Beyond the Parameter Monolith: Reconstructive Memories, Executable Skills, and Residual Assembly for Language Models

Anonymous authors
Paper under double-blind review

Abstract

Language-model systems can separate contextual computation, persistent storage, and exact execution instead of updating all capabilities through one shared parameter system. We investigate FEM-ASM, a finite-element-method-inspired organization in which independently constructed document states and deterministic executable skills contribute typed proposals to a shared language-model state. An explicit residual operator reconciles proposals attached to common interface nodes. We evaluate this organization through controlled experiments and negative results rather than claiming a physical finite-element formulation of language. An attention-free MultiMesh prototype learns causal language modeling but does not establish competitive general capability. A versioned store contains 52,809 reconstructive memory elements near a 1.7-billion-floating-value budget; reconstruction is incomplete, with approximately 75% token accuracy. Support-aware lexical indices make these elements addressable under provenance-controlled query construction. For executable arithmetic, positional result observations substantially improve neural rendering relative to a repeated global result vector, and output substitutions change the model's preferred answer. A bounded attachment demonstration further measures the effect of making selected evidence available, without establishing the utility of loading an entire multi-billion-value store. The results support a separation of storage, execution, and neural coordination, while identifying unresolved limitations in question-only retrieval, unrestricted answer generation, and end-to-end efficiency.

1 Introduction

A language model's contextual state, a document archive, and an arithmetic procedure differ in how they are produced, verified, updated, and replaced. They need not share a single learning schedule. Transformer language models provide an effective general-purpose architecture (Vaswani et al., 2017), and attention-free state-space and convolutional alternatives demonstrate that attention is not the only viable sequence operator (Gu & Dao, 2024; Poli et al., 2023). Our question concerns a different axis: how can a language-model system coordinate independently constructed components with different representations and life cycles?

We investigate a modular organization inspired by domain decomposition (Toselli & Widlund, 2005). A document element stores a locally optimized state; an executable element implements deterministic instructions; and a shared neural field incorporates their proposals through explicit interface mappings. We call this organization FEM-ASM. The name denotes an architectural inspiration, not a claim that language obeys a physical partial differential equation or that the model is a finite-element discretization of a known weak form.

The distinction between fixed observations and learned contextual computation also motivates our input representation. Frozen visual Unicode representations and fixed binary-code ablations have already been studied in Transformer language models (Bochkov, 2025).

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

Our contributions are threefold:

1. An explicit typed residual-assembly construction, with a precisely delimited consistency guarantee and an attention-free implementation.
2. A versioned reconstructive-memory pipeline that separates document-state optimization, publication, indexing, and subsequent use by a language-model core.
3. Controlled interface experiments that distinguish addressability, reconstruction, execution, and output rendering, including unsuccessful designs and their replacements.

We do not claim superiority over Transformers, raw-text retrieval, or existing tool-using systems. The central empirical question is which modular interfaces work, which fail, and what each successful experiment actually establishes.

2 Related work

Fixed and visual language inputs. PIXEL represents text through rendered images and learns a pixel-based language representation (Rust et al., 2023). Bochkov (2025) studies frozen visual Unicode inputs and includes fixed binary and other frozen-input ablations. These precedents establish that visual observations and frozen token coordinates are not new in themselves. We use fixed Binary16 identity inputs in the present architectures, without claiming that learned embeddings are universally unnecessary or harmful.

External memory. RAG, kNN language models, RETRO, and Memorizing Transformers already demonstrate the usefulness of external storage (Lewis et al., 2020; Khandelwal et al., 2020; Borgeaud et al., 2022; Wu et al., 2022). Our study combines locally optimized document states, immutable versions, explicit support mappings, and executable elements. We have not demonstrated that the reconstructive representation outperforms storage and retrieval of the original text. Raw-text retrieval is therefore an essential practical comparator.

Executable computation. Toolformer provides a precedent for learning to use external APIs (Schick et al., 2023). Our arithmetic experiments use a deliberately restricted controller, deterministic operand extraction, and a fixed program compiler. Their purpose is to study the interface through which an execution result enters a neural output field, not to introduce tool use or demonstrate general program synthesis.

Shared computation and aggregation. Universal Transformers and deep equilibrium models provide precedents for recurrent or shared computation (Dehghani et al., 2019; Bai et al., 2019). Permutation-invariant set aggregation is also well established (Zaheer et al., 2017). Our use of summation is not a novelty claim by itself. The studied combination is a typed interface with explicit availability, independently managed artifacts, and a measurable consistency substep. Compressed and multiresolution history similarly have precedents, including Compressive Transformers (Rae et al., 2020).

3 FEM-ASM: Typed elements and residual assembly

3.1 Implemented interface

Let $Z_i^k \in \mathbb{R}^d$ be the neural state at interface node i and iteration k. An external element specifies its type τ_e, source state u_e, target node $i(e)$, availability boundary a_e, nonnegative weight w_e, and provenance metadata p_e.

In the implemented pilot, a shared adapter for each type computes

$$z_e = T_{\tau_e}(u_e).$$

The adapter is shared across element instances: there is no learned adapter indexed by document identity. Provenance is retained for auditing and lifecycle management; it is not an additional input to this pilot adapter. Query-conditioned adaptation is a possible extension, not a property assumed in the reported implementation.

The current support operator selects a target node:

$$B_e Z = Z_{i(e)}.$$

It does not implement a general learned finite-element basis. The normalized external residual is

$$R_i^k = \frac{\sum_{e:i(e)=i} w_e (z_e - Z_i^k)}{\epsilon + \sum_{e:i(e)=i} w_e}.$$

A node with no external contribution receives an exactly zero external residual.

The mathematical sum is invariant to the ordering of elements. Floating-point reductions can exhibit small order-dependent errors, so implementation tests use a stated numerical tolerance rather than requiring universal bitwise equality.

3.2 Consistency and neural correction

For fixed proposals, define the external consistency energy

$$\mathcal{E}(Z) = \frac{1}{2} \sum_e w_e \|z_e - Z_{i(e)}\|_2^2.$$

At a constrained node, let

$$W_i = \sum_{e:i(e)=i} w_e, \qquad \bar{z}_i = \frac{\sum_{e:i(e)=i} w_e z_e}{W_i}.$$

The deterministic substep is

$$Z_{\mathrm{c},i}^{k+1} = Z_i^k + \alpha R_i^k, \qquad 0 < \alpha \leq 1.$$

Equivalently, its effective interpolation coefficient is

$$\eta_i = \frac{\alpha W_i}{\epsilon + W_i}.$$

Because $0 < \eta_i \leq 1$, this substep cannot increase the quadratic energy for fixed proposals.

A bounded learned correction follows:

$$Z^{k+1} = Z_{\mathrm{c}}^{k+1} + \beta \tanh P_\theta\big(Z_{\mathrm{c}}^{k+1}, R^k, S_\theta(Z^k)\big).$$

The local causal operator S_θ and correction operator P_θ share parameters across runtime iterations. The complete iteration need not decrease external consistency energy or language-model loss.

This distinction matters experimentally. For negligible ϵ and $\alpha = 0.25$, the excess quadratic energy above its minimum contracts by approximately $(1-\alpha)^2 = 0.5625$. Thus an approximately 44% reduction is expected from the interpolation rule itself. It checks implementation consistency; it is not independent evidence of reasoning or a learned convergence mechanism.

3.3 Conditional causality

The local sequence operators are causal. An external proposal is permitted only at positions compatible with its availability boundary. This is a conditional guarantee: it assumes that the proposal and the decision to attach it do not themselves depend on unavailable target information.

Offline document memory can contain information about an entire stored document. Reading that memory is appropriate for an external-memory task, but it is not equivalent to closed-book next-token prediction. In particular, a one-block delay alone does not certify that a jointly optimized latent cell or coarse parent depends only on earlier source tokens. Standalone LM evaluation, external-memory evaluation, and oracle-coordinate diagnostics are consequently reported separately.

4 System components

4.1 Fixed observations and the Multi-Mesh prototype

Binary16 assigns distinct fixed codes to token IDs when $|\mathcal{V}| \leq 2^{16}$, followed by a parameter-free lift to the model width. Zero trainable input-table parameters do not imply the absence of stored buffers: a materialized codebook, if present, is reported separately.

The earlier Multi-Mesh LM additionally uses deterministic visual-form observations, structured binary tiles, coarse completed-span fields, and a scratch field. A short block packs

$$64 \cdot 16 = 1024 = 32 \cdot 32$$

identity bits into a spatial tile. The tile is not a QR code and does not imply a frequency-domain representation.

This prototype and the later modular core are distinct architectures. The prototype uses several stage-specific parameter sets with weight sharing within each stage. The modular pilot instead reuses one local operator and one correction operator across iterations.

4.2 Reconstructive document memory

A shared attention-free codec maps a completed block to an initial cell. A worker optimizes a document-specific residual and stores the resulting final cells:

$$C_b = C_b^0 + M_b.$$

The frozen decoder reconstructs output slots without receiving the block's correct previous tokens as decoder inputs:

$$(C_b, p_i, \text{optional parent context}) \longmapsto p_\phi(x_{b,i}).$$

Parent context, when enabled, is derived from the document's cells. It must therefore be included in the effective readout dependency and resource accounting.

Published elements have immutable versions, codec identity, provenance, and checksums. A snapshot specifies the exact versions available to a run. These properties support reproducible attachment and replacement protocols, but do not themselves establish factual correctness or robustness to malicious elements.

The measured reconstruction is incomplete. We therefore use *reconstructive memory*, not lossless exact memory.

4.3 Executable elements

The frozen VM implements bounded signed-integer computation. In the successful arithmetic protocol, the neural controller selects among addition, subtraction, multiplication, division, and modulo. Operands are obtained deterministically and inserted into a fixed program skeleton.

Two return interfaces are compared:

- a global result vector repeated over output positions;
- position-specific render observations containing the intended output-token identities, optionally augmented by character observations.

The second interface exposes an executable element's rendered output. It tests neural adoption and serialization of an external result, not unaided arithmetic. The reported render-buffer generation also uses the buffer's declared token count to determine termination; it is not an independent learned stopping test.

4.4 Retrieval and QA diagnostics

The initial learned retriever mapped text-query states into mean-pooled document-memory keys. Its retrieval performance was poor. This does not establish that independently optimized cells have no usable information, nor that mean pooling must fail in every setting.

The repaired candidate generator uses fixed prefix, sliding-window, token-bag, and n-gram sketches. A shared reranker scores candidates before memory attachment. The evaluation tasks are prefix completion, masked-span restoration, value restoration, and quote continuation.

Several experiments construct retrieval cues from the original evidence token IDs and stored answer offsets. This preserves tokenization but grants access to gold evidence coordinates. It is a controlled addressability protocol, not a question-only retrieval system. A deployable question-only interface must be evaluated without these privileged fields.

5 Experimental protocol and resource accounting

5.1 Data and separation

The LM experiments use FineWeb-Edu (Penedo et al., 2024) and a SmolLM2-family tokenizer (Ben Allal et al., 2025). Using this tokenizer does not mean that the experimental models were initialized from SmolLM2 weights.

Memory workers use designated odd-numbered training shards. The base modular language stream uses even-numbered training shards. QA records are assigned to train, validation, and test by element identity before task generation.

This protocol prevents the same element identity from appearing in multiple QA splits. It does not by itself exclude duplicated text under different identities, establish benchmark decontamination, or imply that all initialization checkpoints were unexposed to the held-out documents. R1 and QA supervision also expose information from memory-source documents to the reasoning network; the system must not be described as having trained only on even-shard information.

Appendix A summarizes the reported architecture and training configurations.

The QA split is an element-identity holdout unless a stronger content-deduplication audit is explicitly reported.

5.2 Separate ledgers

We distinguish trainable parameters, frozen shared parameters, external state values, and derived caches:

$$P_{\text{train}}, \quad P_{\text{codec}}, \quad N_{\text{memory}}, \quad B_{\text{index}}, \quad B_{\text{readout}}.$$

A system with a small trainable core and billions of frozen values is not a parameter-matched dense model of the same nominal size.

The current latent representation is not storage compression relative to raw token IDs. With 768 BF16 values per 64-token cell, the local cell occupies 1,536 bytes, whereas 64 uint16 token IDs occupy 128 bytes. This is a factor of 12 before coarse states, indices, and readout caches are included.

5.3 Evaluation conventions

Standalone benchmarks disable external memory and VM elements. Memory and skill experiments report their own conditioning, supervision, and controls. Attachment demonstrations hold core weights fixed while changing evidence availability.

Accuracy is reported as a percentage. Token NLL is in natural logarithm units. Perplexity definitions, including word versus token normalization, are kept distinct. Benchmark config-

Table 1: Standalone Multi-Mesh results. Accuracy entries are percentages; WikiText metrics use their stated units. These are not scores of the modular attachment core.

Metric	Multi-Mesh LM
HellaSwag accuracy	29.00
HellaSwag normalized accuracy	30.77
ARC-Easy accuracy	53.37
ARC-Challenge accuracy	21.59
PIQA accuracy	62.95
WinoGrande accuracy	49.33
MMLU 0-shot accuracy	23.95
MMLU 5-shot accuracy	25.05
LAMBADA accuracy	4.87
WikiText word perplexity	65.02
WikiText bits per byte	1.13

Table 2: Frozen-codec readout-cache audit. This is a selected QA cache, not the entire memory store.

Quantity	Value
Evidence blocks	2,910
Elements represented	98
QA task references	10,000
Valid source tokens	186,031
Exactly reconstructed tokens	139,468
Token reconstruction accuracy	74.97%
Completely reconstructed blocks	1
Completely reconstructed blocks, fraction	0.0344%
Cache shards	12
Cache size, tensors and records	300,395,921 bytes
Measured build stage	13.78 seconds

uration and sample counts are required for every table. Single-run point estimates are not presented as multi-seed statistical conclusions.

6 Results

6.1 Standalone Multi-Mesh evaluation

Table 1 reports the standalone Multi-Mesh checkpoint without external memory or VM.

The tasks follow HellaSwag, ARC, PIQA, WinoGrande, MMLU, LAMBADA, and WikiText (Zellers et al., 2019; Clark et al., 2018; Bisk et al., 2020; Sakaguchi et al., 2020; Hendrycks et al., 2021; Paperno et al., 2016; Merity et al., 2017). These results establish that the prototype performs causal language modeling, not competitive general reasoning. The MMLU point estimates are around the nominal four-choice chance level and should not be described as strong reasoning performance.

6.2 Memory construction and positional decoding

The reported memory production run yielded 52,809 active element versions near a budget of 1.7 billion physically stored floating-point values. A separate readout-cache audit covered 2,910 unique evidence blocks from 98 elements and 10,000 QA task references.

The build-stage timing excludes some startup work and is not an end-to-end latency measurement. The cache audit passed its record-count and checksum checks. It demonstrates repeatable decoding of stored states, not lossless block storage or successful answer generation.

Table 3: Reported VM-rendering point estimates. The protocol uses the external buffer's declared output length. The evaluation configuration specifies 128 generation examples per range. Token and combined modes contain intended rendered token IDs.

Return interface	In-range exact (%)	Out-of-range exact (%)
Repeated global vector	8.59	6.25
Character-oriented positional	88.28	35.94
Tokenizer-aware positional	100.00	85.94
Token plus character positional	100.00	88.28

The distinction between token and sequence accuracy is substantial. For illustration only, independent token correctness of 0.75 would imply all-correct probability $0.75^8 \approx 0.10$ for an eight-token span. Errors need not be independent in the actual codec, so answer-span accuracy must be measured directly rather than inferred from token accuracy.

6.3 Controlled retrieval

On the reported full-S10 QA run, the fixed multi-chart generator achieved approximately 99.6% test candidate recall, and the reranker achieved approximately 97.7% test recall at one. The dataset contains 533,978 tasks from 5,296 elements.

These figures concern provenance-controlled, lexically derived tasks. In the coordinate-assisted protocol, retrieval cues use source token spans and stored offsets. Therefore the figures do not establish semantic retrieval from an arbitrary user question.

The corresponding answer-token losses were approximately 6.6930 with retrieved evidence, 6.7726 with wrong evidence, and 7.2544 without evidence. Wrong evidence accounts for much of the improvement over the no-evidence condition. The smaller additional gain from correct evidence is consistent with a document-specific contribution, but does not eliminate a generic in-domain conditioning effect.

Free QA generation in that run had zero measured greedy exact match. Successful retrieval and improved likelihood therefore did not establish a functioning free-form QA system.

6.4 Executable-result interfaces

In the structured arithmetic-routing experiment, the controller selected one of five operations and deterministic execution reached 100% accuracy on the reported test sets, including operands outside the training range. The out-of-range interval extends far beyond the training interval, but is not a formally disjoint range because it includes the training interval.

The positional-render comparison is summarized in Table 3. All variants are interpreted as external-result interfaces, not unaided arithmetic models.

The large difference between global and positional interfaces supports the usefulness of explicit output-slot correspondence in this experiment. The small difference between tokenizer-only and combined interfaces does not establish a statistically reliable benefit from the character channel.

There are protocol limitations. The supplied training code uses answer-position placeholders, whereas greedy evaluation feeds back generated tokens. Moreover, some evaluation code reconstructs only the current position's external attachment during greedy decoding rather than preserving the full historical attachment pattern. These train–inference differences can contribute to readout errors and must not be attributed exclusively to the representation.

The reported numeric exact metric also uses extraction of the first signed integer from decoded text. Strict whole-output equality, token-sequence equality, and termination accuracy should therefore be reported separately before interpreting this metric as unrestricted text exact match.

Table 4: Selected-evidence attachment demonstration. Source labels do not denote the state physically loaded by this evaluation. Accuracy is token-level, not answer exact match.

Source condition	Coverage	NLL	Token accuracy (%)
`G1p7`	0.00	6.5716	12.10
`G6p5`	1.00	2.8073	58.72

Substitution experiments show that changing the returned execution result changes answer likelihood. Because some runs mix substituted and unchanged results, intervention effects should be stratified by whether the returned value actually differs from the original value. Aggregate "follows returned" and "stays correct" rates can otherwise overlap.

6.5 Bounded attachment demonstration

A packaged demonstration evaluated 417 examples and 2,396 target tokens using unchanged reasoning weights.

The loss difference is 3.7643 nats per target token. The package supplies selected evidence rather than the complete originating store.

Because coverage changes from zero to one, this experiment measures the effect of making required evidence available. It does not isolate improved quality from increasing store size at fixed answerability, and it does not measure the latency or memory cost of serving the complete originating store.

7 Discussion

The experiments distinguish four problems that are often conflated: storing information, addressing the right stored element, extracting the appropriate content, and rendering an answer. A cell can reconstruct many source tokens while providing a poor pooled retrieval key. A retriever can find the right block while the answer interface fails. An executable skill can compute the exact value while a neural output layer renders it incorrectly.

Modularity makes these failures easier to isolate. A key construction can be changed without retraining document cells. An executable result can be replayed independently of the language head. An immutable state can be versioned without mutating the reasoning core. These are useful engineering properties, although immutable files and type-shared adapters alone do not prove superior reasoning, fault isolation, or distributed serving efficiency.

The FEM analogy is correspondingly limited. Our current assembly is a weighted least-squares reconciliation at explicitly selected nodes. It is not a physical weak formulation, and the observed consistency contraction is largely determined by its update rule. The substantive question is whether the organization provides useful behavior and favorable costs relative to simpler modular systems.

8 Limitations

Incomplete reconstruction and storage overhead. The current memory is neither lossless nor compact relative to raw token storage. The full cost includes frozen codec weights, coarse cells, search keys, readout caches, and metadata.

Privileged coordinates and initialization exposure. Evidence-derived query spans and answer offsets are unavailable in unrestricted QA. Element-ID splitting also does not guarantee content disjointness or absence of earlier exposure through initialization checkpoints.

Limited executable task. The successful controller chooses a small arithmetic skill set. Operand parsing and compilation are deterministic. Positional token buffers expose intended output identities, and buffer length controls termination in the reported protocol.

Incomplete baseline coverage. A matched raw-text retrieval baseline, together with storage, latency, and answer-quality measurements, is necessary. Permutation-invariant aggregation and tool use are established ideas; their presence alone does not demonstrate an advantage over RAG or other tool-augmented systems.

Statistical and systems limits. Reported interface accuracies are finite-sample point estimates. We do not infer seed robustness from a single run. Large-store coverage, end-to-end retrieval latency, active state, failure localization, and replacement behavior require dedicated measurements rather than extrapolation from a selected-evidence package.

9 Conclusion

FEM-ASM explores a language-model system with separately managed contextual computation, reconstructive document state, and deterministic execution. The experiments establish working instances of these interfaces and identify important failures: pooled latent retrieval, dense repeated readout, privileged coordinate dependence, and weak free QA generation. The evidence supports further investigation of modular coordination, not replacement of dense language models or equivalence between frozen stored values and trainable parameters.

Reproducibility statement

Appendix A summarizes the reported architecture and training configurations, and Appendix G identifies the released artifacts. The supplementary material includes documented runnable paths and selected research code. A startup test or short training run verifies only that path; it does not reproduce a long-training checkpoint. The package does not constitute a complete end-to-end reproduction of every experiment reported here.

AI use statement

Generative AI tools assisted with architectural ideation, experimental design, literature discovery, code drafting, diagnostic interpretation, and writing.

The authors reviewed the AI-assisted material, executed the reported experiments and checks, and take responsibility for the manuscript, numerical claims, and released artifacts.

The deterministic QA construction described here does not call an LLM to generate its questions or answers.

Ethics statement

The experiments use web-derived text and research language models that have not been established as safe for deployment. Source-data rights, privacy obligations, and third-party licenses remain applicable to distributed artifacts. Checksums establish file integrity, not factual reliability. The VM has no intended host file, network, or shell access; its resource bounds do not constitute a formal security audit. Future online memory collection would require explicit retention, consent, deletion, and poisoning policies. No autonomous collection of user interaction histories is claimed in the reported experiments.

References

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1909.01377.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, et al. SmolLM2: When smol goes big—data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025. URL https://arxiv.org/abs/2502.02737.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1911.11641.

Andrey Bochkov. Emergent semantics beyond token embeddings: Transformer LMs with frozen visual unicode representations. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=0dh8IynO1o.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George B. van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Edward Grefenstette, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

GNU Unifont contributors. GNU Unifont, version 14.0.01. Versioned font release, 2021. URL https://ftp.gnu.org/gnu/unifont/unifont-14.0.01/.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2312.00752.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2009.03300.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1911.00172.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2005.11401.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.07843.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.17557.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2302.10866.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1911.05507.

Phillip Rust, Jonas F. Lotz, Emanuele Bugliarello, Elizabeth Salesky, Miryam de Lhoneux, and Desmond Elliott. Language modelling with pixels. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=FkSp8VW8RjH.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1907.10641.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2302.04761.

Andrea Toselli and Olof Widlund. *Domain Decomposition Methods: Algorithms and Theory*. Springer, 2005. doi: 10.1007/b137868.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1706.03762.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. *arXiv preprint arXiv:2203.08913*, 2022. URL https://arxiv.org/abs/2203.08913.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017. URL https://arxiv.org/abs/1703.06114.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://aclanthology.org/P19-1472/.

A Architecture and training ledger

A.1 Reported prototype configurations

Table 5 summarizes the prototype configurations documented in the experimental code and run specifications.

A.2 Training details required for reproduction

The supplied large-run commands use BF16 autocast with FP32 parameters, AdamW, external next-token label shifting, and gradient clipping. The reported two-GPU Transformer configuration uses microbatch eight per GPU and eight accumulation steps:

$$2 \cdot 8 \cdot 8 \cdot 2048 = 262{,}144$$

processed tokens per optimizer update.

Table 5: Reported prototype configurations.

Component	Reported configuration
Multi-Mesh large-run specification	Width 2,048; five parameterized stages; two iterations per stage; three coarse levels; ratio four; tile span 64; fine/coarse/scratch kernels 7/5/3; FFN multiplier 4.25; tile width 64; tile-state dimension 2,048.
Modular interface pilot	Width 512; context 512; four shared iterations; local/preconditioner kernels 7/3; FFN multiplier two; memory input dimension 768; VM input dimension 128; consistency step scale 0.25.
Reconstruction codec-v2 production specification	Hidden and memory dimensions 768; six decoder operators; kernel seven; FFN multiplier 2.5 rounded to a multiple of 256; tile span 64; tile width 48; optional coarse-parent ratio four.

Its initial peak learning rate is $1.5 \cdot 10^{-4}$, minimum learning rate 10^{-5}, warmup 2,000 updates, weight decay 0.01, Adam coefficients 0.9 and 0.95, and clipping threshold 1.0 in the supplied command.

The Multi-Mesh command targets the same token batch using microbatch one and 64 accumulation steps per GPU. The initial correction scale is 0.18 in that specification. Any later learning-rate, clipping, correction-scale, or activation changes constitute an explicitly documented fork.

The v2 shared codec is trained by alternating local state adaptation and shared parameter updates. The supplied production specification uses 32 local adaptation steps per training episode and 128 for validation. Workers subsequently optimize only local states under a frozen codec. Reported worker production settings include 1,024 adaptation steps and publication thresholds of token recall at least 0.70 and reconstruction loss at most 2.60. Actual invocation logs, rather than defaults alone, identify the settings used by a published element.

A.3 Counting trainable, frozen, and active state

Trainable parameters, frozen buffers, and external artifacts are distinct resource categories. The relevant accounting categories are:

- trainable core, adapter, router, and reranker parameters;
- frozen shared codec parameters;
- stored memory values by dtype and bytes;
- derived coarse states and readout-cache bytes;
- index keys, metadata, and physical artifact bytes;
- selected elements and active values per query;
- peak allocated/reserved HBM and measured latency.

Frozen storage can avoid optimizer state and backward computation through that storage. It does not eliminate memory loading, retrieval, frozen-decoder computation, or adapter activation costs.

B Evaluation reporting considerations

A complete evaluation manifest would associate each numerical result with the following:

- an immutable checkpoint file hash and configuration hash;
- an evaluation-code commit and dependency versions;
- task names, dataset revisions, split names, and sample counts;

- prompt templates, few-shot seeds, and scoring conventions;
- BOS/EOS, truncation, context-window, and label-shift rules;
- memory snapshot and codec identities, if applicable;
- generation limits, stop conditions, and normalization rules;
- uncertainty estimates and the unit of resampling.

For QA, confidence intervals should resample elements/documents, not treat multiple tasks from one evidence block as independent. For paired interventions, the same tasks and output-scoring rules should be used in both conditions.

C Consistency calculation

At a constrained node,

$$\sum_{e:i(e)=i} w_e \|Z_i - z_e\|^2 = W_i \|Z_i - \bar{z}_i\|^2 + \sum_{e:i(e)=i} w_e \|z_e - \bar{z}_i\|^2.$$

The second term is a disagreement floor between fixed proposals. After the consistency step, the first term is multiplied by $(1 - \eta_i)^2$.

The guarantee is therefore:

- for fixed proposals and nonnegative weights;
- for the selecting-node support used here;
- for the deterministic consistency substep;
- not for the subsequent neural correction or LM loss.

A sample-global, data-dependent acceptance rule could couple future constraints to earlier positions. A strict causal implementation must avoid such coupling or separately establish that acceptance decisions are prefix-independent. A position-local analytic step avoids this particular issue.

D Visual resource and licensing

The visual-form resource is GNU Unifont 14.0.01 (GNU Unifont contributors, 2021). The reported rendering configuration uses Unicode NFC normalization, fixed whitespace/control-symbol handling, a fixed font and rasterization pipeline, area downsampling, and an explicit binary threshold. Token identity remains a separate Binary16 channel: visual similarity does not define token identity.

The supplied configuration records font size 64, a 64-pixel-high canvas with 64-pixel character cells, a threshold of 192, and a maximum of 32 normalized characters per token piece. Glyph observations are resized to the configured form-code resolution.

The font is a third-party resource, not an original contribution. The release retains the upstream copyright and license notices. GNU Unifont font releases from version 13.0.04 provide licensing under SIL OFL 1.1 and GPL version 2 or later with the GNU font embedding exception. The distribution separately documents the licensing of its own code, neural parameters, font files, and visual-codebook resources.

E Executable-element protocol

The implementation in `skill_vm.py` provides immediate loads, register transfers, bounded memory access, arithmetic, bitwise operations, shifts, comparison, conditional jumps, integer/character output, and halting.

The reference configuration provides 16 registers, 256 memory cells, signed 64-bit arithmetic, maximum program length 256, maximum execution length 1,024 steps, and a 4,096-character

output limit. Register writes use signed wrapping. Division truncates toward zero; division by zero and invalid accesses produce typed execution failures.

A complete replay record must bind both the program and the VM configuration, including arithmetic width and resource limits. A checksum of the result alone is not a proof that the intended program, configuration, or semantics were used.

In the successful arithmetic experiments, only the operation selection is learned. The fixed skeleton loads two supplied operands, applies the chosen operation, outputs the result, and halts. No claim of learned loop synthesis follows from the VM's ability to execute loops in standalone unit tests.

F Research trajectory and negative results

The initial Multi-Mesh prototype trained heterogeneous causal fields, but the fine convolutional branch dominated several transfer paths. A first memory objective optimized local next-token adaptations and did not reliably improve held-out positions relative to zero memory. The reconstruction codec instead trained cells to recover their associated blocks, making local memory useful while leaving incomplete reconstruction.

Mean-pooled document keys paired with separately learned queries did not yield useful retrieval in the reported runs. Fixed prefix sketches recovered literal cues, while internal quotes and missing spans required different support-aware charts. These experiments identify successful addressability protocols, not a universal impossibility result for learned retrieval.

The generic VM-program controller learned much of a fixed instruction scaffold but failed to recover exact variable operands. Deterministic operand extraction isolated operation routing. Localizing the routing decision near the operation cue removed dependence on later operand length in the structured protocol; it did not demonstrate arbitrary long-range understanding of operation descriptions.

Finally, positional output observations substantially improved rendering over a repeated global vector. The comparison also changed the information supplied to each output position and used an externally declared output length. These conditions are part of the result, not implementation details to omit.

G Artifacts

The research artifacts are organized by operating mode:

- standalone Multi-Mesh LM: `https://huggingface.co/E6E831728/fem-multi-mesh-1p7b`;
- selected-evidence modular attachment demonstration: `https://huggingface.co/E6E831728/modular-reasoning-0p5b-g6p5-demo`.

The supplementary material includes the Multi-Mesh preprocessing and training path and selected, documented source files for the memory, retrieval, and VM experiments. We distinguish runnable packaged paths from research components that require additional artifacts or datasets.