the Architecture Wins and Where It Can Fail

Let one recurrent step cost

$$C_{\text{step}}(n) = C_A(n, m) + C_R(E, B, d_r) + kC_E + C_W + C_H, \tag{24}$$

so

$$C_{\text{rec}}(x) = \sum_{t=1}^{T(x)} C_{\text{step}}^{(t)}. \tag{25}$$

A fixed L-layer baseline costs

$$C_{\text{fixed}}(x) \approx \sum_{\ell=1}^{L} \left[C_A^{(\ell)}(n) + C_F^{(\ell)} \right]. \tag{26}$$

No generic inequality makes the recurrent model cheaper at matched quality. Its case rests on *weight reuse, sparse active experts, heterogeneous $T(x)$, bounded working memory, and a runtime that exploits the conditional path*.

5.1 The long-context attention bottleneck

The largest architectural issue in a naive implementation is repeated attention over all n committed tokens. If every recurrent step performs full attention, the attention term scales roughly with $T(x)n$ during decode, which can erase the savings from adaptive expert execution.

We therefore distinguish:

Full-evidence recurrence.
Every step attends to all token KVs plus scratchpad. This is the simplest and highest-fidelity baseline.

Two-tier recurrence.
The first recurrent step uses ordinary attention; later steps attend to all m scratchpad slots plus a retrieved subset $R_t \subset \mathcal{M}_{\text{tok}}$ of $r \ll n$ evidence KVs, optionally including a fixed recent window.

Summary recurrence.
Later steps reuse a context summary from the first pass and repeatedly update only state/scratchpad, periodically refreshing full evidence.

The two-tier variant gives

$$C_A^{\text{two-tier}} \approx C_A(n+m) + \sum_{t=2}^{T(x)} C_A(r+m), \qquad r \ll n. \tag{27}$$

This is not required for the architectural thesis, but it is likely required for compelling long-context efficiency. The relevant ablation is quality versus (r, T), not just T.

The retrieval approximation admits a simple sanity bound. Let full attention be $a = \sum_i \alpha_i v_i$, let R retain attention mass $\sum_{i \in R} \alpha_i = 1 - \epsilon$, and let $a_R = \sum_{i \in R} \alpha_i v_i / (1-\epsilon)$ be attention renormalized on the retained evidence. If $\|v_i\| \le V$ for all values, then

Proposition 4 (Retained-evidence attention bound)**.** $\|a - a_R\| \le 2\epsilon V$.

Proof. Write the full output as $(1-\epsilon)a_R + \epsilon a_{\bar{R}}$, where $a_{\bar{R}}$ is the normalized average over omitted values. Then $a - a_R = \epsilon(a_{\bar{R}} - a_R)$ and both normalized averages have norm at most V. □

The bound is deliberately coarse, but it identifies the right retrieval diagnostic: omitted *attention mass*, not only token-retrieval recall. A learned evidence retriever should therefore be evaluated by retained attention mass and end-to-end quality in addition to ANN recall.

5.2 Why the scratchpad changes recurrence

A tied block without explicit writable state repeatedly transforms a residual stream while re-reading immutable evidence. With scratchpad M_t,

$$(h_{t+1}, M_{t+1}) = F_\theta(h_t, M_t; \mathcal{M}_{\text{tok}}), \tag{28}$$

so repeated visits to shared parameters can implement different operations because controller state and memory evolve. The analogy is a processor executing the same instruction hardware over different machine states: useful complexity lies in state and control path, not only in unique physical layers.

5.3 Pretraining parallelism is a first-class constraint

A sequence-scoped writable scratchpad creates a dependency from token position n to $n+1$ in addition to the usual causal mask. If updated after every committed token, it prevents the standard trick of training all teacher-forced positions of a sequence fully in parallel. This is a potentially decisive cost, not an implementation detail.

The token-local baseline avoids this outer dependency: at recurrent step t, all sequence positions can execute the same shared block in parallel with independent per-position working state. Its activation footprint is approximately $O(nmd)$ for m scratchpad slots per position, so m must remain small and checkpointing/recomputation

may still be necessary. For long pretraining sequences, practical options include low-rank/value-only slots, training scratchpads only for sampled target positions, or chunking the sequence.

Sequence-scoped memory should therefore be introduced only when its benefit is measured. Plausible regimes are post-training, truncated/chunked BPTT, sparse memory updates at chunk boundaries, or distillation from a slower recurrent teacher. The paper treats token-local working memory as the primary architecture and sequence-scoped memory as a higher-cost extension.

5.4 A falsifiable efficiency thesis

The strongest defensible claim is:

> *When token difficulty is heterogeneous, useful computation is iterative, expert specialization is sparse, and repeated evidence access can be made sufficiently cheap, separately scaling stored capacity, recurrent time, and writable state may achieve a better quality–latency–energy frontier than encoding all three as fixed unique depth.*

This thesis fails if easy tokens do not halt early, hard tokens do not benefit from extra steps, approximate routes perturb trajectories, scratchpad interventions do not matter, repeated attention dominates, or wall-clock utilization collapses.

6 Conditional Analysis

The purpose of this section is to separate properties that follow from the construction from hypotheses that require experiments.

Lemma 1 (Bounded gated memory)**.** *Assume $\|\hat{v}_{t,j}\| \le B$, $\|v_{t,j}\| \le B$, and $z_{t,j} \in [0,1]$. Then*

$$v_{t+1,j} = (1 - z_{t,j})v_{t,j} + z_{t,j}\hat{v}_{t,j}$$

preserves $\|v_{t+1,j}\| \le B$ for any norm with a convex unit ball.

Proof. The update is a convex combination of two points in the radius-B norm ball. □

Proposition 5 (Contractive joint-state stability)**.** *Let $s_t = (h_t, M_t)$ and $s_{t+1} = F_\theta(s_t; x)$. If for fixed committed history x, F_θ is ρ-Lipschitz in s with $\rho < 1$, then a unique fixed point exists and*

$$\|s_t - s^\star\| \le \rho^t \|s_0 - s^\star\|.$$

Proof. Banach's fixed-point theorem. □

Contraction is sufficient, not necessary, and is not the intended semantics of every useful trajectory. The result motivates residual gating, Jacobian control, and bounded writes when long recurrences become unstable.

Observation 1 (Adaptive-compute accounting)**.** *If a recurrent baseline always takes L steps of cost c and an adaptive policy preserves the same output-quality constraint while taking $T(x) \le L$, then its expected step compute is $c\mathbb{E}[T] \le cL$. The substantive question is not the inequality but whether quality and hardware efficiency survive the adaptive policy.*

Observation 2 (Stored capacity can grow faster than active expert compute)**.** *For fixed k, expert granularity P_e, and candidate budget B, increasing the stored library from E to E' experts increases expert parameters from EP_e to $E'P_e$ while the expert-evaluation term remains kC_e. Any practical decoupling therefore depends on keeping retrieval/index cost and expert-weight movement subdominant. Indexed routing addresses only the search term; it does not make expert storage or bandwidth free.*

Observation 3 (Finite programmable-computation interpretation)**.** *Under idealized assumptions—addressable bounded memory, a state that carries finite control, experts that implement a finite instruction set, and a router that selects instructions—the recurrent architecture can simulate any* finite *execution of a register-machine program step-by-step. This follows by encoding the machine configuration in* (h_t, M_t) *and routing each instruction to an expert. This is related to Neural Turing Machines and Looped Transformers [30, 36]; it is a representational analogy, not a claim of learnability or an unbounded practical Turing machine.*

Conjecture 1 (Latent expert trajectories become reusable programs)**.** *If experts specialize and scratchpad state is causally used, expert-transition motifs across recurrent time will predict semantic operation classes more strongly than marginal expert identity. This can be tested by transition mutual information, causal rerouting, and scratchpad interventions.*

7 Depth as Latent Time: Training From Scratch

A stable recipe should not ask optimization to discover useful recurrence, sparse expert specialization, memory semantics, indexed retrieval, and halting simultaneously.

7.1 Stage 1: train a recurrent backbone before making it adaptive

Start with fixed recurrent depth T_0, a dense FFN or small exact expert bank, no scratchpad writes, and recurrent-step or phase embeddings. Randomize the executed depth for a fraction of batches and apply auxiliary token loss at intermediate exits:

$$\mathcal{L}_{\text{depth}} = \mathcal{L}_{\text{LM}}^{(T)} + \alpha \sum_{t<T} w_t \mathcal{L}_{\text{LM}}^{(t)}. \tag{29}$$

Use pre-norm, gated residuals, gradient clipping, and optionally a Jacobian/spectral penalty if states diverge with depth.

7.2 Stage 2: expertize with exact routing

Introduce top-k experts only after the recurrent backbone is predictive. Initialize experts as noisy copies, transplanted teacher FFNs, or clustered teacher transformations. Use load-balancing and router-logit regularization. Balance should be conditioned on recurrent phase or measured per-depth as well as globally; a router that is globally balanced can still collapse at individual recurrent steps.

7.3 Stage 3: warm the scratchpad

Add learned scratchpad keys and zero or learned-seed values as read-only memory. The primary pretraining baseline uses token-local scratchpads so positions remain parallel across the sequence. Then enable sparse value writes while keys remain fixed. Only after stable use is established should key rewrites be attempted. Anneal a write penalty and evaluate memory with causal ablations rather than only reconstruction losses. Sequence-scoped persistence should be a later post-training stage, with its training-throughput cost reported explicitly.

7.4 Stage 4: discrete adaptive halting is the primary baseline

Use Eq. 19. Begin with halt biases that place probability mass near the already-trained depth T_0, enforce a short minimum-depth curriculum, then gradually increase λ_T. A complete objective can be

$$\mathcal{L} = \mathcal{L}_{\text{exit}} + \lambda_b \mathcal{L}_{\text{balance}} + \lambda_w \mathcal{L}_{\text{write}} + \lambda_s \mathcal{L}_{\text{stability}} + \lambda_r \mathcal{L}_{\text{route}}. \tag{30}$$

This should be the first empirical implementation because it directly optimizes the discrete computation that deployment will execute. Add $\mathcal{L}_{\text{exitKD}}$ when intermediate exits are substantially weaker than the terminal state. To learn a controllable Pareto frontier, sample several compute prices λ_T (or provide a target-budget embedding) during post-training and calibrate a small set of serving operating points on held-out data.

A soft expected-exit objective normally unrolls all $T_{\max}$ steps during training, so it does *not* by itself reduce training FLOPs. Once the halt policy is stable, train-time savings require masked/hard exits, depth sampling, or a straight-through/stochastic policy; these introduce optimization bias or variance and must be reported separately from inference savings.

7.5 Optional continuous-depth / Neural-Event training

A more principled but more complex view treats recurrence as a discretization of latent computational time. Let $z_i(\tau)$ collect hidden state and differentiable scratchpad state:

$$\frac{dz_i(\tau)}{d\tau} = F_\theta(z_i(\tau), \mathcal{M}_{\text{tok}}, \tau), \qquad z_i(0) = z_{i,0}. \tag{31}$$

The discrete residual update

$$z_{i,t+1} = z_{i,t} + \Delta\tau\, F_\theta(z_{i,t}, \mathcal{M}_{\text{tok}}, \tau_t) \tag{32}$$

is forward Euler. Neural ODEs therefore provide a useful interpretation of recurrent depth as sampled latent time [3].

Numerical adaptation is not semantic completion. An ODE solver adapts evaluation points to satisfy a numerical error tolerance at a requested terminal time. That can allocate more function evaluations to numerically difficult trajectories, but it does not answer whether the *token has reasoned enough*. Semantic completion requires a learned event.

Following Neural Event ODEs [4], introduce

$$e_{\phi,i}(\tau) = g_\phi(z_i(\tau), \mathcal{M}_{\text{tok}}, \tau) \tag{33}$$

and terminal time

$$\tau_i^\star = \inf\{\tau > 0 : e_{\phi,i}(\tau) \le 0\}, \qquad \tau_i^\star \le \tau_{\max}. \tag{34}$$

Depth becomes integration time; halting becomes event detection. If the event crossing is transversal, the terminal time is locally differentiable by the implicit-function theorem. Grazing crossings are poorly conditioned and motivate event-margin regularization.

A continuous compute-aware objective is

$$\mathcal{L}_{\text{cont}} = \mathcal{L}_{\text{LM}}(z(\tau^\star)) + \lambda_\tau \mathbb{E}[\tau^\star] + \lambda_{\text{nfe}} \mathbb{E}[\text{NFE}] + \lambda_C \mathbb{E}\left[\int_0^{\tau^\star} c(\tau)\, d\tau\right] + \lambda_e \mathcal{L}_{\text{event}}, \tag{35}$$

where $c(\tau)$ may price expert FLOPs, bytes moved, scratchpad traffic, or predicted latency.

Why this is optional. Hard top-k routing, hash lookups, sparse writes, and discrete halt decisions are nonsmooth hybrid events. The original continuous-adjoint Neural-ODE recipe is therefore not automatically appropriate; ANODE and later work caution that reverse integration/continuous-adjoint gradients can disagree with the actual discretized computation [5]. For the final sparse model, exact unrolled/checkpointed discrete gradients are the safe baseline.

Hybrid ODE-to-discrete curriculum. A practical use of the continuous formulation is as a teacher or regularizer:

1. train the fixed-depth recurrent backbone;
2. train discrete expected-exit halting;
3. optionally soften routing/writes and fit a continuous trajectory plus event time;
4. distill continuous states and terminal times into discrete recurrent states and halt probabilities;
5. anneal back to hard indexed routing and sparse writes;
6. finish with deployment-measured latency/energy cost.

8 Conversion and Post-Training From Existing Models

Starting from a strong teacher is likely the fastest way to test the architecture.

8.1 Dense Transformer to recurrent sparse model

Suppose a teacher has blocks $B_1, \ldots, B_L$. Choose a recurrent macro-block of r layers and $T_0 \approx L/r$ recurrences.

Core initialization. Use one or more of: representative teacher blocks; weight averaging after alignment; a small group of shared layers as in MoEUT; or the RRT recipe with phase-specific low-rank residuals [11, 12]. Full tying should be introduced progressively rather than as a one-step architecture shock.

Expert transplantation. Teacher FFNs are pretrained transformations. Copy FFNs from different depths into a global expert library, cluster them by activation/Jacobian signatures, or factor each into a shared trunk plus expert residual. For an MoE teacher, preserve native experts and flatten former layer-local pools into a global bank when dimensions permit. ReXMoE motivates progressively enlarging the candidate pool rather than exposing all cross-depth experts at once [21].

Trajectory distillation. Map recurrent steps to teacher depth anchors and train

$$\mathcal{L}_{\text{traj}} = \sum_t \beta_t \left[1 - \cos(P_t h_t, h_{\ell_t}^{\text{teach}})\right] + \gamma \, \text{KL}(p_{\text{teach}} \| p_{\text{rec}}). \tag{36}$$

RRT is an especially important baseline because it already demonstrates a strong pretrained-to-recursive conversion path [12]; more recent retrofit work also studies recurrence curricula for pretrained LMs [39]. Any claimed advantage must exceed these baselines rather than only a naive tied model.

Memory and halt warm-up. Keep scratchpad writes zero during initial conversion. Then add read-only memory, sparse value writes, and finally adaptive halting. Indexed retrieval should be introduced after the exact recurrent router is stable.

8.2 Pretrained MoE to recurrent MoE

Preserve experts directly when dimensions match. A gentle migration uses a hierarchical bridge: route first to a former layer/group, then to an expert; progressively relax the layer-group prior until routing becomes globally reusable. Compare against ReXMoE, SUT, and MoEUT to isolate whether token-local writable state (and the optional sequence-scoped extension) plus indexed global retrieval add value beyond existing expert reuse [10, 11, 21].

8.3 Post-training data

No explicit chain-of-thought corpus is required for latent recurrence, but the halt policy needs heterogeneous difficulty. Use a mixture of ordinary next-token data, code, math, retrieval, long-context tasks, and algorithmic curricula. After supervised/continued pretraining, preference or RL optimization can use

$$R = R_{\text{quality}} - \eta\,\text{latency} - \xi\,\text{bytes} - \zeta\,\text{energy}. \tag{37}$$

The cost terms should come from the target serving stack rather than nominal FLOPs alone.

9 Adaptive Halting and Speculative Decoding

Adaptive recurrence and speculative decoding optimize different serial loops:

$$\underbrace{x_i \to x_{i+1} \to x_{i+2}}_{\text{outer autoregressive loop}}, \qquad \underbrace{h_0 \to h_1 \to \cdots \to h_{T(x_i)}}_{\text{inner latent loop}}.$$

Speculative decoding advances the outer loop by drafting/verifying future tokens [45, 46]; Medusa and EAGLE pursue related acceleration [47, 48]. Adaptive recurrence shortens the inner loop.

Therefore the defensible claim is not that recurrence "removes speculative decoding." If the recurrent target becomes sufficiently cheap on most tokens, the marginal value of a separate draft/verify path can shrink. If outer-loop seriality dominates, speculation can remain highly valuable.

A useful benchmark reports both expected compute and wall-clock latency:

$$\bar{C}_{\text{spec}}(m) = \frac{C_D(m) + C_V(m)}{A(m)}, \qquad \bar{C}_{\text{rec}} = \mathbb{E}\left[\sum_{t=1}^{T} C_{\text{step}}^{(t)}\right]. \tag{38}$$

Even if $\bar{C}_{\text{rec}} < \bar{C}_{\text{spec}}$, speculative verification can be faster in wall-clock time because it parallelizes work. The decisive result is a hardware-dependent phase diagram, not one universal winner.

A natural hybrid uses a shallow recurrent pass as the draft/confidence mechanism and continues recurrent refinement only for uncertain positions or failed candidates. This reduces "draft" and "target" toward two compute levels of the same shared model.

10 Hardware and Systems Co-Design

The architecture changes the serving problem from a long static pipeline into a persistent state machine with sparse data-dependent expert access.

10.1 Resident core and expert prefetch

Repeated attention/router/shared projections are candidates for residency in SRAM/cache or a persistent accelerator partition. The expert bank can occupy a larger memory tier. Indexed routing returns candidate IDs early enough to prefetch likely expert weights before final re-ranking completes.

10.2 Adapting continuous depth-wise batching to recurrent MoE state

RRT already identifies *continuous depth-wise batching* as a systems opportunity enabled by shared recursive layers [12]. We adopt that idea rather than claiming a new batching primitive, and ask what changes when each work item additionally carries an indexed expert route and writable state. Related dynamic-rebatching systems for early-exit models reinforce that depth divergence is fundamentally a scheduling problem [43].

Halting creates depth divergence, but strict lockstep execution is not required. Weight sharing makes heterogeneous latent depth batch-friendly: states at recurrent steps $t = 1, 2, \ldots$ all invoke the same shared core. The recurrent-step index or phase embedding is simply carried as input metadata. A serving system can therefore use a *continuous latent-time batch* whose work item is

$$\xi_i = (s_i, n_i, t_i, h_i, M_{\mathrm{sp},i}, \mathcal{P}_i), \tag{39}$$

where s_i is the request/sequence ID, n_i the current output-token position, t_i the latent recurrent step, h_i the recurrent state, $M_{\mathrm{sp},i}$ the scratchpad, and $\mathcal{P}_i$ references the paged token-KV history. The batch need not contain a common t_i; all items use the same shared parameters, while step or phase information enters through a small embedding or adapter selector. If a design uses a few phase-specific adapters, the scheduler can bucket only by phase rather than by exact depth.

At scheduler round r, take up to B work items from the ready queue, execute joint attention and routing, group the resulting states by selected expert, execute expert sub-batches, perform write-back, and evaluate halting. Each completed item follows one of two transitions:

$$\text{continue}: \quad (s, n, t, h, M_{\mathrm{sp}}) \to (s, n, t+1, h', M'_{\mathrm{sp}}), \tag{40}$$

$$\text{halt}: \quad (s, n, t, h, M_{\mathrm{sp}}) \to (s, n+1, 0, h_{\mathrm{init}}, M'_{\mathrm{sp}}), \tag{41}$$

where the halt transition first emits one token and appends its committed KV exactly once. Both successor items can be returned to the global ready queue immediately. Thus a hard token that needs six latent steps does not force easier tokens to wait; freed slots are continuously filled by later tokens from the same or other requests. Autoregressive causality is preserved because token $n+1$ of a sequence becomes eligible only after token n halts and commits.

The utilization advantage is easy to see in an idealized batch of N tokens with latent depths $T_1, \ldots, T_N$. A barrier-synchronous implementation that keeps the original batch intact until the hardest token finishes allocates $NT_{\max}$ token-steps, whereas perfect active compaction performs only

$$C_{\mathrm{useful}} = \sum_{i=1}^{N} T_i, \qquad \eta_{\mathrm{barrier}} = \frac{\sum_i T_i}{NT_{\max}}. \tag{42}$$

The reciprocal $1/\eta_{\mathrm{barrier}}$ is an upper bound on the token-step utilization gain available from compaction before accounting for queueing, gathers/scatters, and expert communication. The result is especially favorable when the depth distribution is broad—precisely the regime in which adaptive halting is useful.

Expert-side batching. After routing, tokens are already sparse. For expert e, form

$$\mathcal{B}_e = \{i : e \in S_i\}, \tag{43}$$

and execute each E_e on the packed matrix of states in $\mathcal{B}_e$. This converts irregular token-level choices into ordinary expert-local GEMMs. On a distributed system, the scheduler can coalesce dispatches by device, overlap indexed-route lookup with expert prefetch, and use a bounded locality bias only when it does not change semantic top-k decisions materially. Very small expert groups can be delayed for a short microbatch window, fused with nearby requests, or handled by a dense/local fallback depending on latency targets.

Attention and memory layout. Token KVs should use paged/ragged attention so sequence length need not be uniform across the batch. The fixed-size scratchpad and recurrent state are compact tensors indexed by active work item, which makes gather/compact/scatter inexpensive relative to repeatedly moving the full token history. Later recurrent iterations can use the two-tier evidence-attention mechanism described in Section 5.1: scratchpad plus a recent/retrieved token subset, reducing both attention FLOPs and batch-to-batch memory traffic.

Practical scheduler. A high-throughput implementation can maintain separate ready queues for (i) attention/router work, (ii) expert sub-batches, and (iii) post-expert write/halt work, with double buffering between stages. The scheduler may additionally bucket by predicted remaining depth to reduce churn, but this is an optimization rather than a correctness requirement. A hard cap $T_{\max}$ and per-request fairness policy prevent pathological tokens from monopolizing recurrent capacity. Importantly, batching should be optimized against *measured* latency and bytes moved: when batches are tiny or the expert bank is modest, dense routing and simpler synchronous execution may outperform indexed search plus queue management.

This schedule is related to continuous depth-wise batching in RRT [12], but the work item here carries expert-route state and writable memory, and the same shared core can mix requests at different latent-step indices in one microbatch. The design turns variable recurrent depth from a synchronization liability into a queueing problem that modern continuous-batching and paged-attention runtimes can manage explicitly [41, 42].

10.3 Index as a first-class control structure

Binary codes, product-key centroids, or quantizer metadata can remain resident even when expert weights do not. A hardware router can implement query projection, code generation, bucket probes, candidate queues, exact small-matrix re-ranking, and prefetch. Hash tables and Hamming operations are particularly amenable to FPGA/ASIC pipelines; product-key and PQ variants offer different trade-offs.

10.4 Locality-aware routing

Distributed MoE execution pays dispatch/return cost. Add a small locality prior only after semantic routing is stable:

$$s'_{t,e} = s_{t,e} - \lambda_{\text{net}} \,\text{remote}(e) - \lambda_{\text{miss}} \,\text{cacheMiss}(e). \tag{44}$$

This should be constrained so hardware locality cannot silently overwhelm model quality. Recent MoE accelerator work emphasizes dynamic expert scheduling and token movement as core bottlenecks [49, 50].

10.5 Scratchpad as a memory tier

The scratchpad is intentionally small. Keeping it near compute makes repeated latent steps cheap and supports fused residual update, write gate, and halt-score evaluation.

10.6 No-free-lunch systems risks

Dynamic execution can lower occupancy, increase synchronization, and fragment expert traffic. Repeated attention can dominate at long context. Large indexes may lose their advantage at small batches where a dense router GEMM is efficient. Hardware evaluation must therefore report latency, throughput, bytes moved, utilization, energy, route fallback rate, and P50/P95 recurrent depth—not only FLOPs.

11 Experimental Program and Falsification Criteria

A credible empirical follow-up should separate *architecture quality* from *systems quality*.

Variant	Recurrence	Experts	Scratchpad	Halt	Indexed route	Phase adapters
Tied recurrent	✓	–	–	–	–	–
SUT-like	✓	✓	–	✓	–	optional
+ scratchpad	✓	✓	✓	✓	–	optional
+ indexed route	✓	✓	✓	✓	✓	optional
Full proposed	✓	✓	✓	✓	✓	$q \in \{1, 2, 4\}$

Table 2: Ablations needed to attribute gains to the new pieces rather than recurrence/MoE mechanisms already present in prior work.

11.1 Closest baselines

At minimum compare:

- standard dense Transformer and matched-active-parameter MoE;
- UT, SUT, and MoEUT [7, 10, 11];
- RRT and MoR [12, 14];
- recurrent-depth latent-reasoning and pretrained-retrofit models [13, 39];
- Attractor Models as an implicit/fixed-point iterative-refinement baseline [40];
- Mixture-of-Depths / CALM / LayerSkip for adaptive compute [27, 26, 28];
- PEER and dense/hierarchical MoE routers for large-expert retrieval [20];
- ReXMoE for cross-depth expert reuse [21].

A weak comparison against only a standard Transformer would not establish the contribution.

11.2 Core architecture ablations

Two ablations are mandatory beyond Table 2. First, compare embedding-only canonical token KVs, terminal-state-conditioned commit KVs, and (as an upper-cost reference) recomputed/refined token KVs. Second, compare no scratchpad, token-local scratchpad, and sequence-scoped scratchpad while reporting both quality and training throughput. Without these tests the paper cannot establish that the immutable-history/writable-state split is doing useful work rather than merely moving cost.

11.3 Model and systems metrics

Report language modeling loss/perplexity, downstream reasoning/code/math quality, algorithmic length generalization, and long-context retrieval. For efficiency report active FLOPs, parameters, training tokens/s, training activation memory, prefill cost, mean/P50/P95 recurrent depth, full-attention refresh count, indexed-route recall@candidate budget, retained evidence-attention mass, exact-route agreement, fallback rate, bytes/token, KV traffic, latency, throughput, and energy.

11.4 Memory causality

Measure scratchpad attention mass, write frequency, slot lifetime, and intervention sensitivity. Freeze, erase, permute, or swap scratchpad slots. If quality is unchanged, the memory is decorative and the architectural claim weakens.

11.5 Routing study

Freeze a trained exact router and sweep expert count E, candidate budget B, code length, probe radius, and batch size. Compare an exact indexable inner-product router against richer dense/MLP routers to measure any expressivity tax, then compare dense inner-product routing, hierarchical routing, LSH, product-key retrieval, and quantized ANN. Then unfreeze for router-aware post-training. The relevant endpoint is end-to-end token latency at matched model quality, not ANN benchmark throughput in isolation.

11.6 Attention study

Sweep context length and compare full-evidence recurrence with two-tier evidence retrieval. Track retained attention mass as well as token recall, and test whether the $2\epsilon V$ sanity bound is predictive of downstream degradation. If repeated full attention eliminates the compute advantage at realistic context lengths, or if sparse evidence retrieval requires near-full context to preserve quality, the long-context version of the thesis is falsified.

11.7 Key falsifiers

The proposal should be considered unsupported if any of the following hold after careful optimization:

1. SUT/MoEUT/RRT/MoR match quality and systems cost without writable memory/indexed routing;
2. hard tokens do not improve with additional latent steps;
3. scratchpad interventions have negligible causal effect;
4. approximate routing needs near-dense candidate budgets to preserve trajectories;
5. repeated evidence attention dominates runtime;
6. active-token compaction and expert scheduling fail to recover utilization;
7. canonical single-KV commits lose essential depth-specific information that the scratchpad/phase adapters cannot recover;
8. writable-state training destroys enough token-parallel pretraining throughput to erase downstream serving gains;
9. at matched quality, latency/energy/bytes do not improve on target hardware.

12 Potential Industry Implications

If validated, model deployment could shift from selecting a single fixed depth to provisioning a compute envelope: a resident recurrent core, an expert library, a working-memory budget, an evidence-retrieval budget, and a halt policy. The same checkpoint could expose multiple operating points: aggressive halting for interactive chat, deeper latent recurrence for code/math, smaller expert/evidence budgets on edge hardware, or larger budgets offline.

The more important systems implication is architectural co-design. Current accelerators are optimized around long ordered sequences of matrix multiplications. A recurrent sparse model instead rewards loop locality, resident weights, compact working state, indexed routing, dynamic queues, and data-dependent expert prefetch. Whether this is economically better is an empirical systems question, not a consequence of the neural architecture alone.

13 Limitations and Open Questions

The current manuscript is an architecture/theory proposal without new large-scale empirical evidence. That is its largest limitation.

Novelty is compositional. Recurrence plus halting exists in UT; recurrence plus MoE and dynamic halting exists in SUT; recurrent MoE exists in MoEUT; pretrained recurrent conversion exists in RRT; token-adaptive recursion exists in MoR; large expert retrieval exists in PEER. The paper's claim is therefore the complete execution contract and its consequences, not invention of those mechanisms. Empirical work must demonstrate that writable state, indexed routing, and the immutable-history contract add value beyond these strong baselines.

Expressivity versus sharing. Strict parameter tying may be too restrictive. Phase adapters or small shared groups may be necessary, reducing some parameter-reuse advantage. Attention specialization may be as important as FFN specialization.

Attention can erase the efficiency gain. Full attention at every recurrent step multiplies long-context cost. Two-tier evidence retrieval is a proposed remedy, not a proven one.

Working memory may be hard to train. Writable scratchpad state can be ignored, overwritten, or become an uncontrolled hidden channel. Token-local scratchpads increase activation state during parallel training; sequence-scoped memory introduces an additional recurrence across token positions and can severely reduce pretraining parallelism while accumulating stale hypotheses. Fixed keys, sparse writes, resets, chunking, and causal interventions are safeguards, not guarantees.

A single canonical KV may be too lossy. A standard deep Transformer stores layer-specific KVs; this proposal intentionally commits only one immutable evidence representation per token. The terminal-state-conditioned commit encoder is an attempt to make that representation rich, but it may not preserve all information that later recurrent phases need. This is a central architectural risk and requires the commit-KV ablations described above.

Huge expert counts require fine-grained experts. Indexed routing removes the need to score every expert, but it does not remove expert storage or weight bandwidth. Million-expert claims are sensible only for tiny/fine-grained experts; conventional FFN-sized experts are likely limited to much smaller pools.

Routing approximation compounds over time. High single-step retrieval recall may still lead to low full-trajectory agreement for hard tokens. Confidence fallback may reduce the speedup.

Halting is difficult to calibrate. A halt loss can optimize the wrong cost proxy, stop prematurely, or be brittle under distribution shift. Soft expected-exit training also executes all recurrent steps unless hard/masked policies are introduced, so inference-time adaptive compute should not be misreported as a pretraining-compute reduction. The threshold should be calibrated on quality and actual serving cost, and maximum depth remains necessary.

Continuous-depth training is not free. ODE solver adaptivity controls numerical error, not semantic reasoning. Hard sparse routing and writes are hybrid/discrete dynamics. Continuous training is an optional teacher/regularizer, not a claim that an off-the-shelf ODE solver directly trains the final model.

Systems novelty is also compositional. Continuous depth-wise batching is already proposed in RRT, continuous sequence batching and paged KV management are established serving techniques, and dynamic rebatching has been studied for early-exit models [12, 41, 42, 43]. The systems contribution here is the mapping of those ideas to recurrent MoE routing plus writable state, not the invention of continuous batching.

Speculation remains orthogonal. Early latent halting does not remove outer-loop autoregressive dependency. Speculative decoding may remain useful or may combine with the architecture.

14 Conclusion

A deep untied stack is one way to organize neural computation, not the only one. The literature already establishes many pieces of an alternative: learned wiring, recurrent depth, sparse experts, adaptive computation, read/write memory, expert retrieval, and program-like looped Transformers. The proposal here is intentionally narrower and more testable than "the ultimate Transformer."

It asks whether an autoregressive model can be organized around one explicit execution contract:

$$\boxed{\text{immutable evidence + writable state} \rightarrow \text{attend} \rightarrow \text{retrieve/route} \rightarrow \text{experts} \rightarrow \text{update} \rightarrow \text{halt or recur.}} \quad (45)$$

If this factorization works, stored capacity, latent computation time, and working state become more independent scaling knobs, and the hardware can be designed around a persistent recurrent engine rather than a long fixed pipeline. If it fails, the failure modes are equally informative: untied depth may carry indispensable specialization, repeated attention may be too expensive, memory may not become causal, or dynamic execution may defeat modern accelerators.

The next step is therefore not another architectural argument. It is a controlled experiment against SUT, MoEUT, RRT, MoR, recurrent-depth LMs, and PEER at matched quality and measured systems cost. The thesis earns its strength only if those measurements move the frontier.

Acknowledgments

This manuscript develops ideas that connect the author's earlier work on learned neural wiring with subsequent advances in recurrent, continuous-depth, sparse, memory-augmented, and retrieval-based Transformer models. The author acknowledges ChatGPT by OpenAI for assistance with manuscript organization, writing and editing, literature synthesis, and the design and refinement of explanatory figures, posters, and animation concepts. ChatGPT was used as a writing and design assistant; the author is responsible for the technical ideas, claims, citations, and final content of the manuscript.

References

[1] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering Neural Wirings. *NeurIPS*, 2019.

[2] Ashish Vaswani et al. Attention Is All You Need. *NeurIPS*, 2017.

[3] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *NeurIPS*, 2018. Best Paper.

[4] Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. Learning Neural Event Functions for Ordinary Differential Equations. *ICLR*, 2021.

[5] Amir Gholami, Kurt Keutzer, and George Biros. ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs. *IJCAI*, 2019.

[6] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep Equilibrium Models. *NeurIPS*, 2019.

[7] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal Transformers. *ICLR*, 2019.

[8] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. arXiv:1603.08983, 2016.

[9] Andrea Banino et al. PonderNet: Learning to Ponder. arXiv:2107.05407, 2021.

[10] Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse Universal Transformer. *EMNLP*, 2023.

[11] Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. MoEUT: Mixture-of-Experts Universal Transformers. *NeurIPS*, 2024.

[12] Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed Recursive Transformers: Effective Parameter Sharing with Layer-wise LoRA. *ICLR*, 2025.

[13] Jonas Geiping et al. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv:2502.05171, 2025.

[14] Sangmin Bae et al. Mixture-of-Recursions: Learning Dynamic Recursive Depths for Adaptive Token-Level Computation. arXiv:2507.10524, 2025.

[15] Noam Shazeer et al. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *ICLR*, 2017.

[16] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *JMLR*, 2022.

[17] Yanqi Zhou et al. Mixture-of-Experts with Expert Choice Routing. arXiv:2202.09368, 2022.

[18] Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. Hash Layers For Large Sparse Models. *NeurIPS*, 2021.

[19] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large Memory Layers with Product Keys. *NeurIPS*, 2019.

[20] Xu Owen He. Mixture of A Million Experts. arXiv:2407.04153, 2024.

[21] Zheyue Tan et al. ReXMoE: Reusing Experts with Minimal Overhead in Mixture-of-Experts. arXiv:2510.17483, 2025.

[22] Moses S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. *STOC*, 2002.

[23] Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search. *NeurIPS*, 2014.

[24] Ruiqi Guo et al. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. *ICML*, 2020.

[25] Jianyang Gao and Cheng Long. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. arXiv:2405.12497, 2024.

[26] Tal Schuster et al. Confident Adaptive Language Modeling. *NeurIPS*, 2022.

[27] David Raposo et al. Mixture-of-Depths: Dynamically Allocating Compute in Transformer-Based Language Models. arXiv:2404.02258, 2024.

[28] Mostafa Elhoushi et al. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding. *ACL*, 2024.

[29] Jaber Jaber and Osama Jaber. TIDE: Token-Informed Depth Execution for Per-Token Early Exit in LLM Inference. arXiv:2603.21365, 2026.

[30] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. arXiv:1410.5401, 2014.

[31] Alex Graves et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538:471–476, 2016.

[32] Zihang Dai et al. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *ACL*, 2019.

[33] Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent Memory Transformer. arXiv:2207.06881, 2022.

[34] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing Transformers. *ICLR*, 2022.

[35] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to Memorize at Test Time. arXiv:2501.00663, 2025.

[36] Angeliki Giannou et al. Looped Transformers as Programmable Computers. *ICML*, 2023.

[37] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the Turing Completeness of Modern Neural Network Architectures. arXiv:1901.03429, 2019.

[38] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. arXiv:2412.06769, 2024.

[39] Sean McLeish et al. Teaching Pretrained Language Models to Think Deeper with Retrofitted Recurrence. arXiv:2511.07384, 2025.

[40] Jacob Fein-Ashley and Paria Rashidinejad. Solve the Loop: Attractor Models for Language and Reasoning. arXiv:2605.12466, 2026.

[41] Gyeong-In Yu et al. Orca: A Distributed Serving System for Transformer-Based Generative Models. *OSDI*, 2022.

[42] Woosuk Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. *SOSP*, 2023.

[43] Xuting Liu, Daniel Alexander, Siva Kesava Reddy Kakarla, Behnaz Arzani, and Vincent Liu. Dynamic Rebatching for Efficient Early-Exit Inference with DREX. arXiv:2512.15705, 2025.

[44] Sangmin Yoo, Srikanth Malla, Chiho Choi, Wei D. Lu, and Joon Hee Choi. ADEPT: Adaptive Dynamic Early-Exit Process for Transformers. arXiv:2601.03700, 2026.

[45] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. *ICML*, 2023.

[46] Charlie Chen et al. Accelerating Large Language Model Decoding with Speculative Sampling. arXiv:2302.01318, 2023.

[47] Tianle Cai et al. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. *ICML*, 2024.

[48] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. *ICML*, 2024.

[49] Zifeng Zhao, Jiewen Zheng, Tianxing Xie, Xinghao Zhu, and Gengsheng Chen. MoEA: A Mixture of Experts Accelerator with Direct Token Access and Dynamic Expert Scheduling. *ASP-DAC*, 2026.

[50] Xuanda Lin et al. FLAME: Fully Leveraging MoE Sparsity for Transformer on FPGA. *DAC*, 2024.

A Detailed Conversion Recipe

A robust conversion should introduce one degree of freedom at a time.

Algorithm 2 Progressive conversion of a pretrained Transformer

1: Select macro-block size r, initial recurrent depth T_0, phase-adapter count q, expert count E, and scratchpad size m.
2: Initialize shared core from aligned teacher blocks or RRT-style recursive conversion.
3: Disable writes and halting; distill logits and depth-anchor trajectories.
4: Transplant/cluster FFNs or native MoE experts; train exact global/hierarchical router.
5: Train low-dimensional router queries and expert keys with dense exact scoring.
6: Fit retrieval index to preserve teacher top-k and margins; enable candidate retrieval + exact re-rank.
7: Add scratchpad as read-only KVs; then enable sparse value writes; optionally enable key writes.
8: Train adaptive halting with expected-exit objective; calibrate deterministic threshold.
9: Optionally fit soft ODE/event teacher and distill terminal times to discrete halting.
10: Hardware-calibrate route budget, evidence budget, expert placement, and halt threshold.

B Additional Analysis of Continuous Halting

Let an event surface be $g_\phi(z(\tau), \tau) = 0$. If the crossing is transversal,

$$\left| \frac{d}{d\tau} g_\phi(z(\tau), \tau) \Big|_{\tau^\star} \right| > \kappa > 0,$$

then the implicit-function theorem gives local differentiability of $\tau^\star$ with respect to model parameters. If the denominator approaches zero, the trajectory grazes the event surface and terminal-time gradients become ill-conditioned. Event-margin regularization and minimum crossing speed are therefore useful diagnostics. This observation is one reason the paper recommends continuous event training only as an optional intermediate phase.

C Training Failure Modes and Diagnostics

Always halt early Lower λ_T, use a minimum-depth curriculum, and ensure the exit loss in Eq. 19 actually trains shallow predictions.

Never halt Increase the measured compute price, randomize depth, and distill shallow teacher anchors.

Expert collapse Balance by recurrent phase, mine diverse teacher FFNs, increase transition diversity, or use expert-choice routing.

Index miss cascade Raise candidate budget, probe more buckets, train margin-preserving codes, and use confidence-triggered dense fallback.

Memory ignored Temporarily lower write sparsity and use tasks requiring state; verify causal effect with interventions.

Memory thrashing Fix keys, reduce write rate, add slot age/usage features, and reset sequence state when appropriate.

Recurrent drift Use gated residuals, pre-norm, gradient clipping, spectral/Jacobian penalties, and shallow-to-deep curricula.

Attention dominates Use two-tier evidence attention or periodic full-context refresh.

Hardware divergence Compact active tokens, group by expert path, and train against measured latency/bytes instead of step count alone.

D Suggested Ablations

- Scratchpad lifecycle: none, token-local reset, sequence-scoped persistence.
- Scratchpad keys: fixed learned keys, slow EMA keys, fully writable content keys.
- Evidence attention: full context every step, first-step full + retrieved evidence, periodic refresh.
- Phase adaptation: $q \in \{1, 2, 4, 8\}$ shared-core phases or low-rank loop adapters.
- Expert routing: dense, hierarchical, LSH, product key, PQ/ScaNN; sweep candidate recall and fallback.
- $T_{\max} \in \{2, 4, 8, 16\}$ with fixed, PonderNet/ACT-style, and expected-exit halting.
- Top-$k \in \{1, 2, 4, 8\}$ and expert granularity.
- Direct discrete training versus optional ODE/event teacher-distillation phase.
- Immutable-history contract versus a deliberately mutable-token-KV control.

E What the Theory Does Not Prove

The formal statements in this paper are intentionally limited. They do not establish that realistic LM dynamics are contractive, that gradient descent discovers useful programs, that approximate expert retrieval remains semantically harmless after many recurrent steps, that writable memory improves sample efficiency, or that the architecture beats untied depth in wall-clock cost. Those are precisely the empirical questions the proposed experiments are designed to answer.