LEXICAL CONDITIONAL MEMORY: DECOUPLING SPECIALIZED MEMORY FROM REASONING FOR KNOWLEDGE-INTENSIVE TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLMs couple specialized lexical memory and general reasoning within a shared parametric backbone, creating two challenges for knowledge-intensive tasks. First, domain-specific terminology is often fragmented into multiple native subtokens, potentially requiring repeated composition across layers before higher-level reasoning. Second, the specialized vocabularies required by knowledge-intensive tasks are often large, long-tailed, and continuously evolving. Encoding and maintaining such lexical knowledge entirely within shared model parameters is therefore inefficient. To address these challenges, we introduce **Lexical Conditional Memory (LCM)**, a retrofit architecture that externalizes specialized lexical memory while preserving the native tokenizer and pretrained Transformer backbone. LCM comprises two complementary mechanisms. **Deterministic variable-length lexical routing** recognizes complete domain terms across variable-length subtoken spans and maps them directly to external memory entries, providing direct access to complete-term lexical representations. **Hierarchical conditional memory injection** integrates the retrieved representations through depth-specific interfaces, enabling specialized lexical knowledge to be accessed on demand rather than stored uniformly in the shared parameters. Together, these mechanisms provide an explicit external pathway for specialized lexical representations, reducing reliance on their implicit reconstruction and storage within the shared backbone. Across six benchmarks in medicine, law, and chemistry, LCM outperforms Engram by 2.1–2.2 macro-average points across two controlled Qwen2.5 scales and by 2.4 points in the native-Engram Argon setting. LCM also remains complementary to factual retrieval, with LCM+RAG achieving the highest overall macro-average performance. These results establish external lexical memory as an effective approach for adapting LLMs to knowledge-intensive tasks across architectures and domains.

1 INTRODUCTION

Large Language Models (LLMs) typically encode specialized lexical knowledge and support general reasoning within the same parametric backbone. This coupling creates two challenges for knowledge-intensive tasks, including those in medicine, law, and chemistry. **First, domain-specific terminology is often lexically fragmented.** General-purpose tokenizers can split specialized terms into multiple subtokens (Bostrom & Durrett, 2020), requiring the model to compose distributed token-level representations across Transformer layers before obtaining a coherent lexical representation. This repeated composition can consume computation across network depth (Cheng et al., 2026b). As illustrated in Figure 1, *hypercholesterolemia*—abnormally high blood cholesterol—may span several subtokens despite denoting a single medical concept. **Second, specialized vocabularies are large, long-tailed, and continually evolving.** Their scale makes it inefficient to encode all domain-specific lexical knowledge implicitly in shared parameters. Many specialized terms are also sparsely represented in general pretraining corpora, making robust parametric representation difficult. Moreover, new terminology continually emerges, while incorporating it through repeated large-scale parametric adaptation can be costly (Dong et al., 2024). The shared backbone must therefore support both expanding specialized lexical memory and general reasoning. These challenges motivate providing

Code is available at `https://anonymous.4open.science/r/LCM-7263`.

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.

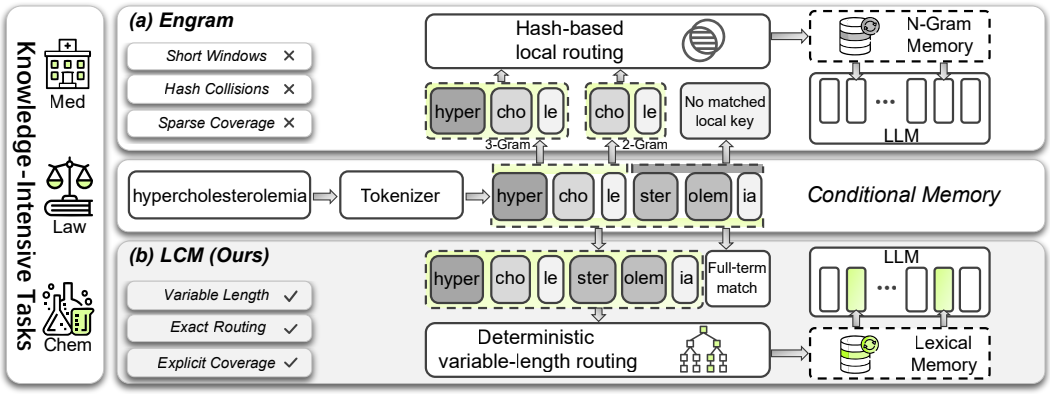


Figure 1: **Comparison of Engram and LCM for knowledge-intensive tasks with conditional memory.** (a) Engram accesses external memory through bounded local n-gram matching and hash-based routing, so specialized terms are accessed through short local keys and may be affected by hash collisions. (b) LCM recognizes complete variable-length terms and routes them deterministically to explicit lexical memory. The retrieved representations are injected through depth-specific interfaces at multiple Transformer depths, providing direct lexical support for fragmented domain terminology while preserving the pretrained backbone.

specialized lexical memory through an external pathway rather than relying exclusively on the shared parametric backbone.

Recently, conditional memory has emerged as a promising way to reduce this coupling. Engram (Cheng et al., 2026b), for example, externalizes recurring local patterns through bounded n-gram lookup and hash-based addressing. However, this access mechanism is not specifically designed for long, fragmented specialized terminology. As illustrated in Figure 1(a), a specialized term is accessed through short local n-gram keys rather than as a complete lexical unit, making it difficult to align memory access with the full lexical span of the term. Hash-based addressing can further introduce collisions by mapping distinct keys to the same memory location. Moreover, Engram constructs its memory offline from local patterns observed during pretraining (Cheng et al., 2026b;a). For sparse, long-tailed terminology, such corpus-derived memory is tied to patterns observed during pretraining rather than to an explicit, adaptable domain lexical inventory. Therefore, the central challenge is how to adapt Conditional Memory to knowledge-intensive domains with reliable full-term access, deterministic lexical addressing, and efficient incorporation of evolving domain vocabularies.

To address these challenges, we propose **Lexical Conditional Memory (LCM)**, a retrofit framework that externalizes specialized lexical representations while preserving the native tokenizer and pretrained backbone. As shown in Figure 1(b), LCM first performs **deterministic variable-length lexical routing**, identifying complete domain terms across variable-length subtoken spans and mapping them directly to an explicit external lexical memory. It then employs **hierarchical conditional memory injection** to integrate the retrieved lexical representations through independent interfaces at multiple Transformer depths. This design replaces bounded local lookup with full-term access, hash-based addressing with deterministic routing, and corpus-induced lexical availability with an explicit domain lexical inventory assembled from curated resources. Because LCM can be retrofitted onto an existing pretrained model through staged retrofitting, new domain terminology can be incorporated without pretraining the model from scratch. By providing direct term-level lexical support through an external memory pathway, LCM reduces reliance on implicit reconstruction and storage of specialized lexical representations within the shared backbone.

Experiments across six knowledge-intensive benchmarks in medicine, law, and chemistry consistently support this design. On Qwen2.5-0.5B and Qwen2.5-1.5B (Yang et al., 2025), LCM reaches macro-averages of 52.5 and 59.0, outperforming a controlled Engram retrofit (Cheng et al., 2026b) by $+2.2$ and $+2.1$ points and improving all 12 benchmark–scale comparisons. We further evaluate LCM on Argon-0.5B (Gong, 2026), a distinct pretrained backbone with native Engram integration. Replacing native Engram with LCM raises the macro-average from 44.1 to 46.5 and improves all six benchmarks, extending the observed advantage beyond the Qwen2.5 retrofit setting. LCM

also remains complementary to factual retrieval (Lewis et al., 2020): LCM+RAG further raises the Qwen2.5 macro-average to 53.2 and 59.8. Overall, our contributions are threefold:

- We identify two challenges in adapting pretrained LLMs to knowledge-intensive tasks: *fragmented lexical access*, where specialized terminology spans multiple native subtokens, and *scalable lexical memory*, which must accommodate large, long-tailed, and evolving domain vocabularies.
- We introduce **LCM**, which combines deterministic variable-length lexical routing, explicit external lexical memory, and hierarchical conditional memory injection. It provides direct full-term access at multiple Transformer depths while preserving the native tokenizer and pretrained backbone.
- We validate LCM across six benchmarks, two Qwen2.5 scales, and a distinct Argon backbone with native pretrained Engram. LCM consistently outperforms Engram in both controlled-retrofit and native-integration settings, while remaining complementary to retrieval-augmented generation.

2 Related Work

2.1 External Memory, Retrieval, and Conditional Memory

External memory expands accessible knowledge without storing all information in model parameters. Retrieval-augmented methods retrieve textual evidence from external corpora and condition generation or intermediate representations on the retrieved context (Lewis et al., 2020; Borgeaud et al., 2022). These methods primarily expose factual information as text, whereas LCM externalizes lexical representations for specialized terminology and injects them directly into hidden states, making lexical memory complementary to factual retrieval and motivating LCM+RAG.

Conditional memory instead operates directly in representation space. Engram (Cheng et al., 2026b) indexes large embedding tables through local n-gram keys and hash-based lookup. Memory Grafting (Cheng et al., 2026a) constructs conditional n-gram memories offline from a frozen grafting model and retrieves them through exact longest-match lookup. Lngram (Zheng et al., 2026b) replaces tokenizer-ID addressing with learned discrete latent symbols, while Lngram v2 (Zheng et al., 2026a) improves latent-memory scalability and interpretability through decoupled memory dimensions and context-aware readout. In contrast, LCM targets explicit domain-specific lexical units: it recognizes complete variable-length terms, routes them deterministically to a curated lexical inventory, and retrofits the resulting memory pathway into a pretrained backbone.

2.2 Tokenization and Lexical Adaptation in Specialized Domains

Subword tokenization enables open-vocabulary modeling by decomposing rare words into reusable units (Sennrich et al., 2016), but the resulting segmentation need not align with lexical or morphological structure. BPE can produce less morphology-aligned segmentations, and tokenizer design can affect downstream performance and efficiency (Bostrom & Durrett, 2020; Dagan et al., 2024). This mismatch is especially relevant in specialized domains, motivating models such as SciBERT and PubMedBERT to construct vocabularies from in-domain corpora (Beltagy et al., 2019; Gu et al., 2021).

Other work adapts tokenization after or alongside pretraining. Tokenizer specialization can improve context utilization and generation efficiency (Dagan et al., 2024), while knowledge-informed tokenization can introduce representations for clinical concepts without repeating full language-model pretraining (Hasan et al., 2025). LCM instead preserves the native tokenizer and pretrained backbone, recognizes complete domain terms after tokenization, and supplies their representations through an external lexical-memory pathway, separating term-level lexical adaptation from the fixed subword vocabulary.

3 Methodology

3.1 LCM Overview and Formulation

Given an input sequence $X = (x_1, \ldots, x_L)$, LCM augments a pretrained language model with an external lexical memory $\mathbf{E}_{\text{lex}}$, containing one representation per domain-specific lexical entry, while preserving the native tokenizer and pretrained backbone. At each lexical-hit endpoint t, LCM

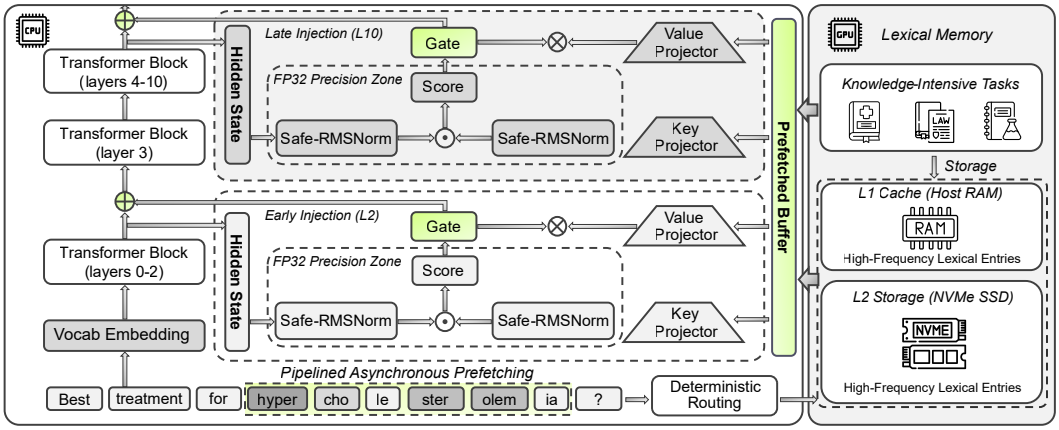


Figure 2: **Overview of Lexical Conditional Memory (LCM).** LCM deterministically routes complete variable-length domain terms to a hierarchically stored external lexical memory and prefetches the retrieved representations before model computation. The representations are conditionally injected through independent interfaces at early and late Transformer depths, providing direct full-term lexical access while preserving the native tokenizer and pretrained backbone.

retrieves the representation $\mathbf{e}_t \in \mathbb{R}^{d_m}$ of the matched complete term and injects its projected value into selected Transformer depths through a gated residual connection. For an injection depth ℓ_m,

$$\tilde{\mathbf{h}}_t^{(\ell_m)} = \mathbf{h}_t^{(\ell_m)} + \mathbb{I}_{\mathrm{hit},t}\, \alpha_t^{(m)} \mathbf{v}_t^{(m)}, \tag{1}$$

where $\mathbf{h}_t^{(\ell_m)} \in \mathbb{R}^d$ is the backbone hidden state, $\mathbf{v}_t^{(m)}$ is the projected lexical-memory value, and $\alpha_t^{(m)} \in [0,1]$ is a learned gate. Non-hit positions therefore follow the unchanged backbone computation.

As shown in Figure 2, LCM combines two mechanisms. **Deterministic variable-length lexical routing** recognizes complete domain terms over native subtokens and maps each match to its exact lexical entry. **Hierarchical conditional memory injection** supplies the retrieved representation through independent interfaces at multiple Transformer depths, providing depth-specific access to complete-term lexical information. The following sections detail these two components.

3.2 Deterministic Variable-Length Lexical Routing

LCM replaces bounded hash-based n-gram addressing with deterministic variable-length lexical routing. We compile the domain lexical inventory into an Aho–Corasick automaton (Aho & Corasick, 1975), which scans the native-token sequence and enumerates matched lexical spans. Each lexical entry has an exact memory index, avoiding hash collisions at the addressing level.

Let $X = (x_1, \ldots, x_L)$ be the tokenized input and $\mathcal{V}_{\mathrm{lex}}$ the lexical inventory in token space. At position t, the valid suffix matches are

$$\mathcal{M}_t = \left\{ (k, j) \;\middle|\; (x_{t-k+1}, \ldots, x_t) = \mathbf{x}^{(j)},\ \mathbf{x}^{(j)} \in \mathcal{V}_{\mathrm{lex}} \right\}, \tag{2}$$

where k is the span length and j the lexical-memory index. If multiple entries terminate at t, LCM selects the longest match,

$$(k_t^\star, j_t^\star) = \arg \max_{(k,j)\in\mathcal{M}_t} k, \tag{3}$$

and retrieves

$$\mathbf{e}_t = \begin{cases} \mathbf{E}_{\mathrm{lex}}[j_t^\star], & \mathcal{M}_t \neq \varnothing, \\ \mathbf{0}, & \text{otherwise}, \end{cases} \tag{4}$$

where $\mathbf{E}_{\mathrm{lex}}$ is the external lexical-memory table.

This routing preserves complete variable-length term identity rather than addressing terms through bounded local keys, and deterministically maps the same token sequence and inventory to the same

memory entry. We additionally include a small set of deterministic tokenization variants when constructing the lexical index; normalization and inventory construction are detailed in Sections A.1 and A.2.

3.3 Hierarchical Conditional Memory Injection

LCM injects retrieved lexical representations at two Transformer depths, ℓ_{early} and ℓ_{late}, through independent stage-specific interfaces. For each stage $m \in \{\text{early}, \text{late}\}$, the retrieved representation $\mathbf{e}_t$ is projected into key and value spaces:

$$\mathbf{k}_t^{(m)} = \mathbf{W}_K^{(m)} \mathbf{e}_t, \qquad \mathbf{v}_t^{(m)} = \mathbf{W}_V^{(m)} \mathbf{e}_t, \tag{5}$$

where the projections are independent across injection depths.

Let $\mathbf{h}_t^{(\ell_m)}$ denote the output of Transformer block ℓ_m. LCM computes a stage-specific compatibility gate

$$\alpha_t^{(m)} = \sigma\Big(\text{Clip}\Big(s\Big(\mathbf{h}_t^{(\ell_m)}, \mathbf{k}_t^{(m)}\Big) + b_{\text{gate}}^{(m)}\Big)\Big), \tag{6}$$

where $s(\cdot, \cdot)$ is the normalized compatibility function and $b_{\text{gate}}^{(m)}$ is a learned bias. The memory value is then injected through a gated residual update:

$$\tilde{\mathbf{h}}_t^{(\ell_m)} = \mathbf{h}_t^{(\ell_m)} + \mathbb{I}_{\text{hit},t}\, \alpha_t^{(m)} \mathbf{v}_t^{(m)}. \tag{7}$$

The update is applied after block ℓ_m and before the subsequent Transformer block. Independent early and late interfaces allow the same lexical memory to interact with depth-specific hidden representations while leaving the native Transformer blocks unchanged. Numerical stabilization of the compatibility and residual paths is detailed in Section A.3.

3.4 Decoupled Multi-Stage Retrofitting

Retrofitting an external memory pathway into a pretrained backbone introduces an optimization mismatch because the lexical representations, new transformations, and conditional gates are initially unaligned with the backbone representation space. LCM addresses this mismatch through a multi-stage curriculum that progressively aligns and integrates the memory pathway before downstream adaptation.

Following **domain lexical warm-up**, the retrofit proceeds through four stages. **Memory alignment** aligns projected lexical representations with the backbone space. **Forced memory utilization** adapts the backbone to use the lexical signal under forced injection. **Gate calibration** transitions from forced utilization to calibrated memory gating based on lexical-hit supervision. Finally, **joint task adaptation** jointly optimizes the backbone and trainable LCM modules on downstream supervision.

This curriculum separates representation alignment, memory utilization, and gate learning rather than optimizing all components simultaneously from initialization. Detailed objectives, freezing schedules, and hyperparameters are provided in Section A.4.

Efficient Retrieval and Storage. LCM uses **Pipelined Asynchronous Prefetching** to overlap off-device lexical retrieval with accelerator execution: background workers prepare the batch-local lexical working set while the GPU processes the current batch. The external lexical store uses frequency-aware tiering, keeping frequently accessed entries in host RAM and the remainder in SSD-backed memory-mapped storage. Only the batch-local working set is transferred to the accelerator, so the full lexical store remains independent of GPU-resident model parameters. Implementation and complexity details are provided in Sections A.5 and A.6.

4 Experiments

4.1 Experimental Setup

Tasks and Data Protocol. We evaluate LCM across three knowledge-intensive domains. For *Medicine*, we use MedQA (Jin et al., 2021) with its official test split and MedMCQA (Pal et al., 2022)

Table 1: **Main results across two Qwen2.5 backbone scales.** Results are mean accuracy (%) over three runs, with standard deviations shown as subscripts; Avg. is the macro-average across all six benchmarks. Lex/Fact denote lexical and retrieved factual information, while H+P/T+R denote hidden-state plus prompt injection and Trie plus retrieval routing, respectively. Best standalone results within each backbone scale are shown in **bold**; LCM+RAG is reported separately as a combined setting and is not included in the standalone ranking.

					Medicine		Law		Chemistry		
	Method	**Info**	**Inj.**	**Route**	**MedQA**	**MedMCQA**	**CaseHOLD**	**ContractNLI**	**ChemBench**	**MMLU-C**	**Avg.**
Qwen2.5-0.5B	SFT	–	Param	–	$34.3_{\pm.59}$	$39.2_{\pm.36}$	$72.8_{\pm.40}$	$79.2_{\pm.31}$	$36.0_{\pm.31}$	$35.6_{\pm.33}$	49.5
	DAPT+SFT	–	Param	–	$33.5_{\pm.31}$	$38.6_{\pm.65}$	$71.1_{\pm.64}$	$78.3_{\pm.43}$	$36.5_{\pm.71}$	$33.7_{\pm.33}$	48.6
	VocabExp	Lex	Emb	Vocab	$33.8_{\pm.63}$	$38.9_{\pm.36}$	$71.3_{\pm.17}$	$76.6_{\pm.24}$	$35.6_{\pm.31}$	$34.5_{\pm.50}$	48.5
	LPA	Lex	Prompt	Trie	$34.7_{\pm.63}$	$39.7_{\pm.24}$	$68.7_{\pm.40}$	$78.6_{\pm.62}$	$34.8_{\pm.59}$	$35.1_{\pm.83}$	48.6
	Engram	Lex	Hidden	Hash	$35.5_{\pm.79}$	$40.1_{\pm.38}$	$73.8_{\pm.68}$	$80.5_{\pm.48}$	$35.9_{\pm.51}$	$36.2_{\pm.50}$	50.3
	RAG	Fact	Prompt	Dense	$36.9_{\pm.39}$	$\mathbf{41.8}_{\pm.36}$	$73.2_{\pm.32}$	$80.7_{\pm.72}$	$37.7_{\pm.35}$	$38.3_{\pm.66}$	51.4
	LCM (Ours)	Lex	Hidden	Trie	$\mathbf{38.3}_{\pm.31}$	$41.6_{\pm.43}$	$\mathbf{75.1}_{\pm.51}$	$\mathbf{82.4}_{\pm.72}$	$\mathbf{38.5}_{\pm.47}$	$\mathbf{39.1}_{\pm.50}$	**52.5**
	LCM+RAG	Lex+Fact	H+P	T+R	$38.9_{\pm.47}$	$42.3_{\pm.19}$	$75.3_{\pm.38}$	$83.2_{\pm.43}$	$39.0_{\pm.75}$	$40.6_{\pm.66}$	53.2
Qwen2.5-1.5B	SFT	–	Param	–	$42.7_{\pm.24}$	$47.1_{\pm.41}$	$75.2_{\pm.56}$	$81.6_{\pm.29}$	$43.1_{\pm.51}$	$48.0_{\pm.19}$	56.3
	DAPT+SFT	–	Param	–	$41.4_{\pm.47}$	$45.6_{\pm.38}$	$75.8_{\pm.53}$	$80.0_{\pm.29}$	$42.9_{\pm.75}$	$47.2_{\pm.66}$	55.5
	VocabExp	Lex	Emb	Vocab	$41.5_{\pm.67}$	$45.8_{\pm.45}$	$73.6_{\pm.38}$	$81.8_{\pm.53}$	$41.2_{\pm.24}$	$44.3_{\pm.19}$	54.7
	LPA	Lex	Prompt	Trie	$42.8_{\pm.83}$	$47.4_{\pm.22}$	$72.7_{\pm.24}$	$81.4_{\pm.48}$	$42.0_{\pm.39}$	$47.3_{\pm.83}$	55.6
	Engram	Lex	Hidden	Hash	$42.1_{\pm.63}$	$48.5_{\pm.62}$	$76.4_{\pm.41}$	$82.0_{\pm.29}$	$43.5_{\pm.79}$	$48.8_{\pm.33}$	56.9
	RAG	Fact	Prompt	Dense	$44.0_{\pm.75}$	$49.0_{\pm.38}$	$76.7_{\pm.41}$	$82.9_{\pm.81}$	$46.3_{\pm.31}$	$\mathbf{50.9}_{\pm.68}$	58.3
	LCM (Ours)	Lex	Hidden	Trie	$\mathbf{45.4}_{\pm.67}$	$\mathbf{49.7}_{\pm.36}$	$\mathbf{79.0}_{\pm.56}$	$\mathbf{83.1}_{\pm.29}$	$\mathbf{47.4}_{\pm.43}$	$49.5_{\pm.33}$	**59.0**
	LCM+RAG	Lex+Fact	H+P	T+R	$46.6_{\pm.24}$	$50.0_{\pm.38}$	$79.8_{\pm.32}$	$83.9_{\pm.43}$	$47.3_{\pm.24}$	$51.4_{\pm.50}$	59.8

with the complete official development split, as labels for the official test set are unavailable. For *Law*, we evaluate on the official test splits of CaseHOLD (Zheng et al., 2021) and ContractNLI (Koreeda & Manning, 2021). For *Chemistry*, we evaluate on the official test sets of the multiple-choice subset of ChemBench (Mirza et al., 2024) and the chemistry subset of MMLU (Hendrycks et al., 2021) (MMLU-C); SciQ-Chem provides supervision, while ChemBench and MMLU-C are reserved for evaluation. We preserve official partitions and cap supervised adaptation at 20,000 examples per dataset. Dataset construction and preprocessing details are provided in Section B.1.

Baselines. We compare against controls for domain exposure, lexical integration, conditional memory, and factual retrieval. *SFT* is standard supervised fine-tuning without external memory. *DAPT+SFT* performs exposure-matched domain-adaptive continued language modeling before SFT, controlling for LCM's additional post-warmup domain exposure. *VocabExp* integrates the same lexical inventory through vocabulary expansion, while *LPA* exposes the same lexical identities through prompt-level augmentation without definitions or factual descriptions. *Engram* (Cheng et al., 2026b) provides the conditional-memory baseline; we adapt its bounded hash-based n-gram routing to the same Qwen2.5 retrofit setting and hidden-state injection interface. *RAG* retrieves factual passages at inference time, while *LCM+RAG* tests the complementarity of lexical memory and factual retrieval. VocabExp, LPA, Engram, and LCM share the same domain lexical inventory.

Backbones and Training. We evaluate Qwen2.5-0.5B and Qwen2.5-1.5B (Yang et al., 2025). For each domain and scale, SFT, LPA, Engram, and LCM start from the same Phase-0 domain-warmed checkpoint. LCM preserves the native tokenizer and keeps its external lexical memory frozen throughout retrofitting. Downstream supervision and optimization budgets are matched whenever applicable; full method-specific settings are given in Sections B.3 and B.5.

Evaluation Protocol. We report accuracy (%) over three independent runs, with standard deviations in the corresponding tables; *Avg.* is the unweighted macro-average across the six benchmarks. All methods use identical held-out examples and the same decoding and scoring procedure, and no evaluation example is used for optimization or model selection.

Table 2: Native-Engram backbone validation on Argon-0.5B. Results are mean accuracy with standard deviation over three runs.

Model	MedQA	MedMCQA	CaseHOLD	ContractNLI	ChemBench	MMLU-C	Avg.
Argon (Native Engram)	$27.0_{\pm.39}$	$33.6_{\pm.24}$	$76.9_{\pm.66}$	$77.5_{\pm.53}$	$30.2_{\pm.28}$	$19.3_{\pm.38}$	44.1
Argon w/o Engram	$26.2_{\pm.20}$	$33.1_{\pm.77}$	$76.5_{\pm.41}$	$78.1_{\pm.38}$	$25.7_{\pm.59}$	$18.4_{\pm.19}$	43.0
Argon w/ LCM	$\mathbf{29.8}_{\pm.31}$	$\mathbf{34.4}_{\pm.26}$	$\mathbf{78.9}_{\pm.32}$	$\mathbf{79.7}_{\pm.48}$	$\mathbf{31.3}_{\pm.55}$	$\mathbf{24.6}_{\pm.50}$	**46.5**

Table 3: **Component and training ablations on Qwen2.5-0.5B.** The first four rows vary the memory-injection architecture under the same decoupled training protocol, while the last two fix the full dual-depth architecture and compare retrofit strategies. Results are mean accuracy (%) over three runs. Medicine, Law, and Chemistry denote averages over the two benchmarks in each domain; Avg. is the macro-average across all six benchmarks.

Variant	Medicine	Law	Chemistry	Avg.
Early injection only	39.1	77.1	38.3	51.5
Late injection only	38.3	76.2	38.0	50.8
Dual-depth, shared	37.7	75.6	37.5	50.3
Dual-depth, independent	**40.0**	**78.8**	**38.8**	**52.5**
Naive E2E retrofit	36.2	75.6	34.5	48.8
Decoupled staged retrofit	**40.0**	**78.8**	**38.8**	**52.5**

4.2 Main Results

Overall Performance. As shown in Table 1, LCM achieves macro-averages of 52.5 and 59.0 on Qwen2.5-0.5B and Qwen2.5-1.5B, improving over SFT by $+3.0$ and $+2.7$ points and over Engram by $+2.2$ and $+2.1$ points, respectively. LCM+RAG further reaches 53.2 and 59.8, the highest macro-averages at both scales.

Comparison with Engram. LCM outperforms Engram in all 12 benchmark–scale comparisons. Unlike Engram's bounded hash-based n-gram addressing, LCM uses deterministic full-term routing to exact lexical entries, avoiding hash collisions at the addressing level and preserving complete variable-length lexical identity. The consistent gains across domains support the effectiveness of this more direct lexical-memory interface for fragmented specialized terminology.

Lexical and Parametric Controls. Exposure-matched DAPT+SFT reaches 48.6 and 55.5, remaining 3.9 and 3.5 points below LCM, showing that additional post-warmup domain exposure alone does not explain LCM's gains. VocabExp obtains 48.5 and 54.7, while LPA reaches 48.6 and 55.6, indicating that vocabulary expansion or prompt-level exposure to the same lexical identities likewise does not reproduce LCM's conditional-memory gains.

Complementarity with Factual Retrieval. LCM exceeds standalone RAG by $+1.1$ and $+0.7$ macro-average points, while LCM+RAG further improves over LCM by $+0.7$ and $+0.8$ points. The combined model improves over LCM on all six benchmarks at 0.5B and five of six at 1.5B, consistent with lexical memory and factual retrieval providing complementary support.

Cross-Architecture Native-Engram Validation. On Argon-0.5B (Gong, 2026), which integrates Engram during pretraining, removing native Engram reduces the macro-average from 44.1 to 43.0, while replacing it with LCM raises performance to 46.5, $+2.4$ points above native Engram and higher on all six benchmarks. Together with the controlled Qwen2.5 comparison, this shows that LCM remains effective across distinct backbone architectures and memory-integration settings; details are provided in Section B.4.

4.3 Component and Training Ablations

Injection Architecture. Table 3 compares lexical-memory integration across Transformer depths. Early-only and late-only injection achieve macro-averages of 51.5 and 50.8, while the full dual-depth architecture with independent interfaces reaches 52.5. Sharing parameters across depths reduces the average to 50.3. These results support multi-depth integration with depth-specific projection, gating, and normalization rather than a single injection point or shared interface. Per-benchmark results and protocols are provided in Section B.6.

Decoupled Retrofitting. With the LCM architecture fixed, naive E2E retrofit reaches 48.8 macro-average, whereas the staged retrofit reaches 52.5, a gain of $+3.7$ points. Gains are also consistent across domains: $+3.8$ in Medicine, $+3.2$ in Law, and $+4.3$ in Chemistry. This shows that separating representation alignment, forced memory utilization, and gate calibration is more effective than jointly optimizing all trainable backbone and memory-interface parameters from the outset.

Gate-Utilization Dynamics. Figure 3 examines Qwen2.5-0.5B on Medicine. Under the same token-dropout data, staged retrofitting maintains higher mean gate activation at lexical-hit positions during Phase III, while naive E2E remains in a lower-utilization regime. This is consistent with the curriculum first establishing memory use and then calibrating gate activation at lexical-hit positions.

4.4 Controlled Analysis

Lexical Content Specificity. We first test whether LCM benefits from term-specific lexical content rather than merely from lexical boundaries or stable term identities. Keeping routing, injection, and training fixed, shuffling the term–representation mapping reduces the macro-average from 52.5 to 49.1, while fixed random per-term and shared representations obtain 48.5 and 46.8, respectively. Thus, LCM's gains depend on preserving the correspondence between lexical terms and their stored representations, rather than on lexical-hit signals alone (Table 11).

Routing Factorial Analysis. We next disentangle the effects of span construction and addressing through a 2×2 factorial comparison between bounded 2–4-gram versus variable-length routing and hashed versus exact addressing. Exact addressing improves bounded routing from 50.3 to 51.7 macro-average accuracy, while replacing bounded spans with variable-length full-term access under exact addressing further improves performance to 52.5; the exact variable-length configuration achieves the highest mean accuracy on all six benchmarks. Routing diagnostics clarify this distinction: even without hash collisions, bounded 2–4-gram keys represent 345 underlying lexical entries per hit on average, whereas exact variable-length routing resolves each hit to a single lexical entry. These results indicate that bounded short-span ambiguity and hash-induced aliasing are distinct sources of routing noise, both addressed by LCM's full-term deterministic routing (Tables 12 and 13).

Fragmentation-Conditioned Performance. Finally, we descriptively examine how LCM's gains vary across pooled lexical-fragmentation bins. Among lexical-hit samples, we group examples by the maximum native-subtoken length $L_{\max}$ of any matched term. The pooled LCM–Engram difference increases from -0.6 points for $L_{\max} = 2$ to $+1.6$ for $L_{\max} = 3$–4 and $+3.2$ for $L_{\max} \geq 5$; relative to SFT, the corresponding gains increase monotonically from $+1.6$ to $+2.7$ and $+3.6$ points. LCM also yields a larger improvement over SFT on lexical-hit than no-hit samples ($+2.8$ vs. $+0.8$ points). These pooled trends are consistent with larger benefits from complete lexical access at higher fragmentation levels, while potentially reflecting benchmark-composition differences (Table 14).

4.5 System Efficiency and Memory Scalability

Inference Efficiency. All latency, throughput, and peak-memory measurements are conducted on a single NVIDIA A100 80GB GPU. As shown in Table 4(a), synchronous retrieval incurs 1.29–$1.46\times$ the latency of SFT, whereas pipelined asynchronous prefetching reduces end-to-end latency to 1.08–$1.12\times$ SFT. The pipelined implementation reduces latency over synchronous retrieval by 13–26%, while increasing peak allocated GPU memory over SFT by only 0.04–0.17 GiB across the three domains. Detailed measurement protocols, throughput, and memory statistics are provided in Section B.8.1.

Table 4: **System efficiency and lexical-memory scalability.** (a) End-to-end latency across domains. (b) Controlled legal lexical-store scaling under a fixed lexical-hit pattern. Store is the off-device FP32 vector payload; Peak GPU is peak allocated GPU memory. Results are reported as mean ± std over three runs.

(a) End-to-End Runtime (ms)

Metric	Med.	Law	Chem.
SFT	$161.3_{\pm 5.8}$	$180.3_{\pm 6.1}$	$151.3_{\pm 3.8}$
LCM-Sync	$213.9_{\pm 1.6}$	$263.3_{\pm 1.7}$	$194.9_{\pm 2.2}$
LCM-Pipe	$177.0_{\pm 4.7}$	$194.8_{\pm 6.1}$	$169.6_{\pm 7.0}$
Pipe/SFT	$1.098_{\pm 0.029}\times$	$1.080_{\pm 0.032}\times$	$1.121_{\pm 0.048}\times$

(b) Controlled Lexical-Store Scaling

Scale	Entries	Store (GiB)	Peak GPU (GiB)	Lat. (ms)
25%	31,853	0.114	1.916	$206.3_{\pm 3.8}$
50%	63,705	0.228	1.916	$208.0_{\pm 4.7}$
75%	95,557	0.342	1.916	$207.8_{\pm 2.6}$
100%	127,409	0.457	1.916	$205.3_{\pm 2.4}$

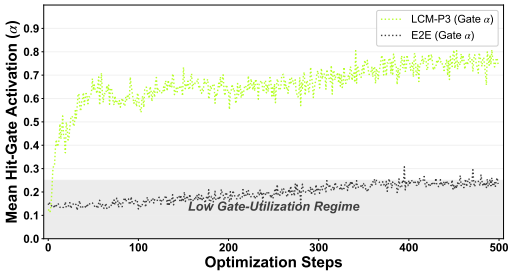


Figure 3: **Gate-utilization dynamics.** Mean hit-gate activation during LCM Phase III and end-to-end retrofitting.

Table 5: **Lexical fragmentation on the full evaluation sets.** Statistics use unique lexical entries matched by the production router; lengths are measured in native subtokens.

Benchmark	Entries	Mean	P95	Max
MedQA	4,541	3.29	6	11
MedMCQA	4,931	3.23	6	14
CaseHOLD	9,663	2.90	7	17
ContractNLI	1,453	2.36	4	13
ChemBench	737	3.31	7	15
MMLU-C	243	3.14	5	10

Memory Scalability. Table 4(b) isolates datastore size by holding the evaluation workload and lexical-hit pattern fixed while scaling the lexical store from 31.9K to 127.4K entries (0.114 to 0.457 GiB). Despite this $4\times$ increase in off-device vector storage, peak allocated GPU memory remains at 1.916 GiB and latency shows no systematic growth. This behavior follows from keeping the complete lexical store in host RAM/SSD storage and transferring only a dense batch-local lexical tensor of $O(BLd)$, together with $O(BL)$ indices, to the GPU. Additional scaling results and implementation details are reported in Section B.8.2.

4.6 Lexical Fragmentation Analysis

We characterize lexical fragmentation using the unique lexical entries matched by the production router on the complete evaluation sets, with lengths measured under the unchanged native tokenizer. Table 5 reports the number of matched entries and their mean, 95th-percentile, and maximum subtoken lengths. Across the six benchmarks, matched entries average 2.36–3.31 subtokens, with 95th-percentile lengths of 4–7 and maxima of 10–17. These statistics show that specialized terminology is frequently fragmented into multiple native subtokens and exhibits a substantial long tail. Additional distributional statistics are provided in Section B.9.

5 Conclusion

We introduced **Lexical Conditional Memory (LCM)**, a retrofit architecture that externalizes specialized lexical memory while preserving the native tokenizer and pretrained Transformer backbone. LCM combines deterministic variable-length lexical routing with hierarchical conditional memory injection to provide direct access to complete-term lexical representations through an external memory pathway. Across six benchmarks in medicine, law, and chemistry, LCM outperforms controlled Engram retrofits by 2.1–2.2 macro-average points across two Qwen2.5 model scales and improves native-Engram Argon from 44.1 to 46.5. LCM also remains complementary to factual retrieval through LCM+RAG. Together, these results support external lexical memory as an effective approach for externalizing specialized lexical representations while preserving a shared pretrained backbone in knowledge-intensive tasks.

AI USE STATEMENT

In this work, we used generative AI tools to assist with research ideation and execution, literature retrieval and discovery, interpretation of experimental results, manuscript drafting and language refinement, and parts of the software implementation. All AI-assisted outputs were critically reviewed and revised by the authors. AI-assisted code and experimental results were checked against actual implementations and runs, and literature identified with AI assistance was verified against the original sources before citation. The authors made all final methodological and interpretive decisions and take full responsibility for the accuracy, originality, and scientific content of the manuscript.

ETHICS STATEMENT

This work uses publicly available benchmarks and domain corpora and does not involve human subjects or private personal data. The experiments are intended for research evaluation only; model outputs in medicine, law, and chemistry should not be interpreted as professional advice or used for high-stakes decision-making without appropriate expert validation. We follow the licenses and usage conditions of the datasets and external resources used in this work.

REPRODUCIBILITY STATEMENT

We provide the LCM architecture and training procedure in Sections A.4 and 3, with baseline construction, optimization hyperparameters, training budgets, and parameter-freezing rules documented in Sections B.3 and B.5. Data preprocessing, lexical-resource construction, and benchmark-specific evaluation protocols are described in the experimental setup and corresponding appendix sections. All reported results use fixed evaluation procedures and are averaged over three independent runs.

REFERENCES

Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975. ISSN 0001-0782. doi: 10.1145/360825.360855. URL https://doi.org/10.1145/360825.360855.

Iz Beltagy, Kyle Lo, and Arman Cohan. SciBERT: A pretrained language model for scientific text. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3615–3620, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1371. URL https://aclanthology.org/D19-1371/.

Henry Campbell Black. *A Dictionary of Law*. West Publishing Company, 1891.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/borgeaud22a.html.

Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4617–4624, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.414. URL https://aclanthology.org/2020.findings-emnlp.414/.

Kathi Canese and Sarah Weis. Pubmed: The bibliographic database. In *The NCBI Handbook*. National Center for Biotechnology Information (US), 2013.

Runxi Cheng, Yuchen Guan, Yongxian Wei, Qianpu Sun, Qixiu Li, Sinan Du, Feng Xiong, Chun Yuan, Yan Lu, and Yeyun Gong. Memory grafting: Scaling language model pre-training via offline conditional memory, 2026a. URL https://arxiv.org/abs/2605.20948.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models, 2026b. URL https://arxiv.org/abs/2601.07372.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Roziere. Getting the most out of your tokenizer for pre-training and domain adaptation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 9784–9805. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/dagan24a.html.

Fang Dong, Mengyi Chen, Jixian Zhou, Yubin Shi, Yixuan Chen, Mingzhi Dong, Yujiang Wang, Dongsheng Li, Xiaochen Yang, Rui Zhu, Robert Dick, Qin Lv, Fan Yang, Tun Lu, Ning Gu, and Li Shang. Once read is enough: Domain-specific pretraining-free language models with cluster-guided sparse experts for long-tail domain knowledge. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 88956–88980. Curran Associates, Inc., 2024. doi: 10.52202/079017-2822. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/a1f12d8d3cc1b4789ff4ebec46e27609-Paper-Conference.pdf.

Pietro Ferrazzi, Aitor Soroa, and Rodrigo Agerri. Grounded multilingual medical reasoning for question answering with large language models, 2025. URL https://arxiv.org/abs/2512.05658.

Yuxuan Gong. Argon-0.5b: A 520m parameter deepseek-style pretraining and engram reproduction experiment, 2026. Custom PyTorch pretraining stack with DeepSeek-style tokenizer path and Engram memory components.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. 3(1), October 2021. doi: 10.1145/3458754.

Abul Hasan, Jinge Wu, Quang Ngoc Nguyen, Salomé Andres, Imane Guellil, Huayu Zhang, Arlene Casey, Beatrice Alex, Bruce Guthrie, and Honghan Wu. Infusing clinical knowledge into language models by subword optimisation and embedding initialisation. *Computers in Biology and Medicine*, 196:110747, 2025. ISSN 0010-4825. doi: https://doi.org/10.1016/j.compbiomed.2025.110747. URL https://www.sciencedirect.com/science/article/pii/S0010482525010984.

Peter Henderson, Mark Krass, Lucia Zheng, Neel Guha, Christopher D Manning, Dan Jurafsky, and Daniel Ho. Pile of law: Learning responsible data filtering from the law and a 256gb open-source legal dataset. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 29217–29234, 2022. doi: 10.52202/068431-2119.

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

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14), 2021. ISSN 2076-3417. doi: 10.3390/app11146421. URL https://www.mdpi.com/2076-3417/11/14/6421.

Sunghwan Kim, Jie Chen, Tiejun Cheng, Asta Gindulyte, Jia He, Siqian He, Qingliang Li, Benjamin A Shoemaker, Paul A Thiessen, Bo Yu, Leonid Zaslavsky, Jian Zhang, and Evan E Bolton. Pubchem 2025 update. *Nucleic Acids Research*, 53(D1):D1516–D1525, 2025. doi: 10.1093/nar/gkae1059.

Yuta Koreeda and Christopher Manning. ContractNLI: A dataset for document-level natural language inference for contracts. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 1907–1919, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.164. URL https://aclanthology.org/2021.findings-emnlp.164/.

Legal Information Institute. Wex legal dictionary and encyclopedia. https://www.law.cornell.edu/wex.

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 H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Carolyn E Lipscomb. Medical Subject Headings (MeSH). *Bulletin of the Medical Library Association*, 88(3):265–266, 2000.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. S2ORC: The semantic scholar open research corpus. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4969–4983. Association for Computational Linguistics, July 2020. doi: 10.18653/v1/2020.acl-main.447.

Adrian Mirza, Nawaf Alampara, Sreekanth Kunchapu, Martiño Ríos-García, Benedict Emoekabu, Aswanth Krishnan, Tanya Gupta, Mara Schilling-Wilhelmi, Macjonathan Okereke, Anagha Aneesh, Amir Mohammad Elahi, Mehrdad Asgari, Juliane Eberhardt, Hani M. Elbeheiry, María Victoria Gil, Maximilian Greiner, Caroline T. Holick, Christina Glaubitz, Tim Hoffmann, Abdelrahman Ibrahim, Lea C. Klepsch, Yannik Köster, Fabian Alexander Kreth, Jakob Meyer, Santiago Miret, Jan Matthias Peschel, Michael Ringleb, Nicole Roesner, Johanna Schreiber, Ulrich S. Schubert, Leanne M. Stafast, Dinga Wonanke, Michael Pieler, Philippe Schwaller, and Kevin Maik Jablonka. Are large language models superhuman chemists? 2024. URL https://arxiv.org/abs/2404.01475.

National Library of Medicine. Release of 2026 MeSH XML, RDF Datasets. https://www.nlm.nih.gov/mesh/, 2026.

Office of the Law Revision Counsel. United States Code. https://uscode.house.gov/.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In Gerardo Flores, George H Chen, Tom Pollard, Joyce C Ho, and Tristan Naumann (eds.), *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pp. 248–260. PMLR, 07–08 Apr 2022. URL https://proceedings.mlr.press/v174/pal22a.html.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 1715–1725, 2016.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Lucia Zheng, Neel Guha, Brandon R. Anderson, Peter Henderson, and Daniel E. Ho. When does pretraining help? assessing self-supervised learning for law and the casehold dataset of 53,000+ legal holdings. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Law*, ICAIL '21, pp. 159–168, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385268. doi: 10.1145/3462757.3466088. URL https://doi.org/10.1145/3462757.3466088.

Yunao Zheng, Bin Wen, and Xiaojie Wang. Lngram v2: Latent n-gram memory with interpretable discrete representations, 2026a. URL https://arxiv.org/abs/2609.03426.

Yunao Zheng, Guoyang Xia, Xiaojie Wang, and Lei Ren. Lngram: N-gram conditional memory in latent space, 2026b. URL https://arxiv.org/abs/2605.24869.

A Additional Method Details

A.1 Lexical Datastore Construction

We construct the lexical datastore from curated domain-specific terminology resources, with the aim of maintaining a high-precision inventory and limiting spurious memory activation.

Medicine. We collect medical terminology from the 2026 release of Medical Subject Headings (MeSH) (National Library of Medicine, 2026; Lipscomb, 2000), including both Descriptors and Supplemental Concept Records.

Law. We build the legal lexicon from the *First Edition* of Black's Law Dictionary (Black, 1891), the Wex Legal Dictionary maintained by the Cornell Legal Information Institute (Legal Information Institute), and the United States Code (Office of the Law Revision Counsel). We additionally include specialized legal glossaries covering Latin expressions, common legal terminology, statutory citation forms, and judicial abbreviations.

Chemistry. We obtain chemical entities from PubChem (Kim et al., 2025), using the top 200,000 compounds by Compound Identifier (CID) ranking and collecting their standardized IUPAC names and associated synonyms. We further supplement the inventory with fundamental terminology from chemistry glossaries.

For each canonical entry, we generate deterministic surface variants that account for tokenizer-sensitive formatting and domain-specific morphology, including capitalization, leading-space forms, inflectional variants, and selected orthographic transformations such as hyphen, symbol, or citation normalization. All variants are then encoded using the unmodified backbone tokenizer without inserting special tokens. We retain only entries fragmented into at least two native tokens and apply maximum tokenized lengths of 40 for medicine and law and 64 for chemistry to remove anomalously long candidates introduced by noisy extraction. These bounds are datastore-construction hyperparameters and do not impose a fixed N-gram window on LCM's routing mechanism. Detailed normalization rules are provided in Section A.2.

We next estimate the empirical usage frequency of the augmented whitelist by scanning large unlabeled domain corpora. Specifically, we use 5 million PubMed abstracts for Medicine (Canese & Weis, 2013), 1 million documents sampled from diverse Pile of Law subsets for Law (Henderson et al., 2022), and 5 million chemistry and materials-science abstracts from S2ORC for Chemistry (Lo et al., 2020). Corpus scanning performs exact matching only against the preconstructed tokenized whitelist and therefore does not introduce new lexical entries. The resulting frequency statistics are used for the frequency-aware storage partitioning described in Section A.5. The same domain corpora are also used throughout Phases 0–III of the staged retrofitting procedure, while Phase IV switches to downstream supervised data.

Let the final tokenized lexical inventory be

$$\mathcal{V}_{\text{lex}} = \left\{ \mathbf{x}^{(j)} = (x_1^{(j)}, \ldots, x_{n_j}^{(j)}) \right\}_{j=1}^{N}. \tag{8}$$

Each unique tokenized variant is retained independently, preserving distinctions induced by the native tokenizer, such as capitalization and leading-space context. Its external representation is initialized by mean pooling the corresponding pretrained token embeddings:

$$\mathbf{e}^{(j)} = \frac{1}{n_j} \sum_{i=1}^{n_j} \mathbf{E}_{\text{tok}} \left[x_i^{(j)} \right], \qquad \mathbf{E}_{\text{lex}} = \left[\mathbf{e}^{(1)}; \ldots; \mathbf{e}^{(N)} \right], \tag{9}$$

where $\mathbf{E}_{\text{tok}}$ denotes the pretrained input-embedding table. The resulting lexical memory table $\mathbf{E}_{\text{lex}}$ remains frozen throughout retrofitting, while adaptation is performed through the projection, gating, and stage-specific backbone parameters described in Section A.4.

A.2 Lexical Normalization and Routing Details

Each entry in $\mathcal{V}_{\text{lex}}$ is treated as an independent routing key in native token-ID space. Surface-form normalization is performed before router construction and includes tokenizer-sensitive variants such as capitalization and leading-space forms, together with domain-specific morphological and orthographic variants, including inflectional forms, hyphen or punctuation normalization, legal citation variants, and Greek-symbol or chemical-name transformations. Runtime routing itself operates only on the resulting token sequences and does not apply additional string-level normalization.

We compile the final tokenized inventory into an Aho–Corasick automaton whose alphabet consists of native tokenizer IDs. Each trie path represents one complete lexical variant, and each terminal state stores the corresponding row index in $\mathbf{E}_{\text{lex}}$. Let q_t denote the automaton state after observing token x_t:

$$q_t = \delta(q_{t-1}, x_t), \tag{10}$$

where δ is the deterministic state-transition function. The output set associated with q_t contains all valid lexical spans terminating at position t. When several matches share the same endpoint, LCM retains the longest suffix match, consistent with the rule defined in Section 3.2.

A matched lexical representation is activated only at the terminal position of its span. Specifically, if an entry spans $[s, t]$, its memory row is assigned to position t, after all constituent tokens have been observed. This end-of-span assignment preserves causal compatibility while allowing subsequent tokens to access the completed entity representation through ordinary causal attention. Matches terminating at different positions are resolved independently: LCM does not impose global non-overlapping segmentation, so an earlier valid entity may remain activated even if a longer overlapping entity is completed at a later position. This prefix-consistent rule avoids relying on future tokens when making routing decisions.

The router therefore produces a deterministic token-level mapping

$$R(X) = \left\{ \left(j_t^\star, \mathbb{I}_{\text{hit},t} \right) \right\}_{t=1}^{L}, \tag{11}$$

where $j_t^\star$ is the selected lexical row index when a valid match terminates at t. The same routing rule is compatible with both prompt prefill and incremental causal processing. The selected row $\mathbf{E}_{\text{lex}}[j_t^\star]$ is then passed to the hierarchical injection mechanism described in Section 3.3. Complexity details are provided in Section A.6.

A.3 Mixed-Precision Stability

Introducing a new memory pathway into a pretrained backbone can perturb activation scales during mixed-precision retrofitting. LCM therefore confines the numerically sensitive compatibility and gating operations to a localized FP32 path while leaving the surrounding backbone computation in its native mixed-precision format.

For the compatibility function $s(\cdot, \cdot)$ in Equation (6), we apply a stabilized RMS normalization,

$$\text{SN}(\mathbf{x}; \mathbf{w}) = \mathbf{w} \odot \left[\mathbf{x} \left(\max \left(\frac{1}{d} \sum_{i=1}^{d} x_i^2, \epsilon_{\text{safe}} \right) \right)^{-1/2} \right], \tag{12}$$

where $\mathbf{w}$ is a learnable scale parameter and $\epsilon_{\text{safe}} = 10^{-6}$. The normalized compatibility score is then instantiated as

$$s(\mathbf{h}, \mathbf{k}) = \frac{\text{SN}(\mathbf{h})^\top \text{SN}(\mathbf{k})}{\sqrt{d} + \epsilon_s}, \qquad \epsilon_s = 10^{-5}. \tag{13}$$

As specified in Equation (6), the learnable gate bias is added to this score before the logit is clipped to $[-\tau, \tau]$, with $\tau = 15$, and passed through the sigmoid. Safe-RMSNorm, normalized similarity, bias addition, logit clipping, and sigmoid evaluation are performed in FP32; the key/value projections remain in the backbone's native mixed-precision pipeline.

We further bound unusually large memory activations before residual integration. Specifically, the projected value is clipped element-wise as

$$\bar{\mathbf{v}}_t^{(m)} = \text{Clip}\left(\mathbf{v}_t^{(m)}, -\eta, \eta \right), \qquad \eta = 50, \tag{14}$$

and $\bar{\mathbf{v}}_t^{(m)}$ is used in place of $\mathbf{v}_t^{(m)}$ in Equation (7). The post-injection hidden state is additionally bounded to $[-\gamma, \gamma]$, with $\gamma = 65000$, as a conservative safeguard against extreme activation excursions. These operations affect only the memory interaction pathway and do not modify the backbone architecture.

A.4 Decoupled Training Details

LCM uses a staged retrofit procedure that separates domain lexical adaptation, memory alignment, backbone adaptation, gate calibration, and downstream task adaptation. Each stage isolates a distinct component of memory integration before subsequent components are jointly optimized. Throughout all stages, the lexical memory table $\mathbf{E}_{\text{lex}}$ remains frozen. Phases 0–III use the same large-scale unlabeled domain corpora introduced in Section A.1, while Phase IV switches to downstream supervised data. The corresponding trainable and frozen parameter groups are summarized in Table 6.

Phase 0: Domain Lexical Warm-Up. We first adapt the lexical input and output interfaces of the base model to the unlabeled domain corpus while keeping all Transformer blocks frozen. Specifically, the input embedding layer and language-model head are optimized, whereas the Transformer blocks and all LCM components remain fixed. LCM is disabled during this stage. This stage adapts the model to the domain lexical distribution without altering its contextual computation, providing the initial checkpoint for memory retrofitting.

Phase I: Memory Alignment. We freeze the backbone, lexical memory table, key projections, normalization modules, and gating components, and optimize only the value projections $\mathbf{W}_V^{(\text{early})}$ and $\mathbf{W}_V^{(\text{late})}$. For each lexical-hit endpoint, both value pathways are aligned to the corresponding Phase-0 input embedding using a masked mean-squared error:

$$\mathcal{L}_{\text{align}} = \sum_{m \in \{\text{early},\text{late}\}} \frac{1}{|\Omega|} \sum_{t \in \Omega} \left\| \mathbf{W}_V^{(m)} \mathbf{e}_t - \mathbf{E}_{\text{native}}[x_t] \right\|_2^2, \tag{15}$$

where $\Omega = \{t : \mathbb{I}_{\text{hit},t} = 1\}$, $\mathbf{e}_t$ denotes the retrieved lexical-memory vector, and $\mathbf{E}_{\text{native}}$ is the input-embedding table after Phase 0. This stage provides a common representation-space initialization for the memory value pathways before contextual adaptation.

Phase II: Forced Memory Utilization. We next optimize the input embeddings, all Transformer blocks, the language-model head, and the value projections, while keeping $\mathbf{W}_K$, the gate bias, Norm_H, Norm_K, and $\mathbf{E}_{\text{lex}}$ frozen. At lexical-hit positions, memory injection is forced open, $\alpha_t = 1$, whereas non-hit positions remain fixed at $\alpha_t = 0$.

During training, we apply endpoint-only token dropout to 30% of lexical-hit positions. For a matched lexical span, only its endpoint token is replaced by <|pad|>; all preceding subtokens are retained. Training labels are copied from the original sequence before this corruption is applied. The language-modeling loss is therefore computed against the original tokens over all non-padding target positions:

$$\mathcal{L}_{\text{II}} = \mathcal{L}_{\text{LM}}. \tag{16}$$

By fixing the gate open during this stage, the backbone adapts to using the injected lexical representation before selective gating is introduced.

Phase III: Gate Calibration. We then freeze the input embeddings, Transformer blocks, language-model head, value projections, and hidden-side normalization, and optimize $\mathbf{W}_K$, the gate bias, and Norm_K. The same 30% endpoint-only token-dropout procedure and pre-corruption language-modeling targets used in Phase II are retained.

To transition from forced memory utilization to learned conditional gating, the gate is annealed over the first $S_{\text{ann}} = 500$ optimization steps. For lexical-hit positions,

$$\widetilde{\alpha}_t^{(s)} = \eta_s + (1 - \eta_s)\, \alpha_t, \qquad \eta_s = \max\left(0,\, 1 - \frac{s}{500}\right), \tag{17}$$

where α_t is the learned gate output and s is the current optimization step. For non-hit positions, $\widetilde{\alpha}_t^{(s)} = 0$ throughout training.

Gate supervision is defined over the full input sequence using

$$y_t = \mathbb{I}_{\text{hit},t}. \tag{18}$$

Table 6: **Parameter states across the LCM training stages.** ✓ and × denote trainable and frozen parameters, respectively. "Forced" denotes fixed memory injection with $\alpha = 1$ at lexical-hit positions and $\alpha = 0$ otherwise.

Stage	Embeddings	Blocks	LM Head	$\mathbf{E}_{\text{lex}}$	W_V	W_K/Gate	Norm.
Phase 0	✓	×	✓	–	–	–	–
Phase I	×	×	×	×	✓	×	×
Phase II	✓	✓	✓	×	✓	Forced	×
Phase III	×	×	×	×	×	✓	✓
Phase IV	✓	✓	✓	×	✓	✓	✓

Let z_t denote the stabilized pre-sigmoid gate logit defined in Equation (6). We optimize a class-weighted binary cross-entropy:

$$\mathcal{L}_{\text{gate}} = \frac{1}{T} \sum_{t=1}^{T} \text{BCEWithLogits}\left(z_t,\, y_t;\, w_+ = 10\right), \tag{19}$$

where T denotes the number of valid input positions. The Phase-III objective combines sequence-level language modeling and gate calibration:

$$\mathcal{L}_{\text{III}} = \mathcal{L}_{\text{LM}} + \lambda_{\text{gate}} \mathcal{L}_{\text{gate}}, \qquad \lambda_{\text{gate}} = 1.0. \tag{20}$$

Here, $\mathcal{L}_{\text{LM}}$ is evaluated over all non-padding target positions, whereas $\mathcal{L}_{\text{gate}}$ is evaluated over all valid input tokens. This stage converts forced memory usage into selective, learned activation while keeping the previously adapted value pathway fixed.

Phase IV: Downstream Task Adaptation. Finally, we jointly optimize the backbone and all trainable LCM components on downstream supervised data. Token dropout is disabled in this stage, and the model receives the complete downstream input.

The task objective is the standard supervised fine-tuning loss over assistant-response tokens. Gate supervision is retained over the full input sequence, including both prompt and response tokens, with a reduced positive-class weight of $w_+ = 2$:

$$\mathcal{L}_{\text{IV}} = \mathcal{L}_{\text{SFT}} + \lambda_{\text{gate}} \mathcal{L}_{\text{gate}}, \qquad \lambda_{\text{gate}} = 1.0. \tag{21}$$

Compared with Phase III, which uses $w_+ = 10$ during gate calibration, Phase IV adopts the milder class weighting during downstream adaptation. The lexical memory table $\mathbf{E}_{\text{lex}}$ remains frozen throughout this stage.

A.5 Asynchronous Retrieval and Tiered Storage

LCM separates the full lexical datastore from accelerator-resident model parameters and retrieves only the lexical working set required by each batch. To reduce host-side lookup overhead, we employ **Pipelined Asynchronous Prefetching**: background data-loading workers perform lexical routing and memory lookup for subsequent batches while the accelerator processes the current batch. The retrieved representations are assembled into a batch-level working set and transferred to the accelerator before model execution, after which the hierarchical injection layers access the prefetched buffer directly without additional host-side lookup.

The datastore is organized using frequency-aware tiered storage based on the corpus-derived access statistics described in Section A.1. Entries are ranked by empirical access frequency, with the most frequently accessed 30% retained in host RAM and the remaining 70% stored in an SSD-backed memory-mapped tier. The long-tail entries are therefore preserved rather than discarded, while frequently accessed lexical representations remain available from the lower-latency storage tier.

Together, pipelined retrieval and tiered storage allow the lexical-memory capacity to scale without requiring the full datastore to reside in accelerator memory. Accelerator memory usage instead depends primarily on the retrieved batch working set, while the external datastore remains independently managed from the backbone. This separation also supports replacement of the lexical datastore without modifying the native tokenizer or backbone architecture. The corresponding routing and computational complexity are discussed in Sections A.2 and A.6.

A.6 Complexity Analysis

Let $N = |\mathcal{V}_{\text{lex}}|$ denote the number of lexical entries, n_j the tokenized length of entry j, and

$$S_{\text{lex}} = \sum_{j=1}^{N} n_j \tag{22}$$

the total number of token IDs stored across the lexical inventory. Constructing the token-level Aho–Corasick automaton is an offline operation with $\mathcal{O}(S_{\text{lex}})$ space and construction cost. For an input sequence of length L, runtime routing requires a single automaton scan with complexity

$$\mathcal{O}(L + Z), \tag{23}$$

where Z denotes the number of lexical matches emitted during the scan. Unlike explicit fixed-order N-gram enumeration, this cost does not scale with a predefined lookup-window order.

Because the lexical representations are initialized in the native embedding space, the memory dimension equals the backbone hidden dimension, i.e., $d_m = d$. The external lexical table therefore requires

$$\mathcal{O}(Nd) \tag{24}$$

storage, while the routing structure contributes $\mathcal{O}(S_{\text{lex}})$ additional space. Hence, the total external storage complexity is $\mathcal{O}(Nd + S_{\text{lex}})$. Under the tiered implementation in Section A.5, the full table is not resident on the accelerator. Instead, a batch of size B and sequence length L uses a prefetched lexical working set of at most $\mathcal{O}(BLd)$ accelerator memory, which is independent of the total datastore size N.

LCM introduces two injection stages, each with one key and one value projection, $\mathbf{W}_K^{(m)}, \mathbf{W}_V^{(m)} \in \mathbb{R}^{d\times d}$. The dominant additional parameter count is therefore

$$P_{\text{LCM}} = 4d^2 + \mathcal{O}(d), \tag{25}$$

where the lower-order term accounts for normalization scales and gate parameters. In the current implementation, key and value projections are evaluated densely over the batch sequence and subsequently masked at inactive positions. The resulting projection cost is $\mathcal{O}(BLd^2)$ per injection stage, and remains $\mathcal{O}(BLd^2)$ asymptotically for the two-stage architecture. Dense batched projections are used to preserve contiguous matrix operations rather than introducing sparse gather/scatter execution.

Pipelined prefetching and tiered placement do not change these asymptotic bounds; instead, they reduce effective wall-clock retrieval overhead and avoid accelerator memory growth proportional to the full lexical datastore.

B Additional Experimental Details

B.1 Benchmark and Evaluation Protocols

Benchmark Overview. Table 7 summarizes the benchmark splits, supervision sources, and evaluation metrics used in our experiments.

Data Splits and Supervision. For MedQA, CaseHOLD, and ContractNLI, we use the official training splits for supervised adaptation and the official test splits for evaluation. Because ground-truth labels for the official MedMCQA test set are not publicly available, we reserve the complete official development split exclusively for evaluation. For Chemistry, ChemBench is evaluated on its complete multiple-choice subset, identified by the presence of answer options, while MMLU-C comprises the `college_chemistry` and `high_school_chemistry` subsets of MMLU. Neither ChemBench nor MMLU-C is used for supervised adaptation. Chemistry supervision is instead provided by SciQ-Chem, a chemistry subset obtained through manual example-level screening of the official SciQ training split; no SciQ test examples are used. For datasets used in supervised adaptation, we cap the training split at 20,000 examples. Within each domain, VocabExp, LPA, Engram, and LCM use identical downstream supervision. The domain lexical inventories are constructed independently of benchmark evaluation data from curated terminology resources, as detailed in Section A.1.

Table 7: **Benchmark and evaluation protocols.** Official data partitions are preserved whenever available. ChemBench and MMLU-C are used exclusively for evaluation, with chemistry supervision provided by SciQ-Chem, constructed from the official SciQ training split.

Domain	Benchmark	Evaluation Split	Supervision	Metric
Medicine	MedQA	Official test	Official train	Accuracy
	MedMCQA	Official development	Official train	Accuracy
Law	CaseHOLD	Official test	Official train	Accuracy
	ContractNLI	Official test	Official train	Accuracy
Chemistry	ChemBench	Complete MC subset	SciQ-Chem	Accuracy
	MMLU-C	College + HS Chemistry	SciQ-Chem	Accuracy

Table 8: **Evaluation–corpus overlap audit.** Each cell reports the number and percentage of evaluation examples whose complete selected text occurs as a contiguous substring in the corresponding Phase 0–III domain corpus. For multiple-choice tasks, the primary input is the question and the complete input additionally includes all answer options; gold labels and answer keys are never included.

Benchmark	N	Exact Primary	Norm. Primary	Exact Complete	Norm. Complete
MedQA	1,273	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
MedMCQA	4,183	15 (0.36%)	66 (1.58%)	0 (0.00%)	0 (0.00%)
CaseHOLD	5,314	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
ContractNLI	2,091	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
ChemBench	2,542	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
MMLU-C	303	2 (0.66%)	4 (1.32%)	0 (0.00%)	0 (0.00%)

Preprocessing and Evaluation. All methods use the same task-specific prompt construction, preprocessing, and evaluation examples within each benchmark. We apply a common decoding and answer-scoring procedure across methods and report accuracy as the primary metric. Results are averaged over three independent runs, with standard deviations reported in the corresponding tables, and the overall *Avg.* is the unweighted macro-average across the six evaluation benchmarks. No evaluation example is used for optimization or model selection.

B.2 Contamination and Lexical-Inventory Provenance Audit

Audit Protocol. We audit potential overlap between the unlabeled domain corpora used during retrofitting and all downstream evaluation sets. Each benchmark is compared against its corresponding domain corpus using both exact and normalized full-string matching. For multiple-choice benchmarks, we audit the question alone and the complete model-visible input consisting of the question and all answer options; for other tasks, we analogously distinguish the benchmark-specific primary input from the complete model-visible input. Gold labels, correct-answer indices, and target responses are excluded from all queries. Normalized matching applies Unicode NFKC normalization, lowercasing, punctuation normalization, and whitespace collapse. This audit tests deterministic string overlap only and does not attempt to detect paraphrastic or semantic overlap.

Overlap Results. Across all 15,706 evaluation examples, we detect no exact or normalized full-input overlap with the corresponding domain corpora. The only nonzero matches occur for the primary-input field of MedMCQA and MMLU-C. For MedMCQA, these matches are short domain-generic prompts such as "Steroids:" and "Vitamin K:"; none remains a match once the answer options are included. Similarly, the four normalized MMLU-C matches are short generic or symbolic expressions, including one match introduced only after punctuation normalization. Thus, under the exact and normalized matching procedures above, we observe no complete evaluation-example overlap in any of the six benchmarks.

Lexical-Inventory Provenance. We separately trace the construction of the external lexical inventory. The implemented pipeline intersects an external domain-term whitelist with terms observed in the corresponding unlabeled domain corpus, then constructs lexical representations and tiered storage solely from the resulting lexical table. Benchmark evaluation files, gold labels, answer keys, and target responses are not consumed by this pipeline. The repository does not contain the upstream procedure used to generate the whitelist itself, so its source-level construction cannot be independently verified from the released code. Accordingly, the present audit establishes the absence of evaluation-data usage in the implemented lexical-memory construction pipeline, while treating upstream whitelist provenance as outside the scope of this repository-level verification.

B.3 Baseline Implementation Details

Standard Fine-Tuning. *SFT* denotes standard supervised fine-tuning of the domain-warmed backbone without external memory or lexical augmentation.

DAPT+SFT. *DAPT+SFT* is designed to isolate whether LCM's gains can be explained by additional post-warmup domain-data exposure alone. Starting from the same Phase 0 checkpoint, it performs standard causal language modeling on the same domain corpus used for LCM's post-warmup adaptation, followed by the identical downstream SFT and evaluation protocol. It uses no lexical routing, external memory, endpoint dropout, alignment objective, or gate supervision.

Vocabulary Expansion. *VocabExp* uses the same domain lexical inventory as LCM, but represents each lexical entry directly as an additional tokenizer symbol. Newly introduced rows in both the input embedding matrix and LM head are initialized by mean-pooling the pretrained embeddings of the entry's original native subtokens. Vocabulary expansion is performed before domain warm-up so that the new lexical parameters are trained under the same corpus and optimization budget before downstream adaptation. This baseline tests whether explicitly encoding the same lexical inventory in the model vocabulary is sufficient without introducing an external memory pathway.

Lexical Prompt Augmentation. *LPA* exposes the same lexical identities through the input prompt rather than through hidden-state injection. For each example, lexical entries matched by the same domain inventory are listed in a short terminology prefix preceding the task input. The augmentation contains only the recognized lexical forms and does not provide definitions, descriptions, or factual evidence. LPA therefore serves as an information-matched prompt-level control for testing whether explicitly exposing lexical identity is sufficient without an external hidden-state memory interface.

Engram. *Engram* (Cheng et al., 2026b) is implemented as a domain-retrofitted conditional-memory baseline on the same Qwen2.5 backbone used by LCM. To enable a controlled comparison, we preserve Engram's defining bounded hash-based n-gram addressing while matching the downstream backbone, domain lexical inventory, memory capacity, and optimization budget to LCM. At inference time, continuous 2–4-token n-grams are extracted from the native token sequence and mapped to memory buckets using MurmurHash-based addressing. Each bucket stores the set of lexical entries associated with the n-grams routed to that location; duplicate term indices are removed, and the corresponding full-term vectors are mean-pooled to form the bucket representation. When multiple valid buckets are active at a token position, their representations are aggregated by masked mean pooling before being passed to the memory interface.

Engram uses the same dual-depth hidden-state injection architecture as LCM, including independent depth-specific projection and gating parameters, and its hash-table capacity is matched to the number of LCM lexical slots within each domain. Engram and LCM share the same Phase 0 checkpoint and downstream supervision, while their memory pathways are trained independently using the same Phase I–IV optimization schedule. This controlled construction isolates the difference between bounded hash-based n-gram addressing and LCM's deterministic variable-length lexical routing under a shared retrofit setting.

Retrieval-Augmented Generation. *RAG* provides a practical factual-retrieval baseline rather than an information-matched lexical control. We use BGE-M3 as the dense retriever and FAISS for nearest-neighbor search over domain-specific retrieval corpora. The retrieval query consists of the

complete question together with its context, when available, and the top three retrieved passages are appended to the model input under a fixed retrieval budget.

For Medicine, the retrieval corpus is derived from the *Medical Wikipedia English* corpus released by NLP-FBK (Ferrazzi et al., 2025). Documents are segmented at natural paragraph boundaries, with non-content sections such as references and external links removed. For Law, the retrieval corpus combines legal definitions and statutory material from Cornell Wex (Legal Information Institute), the United States Code (Office of the Law Revision Counsel), and the *First Edition* of Black's Law Dictionary (Black, 1891). For Chemistry, the corpus is constructed from PubChem (Kim et al., 2025) using the top 200,000 compounds by Compound Identifier (CID) ranking. Common names, standardized IUPAC names, molecular formulas, aliases, and selected compound attributes are converted into textual records for dense retrieval.

LCM+RAG reuses exactly the same retrieval pipeline and retrieved-evidence prompt as the standalone RAG baseline. No additional joint training or RAG-specific adaptation is performed: the retrieved prompt is supplied directly to the trained LCM checkpoint at inference time. This setting therefore tests whether external lexical memory provides complementary benefit when factual evidence is already available in the input.

Shared Controls and Fairness. Within each domain, VocabExp, LPA, Engram, and LCM use the same lexical inventory and identical downstream supervision. SFT, LPA, Engram, and LCM are initialized from the same Phase 0 domain-warmed checkpoint, while VocabExp follows the same domain corpus and warm-up optimization budget after vocabulary expansion. Engram and LCM additionally use matched memory capacities and the same Phase I–IV optimization schedule. Downstream optimization budgets are matched whenever applicable, and all methods are evaluated using the same benchmark examples, decoding procedure, and scoring pipeline.

B.4 Cross-Architecture Native-Engram Validation

Argon Variants. We conduct an additional validation on Argon-0.5B (Gong, 2026), a pretrained model with native Engram integration, to evaluate LCM beyond the Qwen2.5 retrofit setting. We consider three variants. *Argon (Native Engram)* retains the original pretrained Engram modules. *Argon w/o Engram* disables Engram at model construction time, such that no Engram modules, parameters, gates, or forward path are instantiated; all remaining backbone components are unchanged. This variant is therefore a post-hoc Engram-removal control rather than a model pretrained without Engram. *Argon w/ LCM* uses the same Engram-disabled backbone and replaces native Engram with LCM.

LCM Integration on Argon. LCM is adapted to Argon's tokenizer and 1024-dimensional hidden space, with the domain lexical inventory re-tokenized and the lexical memory and projection interfaces reconstructed accordingly. Because both Argon-0.5B and the Qwen2.5-0.5B backbone used in our main experiments contain 24 Transformer blocks, we retain the same post-block injection locations, after Blocks 2 and 10, for the early and late LCM pathways. Native Engram remains disabled in this variant throughout training and evaluation, so the comparison isolates LCM as a replacement memory mechanism on the same Argon backbone.

Matched Domain Adaptation. All three variants first undergo the same Phase 0 domain warm-up. We then match their post-warmup domain-data exposure and optimization-step budget before downstream adaptation. Argon (Native Engram) and Argon w/o Engram each receive $6{,}000$ steps of continued causal language-model training on the same unlabeled domain corpus; native Engram remains active and trainable in the former and is absent in the latter. Argon w/ LCM instead uses the standard LCM Phase I–III retrofit, whose $2{,}000 + 3{,}000 + 1{,}000$ steps yield the same $6{,}000$-step adaptation budget. The three settings subsequently use identical domain-specific supervision and evaluation protocols. Optimization hyperparameters follow the shared settings in Section B.5; objectives remain method-specific.

Table 9: **Optimization configuration for decoupled retrofitting.** Learning rates are reported separately for pretrained backbone parameters and trainable LCM components.

Phase	Trainable Parameters	LR (Backbone / LCM)	Duration
Phase 0	Embeddings, LM Head	$1e{-5}$ / –	2k steps
Phase I	W_V	– / $1e{-4}$	2k steps
Phase II	Embeddings, Blocks, LM Head, W_V	$1e{-5}$ / $1e{-4}$	3k steps
Phase III	W_K/Gate, Norm.	– / $1e{-4}$	1k steps
Phase IV	Embeddings, Blocks, LM Head, W_V, W_K/Gate, Norm.	$1e{-5}$ / $1e{-4}$	1 epoch

B.5 Optimization and Reproducibility Settings

Optimization Configuration. All stages use AdamW with a weight decay of 0.01 and a cosine learning-rate schedule with a 10% warmup ratio. We use differential learning rates for the pretrained backbone and newly introduced LCM components: backbone parameters use 1×10^{-5}, while trainable LCM parameters use 1×10^{-4}. DAPT+SFT uses the backbone learning rate of 1×10^{-5} during continued domain adaptation. Unless otherwise specified, the Argon experiments follow the corresponding Qwen2.5 optimization settings. In particular, continued adaptation for Argon (Native Engram) and Argon w/o Engram uses 1×10^{-5}, while Argon w/ LCM retains the backbone/LCM differential learning rates. Phase-specific settings are summarized in Table 9; objectives and parameter-freezing rules are specified in Section A.4.

Shared Training Budgets. All Qwen2.5 variants share the same Phase 0 domain warm-up unless their construction requires vocabulary expansion before warm-up. SFT and LPA proceed directly from the Phase 0 checkpoint, whereas DAPT+SFT performs $6{,}000$ steps of standard causal language modeling on the same post-warmup domain corpus used by LCM before downstream SFT. This matches LCM's additional domain-data exposure and post-warmup update count, but not computational cost, since the objectives and trainable components differ. Engram and LCM are independently trained from the shared Phase 0 checkpoint using the same Phase I–IV schedule, while VocabExp uses the same Phase 0 corpus and warm-up budget after vocabulary expansion.

For the cross-architecture Argon validation, all three variants undergo the same Phase 0 domain warm-up and matched post-warmup domain-data exposure. Argon (Native Engram) and Argon w/o Engram receive $6{,}000$ steps of continued causal language modeling, while Argon w/ LCM uses the standard Phase I–III retrofit for the same total number of updates. All variants subsequently use identical downstream supervision and matched optimization budgets. The external lexical memory remains frozen throughout LCM retrofitting.

Compute Environment. All experiments are conducted on NVIDIA A100 80GB GPUs using BF16 precision.

B.6 Ablation Protocols

Injection-Architecture Variants. We evaluate four controlled variants to isolate the effect of injection depth and depth-specific parameterization. *Early injection only* and *Late injection only* retain a single lexical-memory pathway at the early or late injection depth, respectively. *Dual-depth, shared* activates both injection depths but shares the complete memory interface across them, including W_V, W_K, the gating parameters, and normalization layers. *Dual-depth, independent* corresponds to the full LCM architecture and uses separate depth-specific interfaces at the two injection locations. All variants use the same pretrained backbone, external lexical memory, deterministic variable-length lexical routing, training data, and retrofit schedule; only the injection configuration and parameter sharing are changed.

Training-Procedure Comparison. To isolate the effect of decoupled retrofitting, we compare the full LCM architecture under *Naive E2E retrofit* and *Decoupled staged retrofit*. Both settings start from the same Phase 0 domain-warmed checkpoint, use the same domain corpus, lexical memory, downstream supervision, and total optimization budget corresponding to Phases I–III, and keep E_{lex} frozen throughout. The naive E2E baseline jointly optimizes the backbone together with all

Table 10: Detailed per-benchmark ablation results on Qwen2.5-0.5B. We report mean accuracy and standard deviation over three runs. Architecture variants differ only in their injection configuration and parameter sharing, while the training ablation compares naive joint optimization with the proposed decoupled staged retrofit under the same full LCM architecture.

Variant	MedQA	MedMCQA	CaseHOLD	ContractNLI	ChemBench	MMLU-C	Avg.
Injection Architecture							
Early injection only	$37.3_{\pm.47}$	$40.9_{\pm.57}$	$73.5_{\pm.24}$	$80.6_{\pm.67}$	$38.1_{\pm.28}$	$38.5_{\pm.83}$	51.5
Late injection only	$36.4_{\pm.31}$	$40.2_{\pm.43}$	$73.1_{\pm.71}$	$79.3_{\pm.43}$	$37.7_{\pm.28}$	$38.2_{\pm.50}$	50.8
Dual-depth, shared	$35.8_{\pm.63}$	$39.6_{\pm.31}$	$72.2_{\pm.40}$	$78.9_{\pm.53}$	$37.0_{\pm.59}$	$38.0_{\pm.33}$	50.3
Dual-depth, independent	$\mathbf{38.3}_{\pm.31}$	$\mathbf{41.6}_{\pm.43}$	$\mathbf{75.1}_{\pm.51}$	$\mathbf{82.4}_{\pm.72}$	$\mathbf{38.5}_{\pm.47}$	$\mathbf{39.1}_{\pm.50}$	**52.5**
Training Procedure							
Naive E2E retrofit	$33.9_{\pm.39}$	$38.5_{\pm.28}$	$71.4_{\pm.62}$	$79.7_{\pm.19}$	$35.2_{\pm.28}$	$33.8_{\pm.66}$	48.8
Decoupled staged retrofit	$\mathbf{38.3}_{\pm.31}$	$\mathbf{41.6}_{\pm.43}$	$\mathbf{75.1}_{\pm.51}$	$\mathbf{82.4}_{\pm.72}$	$\mathbf{38.5}_{\pm.47}$	$\mathbf{39.1}_{\pm.50}$	**52.5**

Table 11: **Lexical content specificity on Qwen2.5-0.5B.** All variants use the same lexical router, injection architecture, lexical-hit positions, and training schedule; only the term–representation correspondence is changed. Results are mean accuracy (%) over three runs.

Variant	MedQA	MedMCQA	CaseHOLD	ContractNLI	ChemBench	MMLU-C	Avg.
Correct lexical mapping	$\mathbf{38.3}_{\pm.31}$	$\mathbf{41.6}_{\pm.43}$	$\mathbf{75.1}_{\pm.51}$	$\mathbf{82.4}_{\pm.72}$	$\mathbf{38.5}_{\pm.47}$	$\mathbf{39.1}_{\pm.50}$	**52.5**
Shuffled lexical mapping	$33.0_{\pm.31}$	$36.8_{\pm.19}$	$72.2_{\pm.45}$	$80.6_{\pm.38}$	$35.3_{\pm.63}$	$36.7_{\pm.38}$	49.1
Random per-term representation	$31.4_{\pm.67}$	$39.0_{\pm.26}$	$72.8_{\pm.34}$	$77.2_{\pm.53}$	$36.1_{\pm.59}$	$34.2_{\pm.19}$	48.5
Shared representation	$30.2_{\pm.55}$	$33.9_{\pm.41}$	$70.1_{\pm.15}$	$76.1_{\pm.24}$	$35.7_{\pm.24}$	$34.5_{\pm.38}$	46.8

trainable memory-interface parameters from the first step, while applying the same 30% endpoint token dropout used in the later LCM retrofit stages and optimizing the joint language-modeling and gate BCE objective throughout. In contrast, the proposed retrofit decomposes the same training budget into representation alignment, forced memory utilization, and gate calibration with stage-specific objectives and parameter schedules, as described in Section A.4.

Detailed Per-Benchmark Results. The main text reports domain-level averages for compactness. We therefore provide the complete per-benchmark results in Table 10, including mean accuracy and standard deviation over three runs. The *Dual-depth, independent* architecture under the *Decoupled staged retrofit* corresponds to the full LCM model and thus appears in both ablation groups.

Gate-Utilization Measurement. The gate-utilization analysis in Figure 3 is conducted on the Qwen2.5-0.5B medical setting. We track the early gate at Layer 2 and the late gate at Layer 10 during training on the medical domain corpus. At each measurement step, gate activation probabilities are averaged over lexical-hit positions in the corresponding training batch, and the reported curves are then averaged across three independent runs. The staged and naive E2E models are evaluated on the same token-dropout training data, so the comparison reflects differences in memory-pathway utilization rather than differences in the observed examples.

B.7 Controlled Analysis Details

Lexical Content Specificity. To isolate whether LCM benefits from term-specific lexical representations rather than from lexical-hit signals alone, we keep the router, injection architecture, optimization schedule, lexical-hit positions, and downstream training fixed while perturbing only the correspondence between lexical terms and stored memory vectors. *Shuffled mapping* applies a fixed permutation over the original lexical vectors, preserving the vector distribution while breaking term–representation correspondence; *random per-term* assigns each term a fixed random vector; and *shared representation* maps all terms to a single common vector. All memory vectors remain frozen throughout adaptation. As shown in Table 11, the correct mapping achieves the highest accuracy on all six benchmarks and a 52.5 macro-average, compared with 49.1, 48.5, and 46.8 for shuffled, random per-term, and shared representations, respectively. These controls indicate that LCM's gains rely on preserving term-specific lexical information, rather than merely detecting lexical boundaries or assigning stable identifiers to matched terms.

Table 12: **Routing-factorial analysis on Qwen2.5-0.5B.** The 2×2 design independently varies span construction (bounded 2–4-gram vs. variable-length full-term matching) and addressing (hashed vs. exact). All variants share the same lexical inventory, memory representations, injection architecture, and training schedule. Results are mean accuracy (%) over three runs.

Routing Variant	MedQA	MedMCQA	CaseHOLD	ContractNLI	ChemBench	MMLU-C	Avg.
Hashed 2–4-gram	$35.5_{\pm.86}$	$40.1_{\pm.36}$	$73.8_{\pm.68}$	$80.5_{\pm.48}$	$35.9_{\pm.51}$	$36.2_{\pm.67}$	50.3
Exact 2–4-gram	$37.2_{\pm.31}$	$40.7_{\pm.31}$	$74.4_{\pm.17}$	$81.0_{\pm.38}$	$37.8_{\pm.39}$	$39.0_{\pm.33}$	51.7
Hashed variable-length	$35.8_{\pm.08}$	$39.9_{\pm.77}$	$74.5_{\pm.41}$	$81.0_{\pm.53}$	$35.7_{\pm.31}$	$36.3_{\pm.33}$	50.5
Exact variable-length	$\mathbf{38.3}_{\pm.31}$	$\mathbf{41.6}_{\pm.43}$	$\mathbf{75.1}_{\pm.51}$	$\mathbf{82.4}_{\pm.72}$	$\mathbf{38.5}_{\pm.47}$	$\mathbf{39.1}_{\pm.50}$	**52.5**

Table 13: **Routing diagnostics for the factorial variants.** Active Collision is the evaluation-active key-level collision rate, i.e., the fraction of activated keys that share a hash bucket with another activated key. Vectors/Hit counts the bucket vectors averaged in the forward pass. Source Entries/Hit counts the underlying lexical entries represented by those frozen bucket vectors. Sample Hit Rate is the fraction of evaluation examples containing at least one lexical hit.

Routing Variant	Active Collision (%)	Vectors / Hit	Source Entries / Hit	Sample Hit Rate (%)
Hashed 2–4-gram	10.2	1.34	355	100.0
Exact 2–4-gram	–	1.34	345	100.0
Hashed variable-length	2.0	1.00	48	88.5
Exact variable-length	–	1.00	**1**	88.6

Routing Factorial Analysis. We disentangle the effects of span construction and addressing using a 2×2 factorial design that crosses bounded 2–4-gram versus variable-length full-term routing with hashed versus exact addressing. All four variants use the same lexical inventory, memory representations, injection architecture, and adaptation schedule; only the routing mechanism is changed. As shown in Table 12, replacing hashed addressing with exact lookup improves bounded 2–4-gram routing from 50.3 to 51.7 macro-average accuracy. Under exact addressing, replacing bounded spans with variable-length full-term access yields a further improvement to 52.5, with the exact variable-length configuration achieving the highest mean accuracy on all six benchmarks.

The routing diagnostics in Table 13 explain these effects more directly. Evaluation-active key-level collision decreases from 10.2% for hashed 2–4-gram routing to 2.0% for hashed variable-length routing. More importantly, removing hashing alone does not eliminate the ambiguity of bounded short-span keys: exact 2–4-gram routing still represents 345 underlying lexical entries per hit on average. Variable-length matching reduces this mixture to 48 source entries per hit under hashed addressing, while exact variable-length routing resolves each hit to a single lexical entry. The forward pass itself averages only 1.34 bucket vectors per bounded hit and one vector per variable-length hit; the substantially larger source-entry counts therefore reflect lexical information already compressed into the frozen bucket representations rather than the number of vectors averaged online.

For completeness, inventory-level collision is substantially higher than the evaluation-active rate. Across the domain-specific inventories, the global key-level collision rate is 88.7–95.1% for hashed 2–4-gram routing and 63.3–63.4% for hashed variable-length routing. We report evaluation-active collision in Table 13 because it characterizes collisions among keys actually exercised by the evaluation sets, whereas the global rate reflects overall hash-table occupancy. Taken together, these results separate two sources of routing ambiguity: bounded short-span keys remain lexically non-specific even under exact addressing, while hashing introduces additional aliasing that is removed by deterministic exact lookup.

Fragmentation-Conditioned Performance. We finally examine whether LCM's gains vary with the severity of lexical fragmentation. For each evaluation example, we apply the production lexical router and define fragmentation severity by the maximum native-subtoken length $L_{\max}$ among all matched lexical terms. Examples with no lexical match are reported separately; hit examples are partitioned into $L_{\max} = 2$, $L_{\max} = 3$–4, and $L_{\max} \geq 5$. These bins are fixed from the pre-computed length distribution before inspecting model accuracy. As shown in Table 14, LCM's advantage increases with fragmentation severity. Relative to SFT, the gain rises from $+1.6$ points for two-subtoken terms to $+2.7$ for 3–4 subtokens and $+3.6$ for terms spanning at least five subtokens.

Table 14: **Pooled performance by lexical coverage and fragmentation on Qwen2.5-0.5B.** Results average the same three runs as Table 1 and pool all six benchmarks. Hit examples contain at least one lexical match, with fragmentation defined by the maximum native-subtoken length $L_{\max}$ among matched terms. Bins are fixed before evaluating model performance.

Subset #Examples	SFT	Engram	LCM	LCM–SFT	LCM–Engram	
No hit	1,785	37.3	36.8	**38.1**	+0.8	+1.3
Hit	13,921	57.2	58.2	**60.0**	+2.8	+1.8
$L_{\max} = 2$	2,228	42.3	**44.5**	43.9	+1.6	−0.6
$L_{\max} = 3$–4	6,272	60.3	61.4	**63.0**	+2.7	+1.6
$L_{\max} \geq 5$	5,421	59.7	60.1	**63.3**	+3.6	+3.2

Table 15: **Detailed inference-efficiency results.** Latency and slowdown are reported as mean ± std over three paired runs. GPU memory denotes peak allocated memory.

Domain	Setting	Lat. (ms)	vs. SFT	Peak GPU (GiB)
Medicine	SFT	$161.3_{\pm 5.8}$	1.000×	1.31
	LCM-Sync	$213.9_{\pm 1.6}$	$1.327_{\pm 0.045}\times$	1.39
	LCM-Pipe	$177.0_{\pm 4.7}$	$1.098_{\pm 0.029}\times$	1.39
Law	SFT	$180.3_{\pm 6.1}$	1.000×	1.75
	LCM-Sync	$263.3_{\pm 1.7}$	$1.461_{\pm 0.041}\times$	1.92
	LCM-Pipe	$194.8_{\pm 6.1}$	$1.080_{\pm 0.032}\times$	1.92
Chemistry	SFT	$151.3_{\pm 3.8}$	1.000×	1.09
	LCM-Sync	$194.9_{\pm 2.2}$	$1.289_{\pm 0.041}\times$	1.13
	LCM-Pipe	$169.6_{\pm 7.0}$	$1.121_{\pm 0.048}\times$	1.13

Relative to Engram, the pooled differences are -0.6, $+1.6$, and $+3.2$ points across increasing fragmentation bins. LCM also improves more over SFT on lexical-hit than no-hit examples ($+2.8$ vs. $+0.8$ points). These pooled trends are consistent with larger benefits from complete-term lexical access at higher fragmentation levels, while potentially reflecting benchmark-composition differences.

B.8 System Efficiency and Scalability

B.8.1 Inference Efficiency

Benchmarks and Evaluation Setup. We evaluate inference efficiency on one representative benchmark per domain: MedQA for Medicine, CaseHOLD for Law, and ChemBench for Chemistry. All measurements use Qwen2.5-0.5B with the corresponding final domain-adapted checkpoints. LCM-Sync and LCM-Pipe use the same Phase-IV LCM checkpoint and access the same complete domain lexical store, differing only in retrieval scheduling. LCM-Sync performs lexical routing, lookup, and batch-local working-set construction on the inference critical path, whereas LCM-Pipe uses **Pipelined Asynchronous Prefetching** to prepare the next batch while the GPU processes the current batch.

Measurement Protocol. All latency and peak-memory measurements use a single NVIDIA A100 80GB GPU under the same runtime configuration, with batch size 8, BF16 inference, left padding, greedy decoding, and at most 15 generated tokens. Each run includes 20 warm-up batches followed by 100 measured batches, and each setting is repeated three times. To reduce execution-order effects, the paired runs rotate the order of SFT, LCM-Sync, and LCM-Pipe. End-to-end batch latency is measured from batch retrieval through synchronized generation, including worker-side waiting exposed to the main process. Peak allocated GPU memory is measured with (`torch.cuda.max_memory_allocated()`).

Detailed Results. LCM-Sync incurs 1.29–1.46× the latency of SFT, whereas LCM-Pipe reduces this to 1.08–1.12×. Relative to LCM-Sync, pipelining lowers latency by 17.3%, 26.0%, and 13.0%

Table 16: **Frequency-based lexical-store scaling.** Nested subsets are formed by descending domain-corpus frequency.

Scale	Entries	Freq. Mass	Store (GiB)	Lat. (ms)
25%	31,853	0.9927	0.114	$207.2_{\pm 1.4}$
50%	63,705	0.9986	0.228	$206.8_{\pm 2.3}$
75%	95,557	0.9996	0.342	$205.6_{\pm 3.5}$
100%	127,409	1.0000	0.457	$206.1_{\pm 2.4}$

Table 17: **Full lexical-store characterization across domains.** Store denotes the FP32 lexical vector payload; GPU denotes peak allocated GPU memory under the production LCM pipeline.

Domain	Entries	Store (GiB)	Peak GPU (GiB)
Medicine	295,871	1.060	1.39
Law	127,409	0.457	1.92
Chemistry	135,205	0.485	1.13

for Medicine, Law, and Chemistry, respectively. Peak allocated GPU memory increases by only 0.08, 0.17, and 0.04 GiB.

LCM-Sync and LCM-Pipe produce identical predictions, evaluation accuracy, and lexical-routing outcomes across all nine paired runs, showing that pipelining changes retrieval scheduling without altering model computation or outputs.

B.8.2 Lexical-Memory Scalability

Controlled Scaling Protocol. We evaluate lexical-memory scalability on Law/CaseHOLD by varying the size of the legal lexical store while holding the evaluation workload and lexical-hit pattern fixed. We first identify all lexical entries accessed by the complete CaseHOLD evaluation set under the production routing procedure. These entries are retained at every scale, while additional entries are progressively added according to their corpus-frequency ranking to construct nested 25%, 50%, 75%, and 100% stores. Each retained entry preserves its tier assignment from the full production datastore. This construction changes only the size of the off-device lexical store while keeping the routed lexical entries and model predictions unchanged across scales.

Controlled Scaling Results. As shown in Table 4(b), increasing the lexical store from 31.9K to 127.4K entries enlarges the off-device vector payload from 0.114 to 0.457 GiB, while peak allocated GPU memory remains unchanged at 1.916 GiB. End-to-end latency remains within 205.3–208.0 ms and exhibits no systematic growth with store size. Lexical routing outcomes and generated predictions are identical across all controlled variants.

Frequency-Based Scaling. We additionally evaluate a realistic lexicon-growth setting using nested frequency-ranked subsets of the legal lexical store. The top 25%, 50%, 75%, and 100% subsets contain 31,853, 63,705, 95,557, and 127,409 entries, respectively. These subsets retain 99.27%, 99.86%, 99.96%, and 100% of the lexical frequency mass observed in the domain corpus. Across the four scales, peak allocated GPU memory remains 1.916 GiB and latency varies by less than 1%, consistent with the controlled scaling results.

Full-Lexicon Characterization. The complete lexical stores contain 295,871 entries for Medicine, 127,409 for Law, and 135,205 for Chemistry, corresponding to FP32 vector payloads of 1.060, 0.457, and 0.485 GiB, respectively. Their peak allocated GPU memory under the production LCM pipeline is 1.39, 1.92, and 1.13 GiB. The largest host-side lexical store therefore does not correspond to the largest GPU footprint, consistent with the off-device memory organization.

GPU Residency. The complete lexical vector store remains in host memory, with the head resident in RAM and the remaining vectors accessed through SSD-backed memory mapping. For each batch,

Table 18: **Detailed lexical-fragmentation statistics.** UNIQUE and OCC. denote the numbers of distinct matched lexical entries and total lexical occurrences, respectively. Mean-U and Mean-O are the unique-entry- and occurrence-weighted mean numbers of native subtokens per entry. Median, P95, and Max are computed over unique matched entries.

Benchmark	Unique	Occ.	Mean-U	Mean-O	Median	P95	Max
MedQA	4,541	15,427	3.29	2.89	3	6	11
MedMCQA	4,931	11,554	3.23	2.99	3	6	14
CaseHOLD	9,663	163,529	2.90	2.47	2	7	17
ContractNLI	1,453	143,862	2.36	2.32	2	4	13
ChemBench	737	4,024	3.31	2.41	3	7	15
MMLU-C	243	558	3.14	2.84	3	5	10

the GPU receives a dense FP32 lexical tensor of shape $[B, L, d]$, together with lexical indices of shape $[B, L]$. The resulting GPU-side lexical working set is therefore $O(BLd)$, with an additional $O(BL)$ index footprint, whereas the complete lexical store scales as $O(N_{\text{lex}}d)$ off-device. The same batch-local lexical tensor is shared by the early and late LCM injection pathways.

B.9 ADDITIONAL LEXICAL FRAGMENTATION STATISTICS

Evaluation Protocol. We compute lexical-fragmentation statistics on the complete evaluation set of each benchmark, using the same prompt construction and deterministic routing procedure as in LCM inference. For every matched lexical entry, tokenized length is obtained directly from the native-tokenizer ID sequence stored with the corresponding lexical-memory entry; no decoding or re-tokenization is applied. We do not subsample evaluation examples or weight entries by their corpus frequencies.

The main-text analysis assigns equal weight to each distinct lexical entry matched at least once in a benchmark. Let $\mathcal{U}_D$ denote the set of unique matched entries for benchmark D, and let $L(e)$ denote the number of native subtokens associated with entry e. The unique-entry mean is

$$L_{\text{uniq}}(D) = \frac{1}{|\mathcal{U}_D|} \sum_{e \in \mathcal{U}_D} L(e). \tag{26}$$

This statistic characterizes the tokenization structure of the lexical entries covered by a benchmark without allowing a small number of frequently repeated entries to dominate the estimate.

Weighting and Length Distribution. As a complementary view, we also compute an occurrence-weighted mean that reflects the lexical fragmentation encountered over all routing events. Let c_e be the number of times entry e is matched in benchmark D. We define

$$L_{\text{occ}}(D) = \frac{\sum_{e \in \mathcal{U}_D} c_e L(e)}{\sum_{e \in \mathcal{U}_D} c_e}. \tag{27}$$

Table 18 reports both weighting schemes together with the full unique-entry length statistics. Across all six benchmarks, L_{occ} is lower than L_{uniq}, indicating that frequently repeated lexical entries tend to be shorter than the broader set of distinct matched entries. Importantly, this weighting difference does not change the qualitative conclusion that LCM routinely encounters multi-subtoken lexical items under the native tokenizer.

The gap between the two means is largest on benchmarks whose routing distribution contains many repeated short entries. For example, ChemBench has a unique-entry mean of 3.31 subtokens but an occurrence-weighted mean of 2.41, while CaseHOLD shifts from 2.90 to 2.47. The percentile statistics also show that mean length alone understates the lexical tail: 95th-percentile lengths range from 4 to 7 subtokens, and the longest matched entries range from 10 to 17 subtokens across benchmarks.

Coverage. Fragmentation statistics are conditioned on successful lexical matches rather than on evaluation examples themselves. Samples containing no lexical match therefore remain part of

the benchmark evaluation but do not contribute an entry to the statistics above. The numbers of such samples are 4 for MedQA, 709 for MedMCQA, 0 for CaseHOLD, 0 for ContractNLI, 934 for ChemBench, and 150 for MMLU-C. This conditioning affects only the descriptive fragmentation analysis and does not alter the evaluation sets or reported task accuracies.