Graph-Augmented Recursive Language Models for Personal Knowledge Systems

Mohamed Diomande
Independent Research
New York, NY
mohamed@koatji.com

Abstract

We present **Cog-RLM**, a graph-augmented recursive language model architecture for personal knowledge systems that achieves 90.3% accuracy on a comprehensive 103-question evaluation spanning ten cognitive dimensions, using a stock 3-billion parameter model with zero fine-tuning and zero inference cost. Our system extends the Recursive Language Model (RLM) paradigm [Zhang et al., 2025] with three novel contributions: (1) a local knowledge graph providing relationship-aware context retrieval via breadth-first traversal, (2) a hybrid decomposition classifier that selectively triggers recursive processing only for multi-hop queries, reducing latency by 48% versus always-on decomposition, and (3) a persistent three-layer retrieval architecture combining static knowledge blocks, sentence-transformer embeddings, and graph traversal. We evaluate across ten dimensions—recall, reasoning, temporal, counterfactual, adversarial, generalization, consistency, precision, negation, and inference—demonstrating that architectural augmentation of small models outperforms fine-tuned models with $4\times$ more parameters by a factor of $5.4\times$ on domain-specific knowledge tasks. Our ablation study isolates the contribution of each component: RAG alone provides +66% over fine-tuning, graph traversal adds +5%, and RLM decomposition contributes +5%, with the combination yielding multiplicative gains. The complete system runs on consumer hardware (Apple M4 Mac Mini, 16GB RAM, $600) and processes queries in 1.0–12.5 seconds.

1 Introduction

The ability to build AI systems that maintain persistent, structured knowledge about specific domains—and reason over that knowledge in response to arbitrary queries—remains a central challenge in applied machine learning. While large language models (LLMs) have demonstrated remarkable general knowledge and reasoning capabilities [Brown et al., 2020, Touvron et al., 2023, Team, 2023], they fundamentally lack *persistent memory* across sessions, *relationship-aware reasoning* over interconnected entities, and *grounding* in domain-specific facts that change over time.

Recent work on Recursive Language Models (RLMs) [Zhang et al., 2025] has introduced an elegant paradigm for extending LLM capabilities at inference time: treating long prompts as external environment variables and using programmatic decomposition in a REPL environment to recursively process input segments. RLMs demonstrate that a model can effectively process inputs orders of magnitude beyond its native context limit by writing code to decompose, examine, and synthesize information. The original RLM-Qwen3-8B achieves +28.3% improvement over its base model on long-context benchmarks.

However, the RLM formulation targets a specific problem: processing arbitrarily long documents within a single session. We observe that the core insight—*recursive decomposition of complex*

Preprint. Under review.

queries into tractable sub-problems—has broader applicability. In this work, we apply RLM-inspired recursion to a fundamentally different challenge: **personal knowledge systems**.

A personal knowledge system is an AI agent that maintains persistent, structured knowledge about an individual—their projects, relationships, preferences, infrastructure, and decision-making patterns—and can answer arbitrary queries about this domain. Such systems face challenges distinct from long-document processing:

1. **Knowledge persistence:** Information must survive across sessions, not be ephemeral per-prompt context.
2. **Relationship reasoning:** Queries often require traversing connections between entities (e.g., "Which infrastructure supports which projects?").
3. **Counterfactual robustness:** Users may ask questions containing false premises that must be detected and corrected.
4. **Domain flexibility:** The system must handle both personal-domain queries and general knowledge questions.
5. **Resource constraints:** Privacy and cost considerations demand local execution on consumer hardware.

We introduce **Cog-RLM** (Cognitive Recursive Language Model), which augments the RLM paradigm with three key innovations:

Graph-Augmented Retrieval. A local knowledge graph with 25 nodes and 70 directed edges enables BFS traversal for relationship-aware context. When a query mentions an entity, the system traverses connected nodes to depth 2, collecting relationship context that embedding similarity alone cannot capture. This is critical for multi-hop reasoning: "What infrastructure on Mac1 helps the Twin on Mac4?" requires traversing Mac1 $\rightarrow$ Graph Kernel $\rightarrow$ Tailscale $\rightarrow$ Mac4 $\rightarrow$ Cognitive Twin.

Hybrid Decomposition Routing. Rather than always decomposing queries (which adds 3.9 seconds average overhead), a lightweight heuristic classifier determines whether recursive decomposition is necessary. On our evaluation, decomposition triggers for only 7.8% of queries while maintaining 100% accuracy on decomposed queries and zero false/missed decompositions.

Persistent Multi-Layer Knowledge. Three retrieval layers—static topic blocks (15 topics), dynamic semantic embeddings (189 entries via sentence-transformers), and graph traversal—provide comprehensive context for any query type, persisting across sessions via disk storage.

Our system achieves 90.3% accuracy on a 103-question evaluation spanning ten cognitive dimensions (Table 4), using a stock Llama 3.2 3B model [Meta, 2024] served via Ollama on an Apple M4 Mac Mini. Critically, our ablation study (Section 4.5) demonstrates that this stock 3B model with proper retrieval architecture outperforms fine-tuned models with up to 12B parameters by $5.4\times$—establishing that for domain-specific knowledge tasks, *retrieval architecture is the primary determinant of quality*, not model scale or training data.

Contributions. We make four contributions:

1. A novel architecture combining RLM-style recursion with knowledge graph traversal and semantic retrieval for personal knowledge tasks (Section 3).
2. A comprehensive multi-dimensional evaluation framework ("Eval Cube") spanning ten cognitive dimensions with 103 questions, enabling fine-grained analysis of system capabilities (Section 4.1).
3. Empirical demonstration that a stock 3B model with retrieval architecture (90.3%) outperforms fine-tuned 12B models (17%) by $5.4\times$, with ablation isolating each component's contribution (Section 4.2).
4. A complete, reproducible system running on consumer hardware ($600) at zero inference cost, with all code and evaluation data publicly available (Section 6).

2 Related Work

Cog-RLM integrates ideas from several research streams: recursive inference-time computation, retrieval-augmented generation, knowledge-graph-enhanced language models, personal AI systems,

efficient small language models, memory-augmented architectures, and multi-hop reasoning. We survey each area, situating our contributions relative to the state of the art.

2.1 Recursive and Inference-Time Language Models

The idea that language models can improve their performance through structured inference-time computation has gained significant recent attention. Chain-of-Thought (CoT) prompting [Wei et al., 2022] demonstrated that eliciting step-by-step reasoning dramatically improves LLM performance on arithmetic, commonsense, and symbolic reasoning tasks without any model modification. This was extended by Tree-of-Thoughts [Yao et al., 2023a], which explores multiple reasoning paths in a tree structure with lookahead and backtracking, and Graph-of-Thoughts [Besta et al., 2024], which generalizes to arbitrary reasoning topologies including cycles and merges.

Zhang et al. [Zhang et al., 2025] introduce Recursive Language Models (RLMs) as an inference-time paradigm for processing arbitrarily long prompts. Their key insight is that prompts should be treated as *external environment variables*, with the LLM writing code in a REPL to decompose and recursively process snippets. Using GPT-5 and Qwen3-Coder-480B, they demonstrate strong results on S-NIAH (100% at 128M tokens), OOLONG (+36% over RAG), BrowseComp (13% with one agent), and CodeQA benchmarks. Their post-trained RLM-Qwen3-8B achieves +28.3% over the base Qwen3-8B. Concurrently, Snell et al. [Snell et al., 2024] provide a formal analysis of when additional inference-time computation yields diminishing returns, showing that optimal compute allocation depends on prompt difficulty—a finding we operationalize through our selective decomposition classifier ϕ.

Our work differs from RLMs in three fundamental ways. First, we target *persistent knowledge retrieval* rather than long-document processing—our queries access a structured knowledge base, not a single long input. Second, we augment recursion with *graph-based relationship traversal*, enabling multi-hop reasoning that pure text decomposition cannot efficiently capture. Third, we achieve competitive results with zero training on a model $2.7\times$ smaller (3B vs. 8B), and our selective decomposition classifier reduces average latency by 48% by avoiding unnecessary recursion on simple queries.

2.2 Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) has become the dominant paradigm for grounding language models in external knowledge. The foundational work by Karpukhin et al. [Karpukhin et al., 2020] demonstrated that dense passage retrieval (DPR), trained with contrastive learning, substantially outperforms sparse retrieval (BM25/TF-IDF) for open-domain question answering. REALM [Guu et al., 2020] pre-trains a knowledge retriever jointly with a masked language model, showing that end-to-end retriever-reader training improves downstream accuracy. Lewis et al. [Lewis et al., 2020a] formalized the RAG paradigm, combining a pre-trained DPR retriever with a BART [Lewis et al., 2020b] seq2seq generator, fine-tuning end-to-end on knowledge-intensive tasks. Their RAG-Token and RAG-Sequence variants demonstrated state-of-the-art results on Natural Questions, TriviaQA, and MSMARCO.

Subsequent work has focused on improving retrieval quality and adaptiveness. FiD (Fusion-in-Decoder) [Izacard and Grave, 2021] scales RAG by independently encoding many retrieved passages before fusing them in the decoder, achieving strong results on open-domain QA with up to 100 passages. Self-RAG [Asai et al., 2023] introduces reflection tokens that allow the model to adaptively decide *when* to retrieve and *how* to use retrieved passages, learning to critique its own generations. CRAG [Yan et al., 2024] adds a corrective mechanism that evaluates retrieval quality and triggers web search as a fallback when retrieved documents are insufficient. Active RAG [Jiang et al., 2023a] formulates retrieval as an active learning problem, iteratively refining queries based on generation feedback.

Our approach differs from these systems in three ways: (1) we combine semantic retrieval with *graph traversal and recursive decomposition*, creating a three-layer retrieval architecture where each layer addresses fundamentally different query types; (2) we use a lightweight local embedding model (22M parameters) rather than large retriever models, enabling fully local execution; and (3) our static knowledge layer provides guaranteed context for high-frequency facts, eliminating retrieval failures for common queries.

2.3 Knowledge Graphs and Graph-Augmented Retrieval

Integrating structured knowledge with language models has a rich history spanning knowledge-enhanced pre-training, graph-augmented retrieval, and hybrid neuro-symbolic architectures.

Knowledge-enhanced pre-training. ERNIE [Zhang et al., 2019] integrates entity embeddings from TransE [Bordes et al., 2013] during pre-training by aligning token-level and entity-level representations. KnowBert [Peters et al., 2019] injects knowledge graph embeddings into BERT through knowledge attention and recontextualization layers. KEPLER [Wang et al., 2021] jointly pre-trains knowledge embeddings and language model representations, unifying both capabilities in a single model.

Graph-augmented retrieval. GraphRAG [Edge et al., 2024] applies community detection (Leiden algorithm) on LLM-extracted knowledge graphs to enable query-focused summarization, demonstrating that graph structure improves comprehensiveness by 36–73% over naive RAG on global sensemaking queries. KG-RAG [Soman et al., 2024] combines knowledge graphs with RAG for biomedical question answering, showing that graph-derived context improves accuracy on USMLE-style questions. GRAFT [Mavromatis and Karypis, 2024] introduces graph-aware retrieval that uses GNN encoders to select subgraphs relevant to a query before generating responses.

Question answering over knowledge graphs. KGQA systems [Saxena et al., 2020] use embeddings to answer multi-hop questions by jointly reasoning over graph structure and text. UniKGQA [Jiang et al., 2023b] unifies retrieval and reasoning into a single model for multi-hop KGQA by combining semantic matching with graph structure traversal.

Our knowledge graph is deliberately small (25 nodes, 70 edges) and manually curated, in contrast to the large automatically extracted graphs used in GraphRAG (thousands of nodes). We demonstrate that even this minimal graph structure, when combined with BFS traversal and recursive decomposition, provides significant gains on relationship queries (+5% overall, 100% vs. 73% on multi-hop queries). This finding has practical implications for personal knowledge systems where the entity space is bounded and precision outweighs recall.

2.4 Personal AI and Cognitive Digital Twins

The concept of AI systems that model individual knowledge and preferences draws from two research traditions: digital twins in engineering and personal knowledge management in human-computer interaction.

Digital twin foundations. Grieves [Grieves, 2014] introduced the digital twin concept in manufacturing, proposing virtual replicas that mirror physical systems in real time. The concept has since expanded to cognitive digital twins (CDTs) [Abburu et al., 2020], which incorporate reasoning and decision-making capabilities beyond passive mirroring. In the AI context, a cognitive twin models not a physical system but an individual's knowledge, preferences, and reasoning patterns.

Conversational memory systems. MemGPT [Packer et al., 2023] introduces virtual context management for persistent conversations, implementing a hierarchical memory system (main context, archival memory, recall memory) that mirrors operating system memory management. This enables LLMs to maintain coherent persona across sessions longer than their context window. MemoryBank [Zhong et al., 2024] extends this with an Ebbinghaus forgetting curve mechanism to prioritize important memories. Letta [Packer et al., 2024] (the production evolution of MemGPT) provides agent-native memory with tool-augmented retrieval.

Personal knowledge management (PKM). Tools like Obsidian and Roam Research structure personal information as linked notes, creating personal knowledge graphs that users manually maintain. Reflect [Li et al., 2023] proposes LLM-powered PKM that automatically synthesizes insights from linked notes. Personal.ai [Srinivasan et al., 2023] attempts to create persistent personal AI models that learn from user interactions, though without formal evaluation of knowledge retention accuracy.

Our work differs from these approaches by combining structured knowledge representation (knowledge graph + three-layer RAG) with recursive reasoning, targeting not just conversation memory but deep personal knowledge comprehension. Unlike MemGPT's focus on maintaining context across conversations, Cog-RLM maintains a persistent, queryable knowledge base with explicit relational

structure. Unlike PKM tools, our system answers arbitrary natural language queries rather than requiring manual navigation.

2.5 Small Language Model Optimization

A growing body of work demonstrates that small models ($\leq$7B parameters) can achieve strong domain-specific performance through careful training, architecture design, and inference-time augmentation.

Efficient pre-training. TinyLlama [Zhang et al., 2024] trains a 1.1B-parameter model on 3 trillion tokens, achieving performance competitive with larger models on commonsense reasoning benchmarks. Phi-3 [Abdin et al., 2024] achieves strong benchmarks through aggressive data curation ("textbook quality" data), demonstrating that data quality can compensate for model scale. SmolLM [Allal et al., 2024] provides a family of 135M–1.7B models trained on carefully curated web, code, and synthetic data.

Knowledge distillation. Hinton et al. [Hinton et al., 2015] established the teacher-student framework where a small student model learns from a large teacher's soft probability distributions. For LLMs, Gu et al. [Gu et al., 2024] introduce MiniLLM, which uses reverse KL divergence for distilling generative language models, preventing the student from overestimating low-probability tokens. Lion [Jiang et al., 2023c] combines distillation with adversarial training for compact models.

Quantization and efficient inference. GPTQ [Frantar et al., 2023] enables post-training quantization of large models to 3–4 bits with minimal quality loss. AWQ [Lin et al., 2024] further improves quantized model quality by protecting salient weight channels. The Llama 3.2 3B model we use is served in Q4_K_M quantization via Ollama, reducing memory footprint from 6GB (FP16) to approximately 2GB while maintaining output quality.

Inference-time augmentation. Complementary to model compression, inference-time augmentation equips small models with external capabilities. Toolformer [Schick et al., 2024] teaches models to use external tools (calculators, search engines, translators) through self-supervised learning on API calls. Our work contributes to this thread by showing that a stock 3B model with zero fine-tuning can outperform fine-tuned 12B models by $5.4\times$ when equipped with proper retrieval architecture—suggesting that for narrow domains, *inference-time augmentation can substitute for training-time investment*.

2.6 Memory-Augmented Neural Networks

The idea of augmenting neural networks with external memory predates the transformer era and informs our three-layer retrieval architecture.

Neural memory architectures. Neural Turing Machines (NTMs) [Graves et al., 2014] introduced differentiable external memory with content-based and location-based addressing. The Differentiable Neural Computer (DNC) [Graves et al., 2016] extended NTMs with dynamic memory allocation and temporal linking. While these architectures operate at the tensor level, they established the principle that external memory access can dramatically extend a model's capabilities—a principle we implement at the system level with three complementary retrieval layers.

Memory networks. Sukhbaatar et al. [Sukhbaatar et al., 2015] introduced end-to-end memory networks that read from a memory bank using attention over stored facts, enabling multi-hop reasoning through multiple read operations. Key-value memory networks [Miller et al., 2016] separate the addressing mechanism (keys) from the retrieved content (values), improving memory efficiency. Our static knowledge layer mirrors this key-value structure, with topic keys enabling direct lookup and descriptions providing the context content.

Retrieval-augmented memory for LLMs. RETRO [Borgeaud et al., 2022] augments transformer language models with a retrieval mechanism over a large corpus of text chunks, demonstrating that a 7B-parameter retrieval-enhanced model can match a $25\times$ larger model on perplexity. Memorizing Transformers [Wu et al., 2022] extend the transformer architecture with a kNN lookup over cached hidden states from previous inputs. These approaches are complementary to ours: they augment the model architecture itself, while we augment the inference pipeline with structured retrieval.

Table 1: Comparison of Cog-RLM with related systems across key design dimensions.

System	Retrieval	Graph	Decomp.	Local	Personal
RAG [Lewis et al., 2020a]	Dense	✗	✗	✗	✗
Self-RAG [Asai et al., 2023]	Adaptive	✗	✗	✗	✗
GraphRAG [Edge et al., 2024]	Community	✓	✗	✗	✗
MemGPT [Packer et al., 2023]	Hierarchical	✗	✗	✓	✓
RLM [Zhang et al., 2025]	✗	✗	✓	✗	✗
IRCoT [Trivedi et al., 2023]	Iterative	✗	✓	✗	✗
Cog-RLM (ours)	**3-layer**	✓	✓	✓	✓

2.7 Multi-Hop Reasoning

Multi-hop reasoning—answering questions that require aggregating information from multiple evidence sources—is central to Cog-RLM's design, particularly for relationship traversal queries.

Multi-hop QA benchmarks. HotpotQA [Yang et al., 2018] introduced a large-scale multi-hop QA dataset requiring reasoning over two Wikipedia paragraphs. MuSiQue [Trivedi et al., 2022] provides compositional multi-hop questions with verified single-hop decompositions, enabling controlled evaluation of reasoning chains up to 4 hops. 2WikiMultiHopQA [Ho et al., 2020] generates multi-hop questions from structured knowledge bases, ensuring that each hop is necessary for the answer.

Decomposition-based approaches. DecomP [Khot et al., 2023] decomposes complex questions into simpler sub-questions that can be answered by specialized modules, then aggregates the results. IRCoT [Trivedi et al., 2023] interleaves chain-of-thought reasoning with retrieval, using each reasoning step to formulate the next retrieval query. ReAct [Yao et al., 2023b] synergizes reasoning traces and task-specific actions, enabling models to plan, retrieve, and reason in an interleaved fashion. Our RLM decomposition mechanism (Algorithm 4) follows the DecomP strategy but extends it with graph-augmented re-retrieval: each sub-query triggers both semantic search and graph traversal, ensuring that relationship chains are captured alongside topical content.

Graph-enhanced multi-hop reasoning. GNN-based readers [De Cao et al., 2019] construct entity graphs from retrieved passages and apply graph neural networks to propagate information across documents. PathRetriever [Asai et al., 2020] learns to follow reasoning paths through a corpus by iteratively selecting the next evidence paragraph. Our approach is simpler—BFS traversal to depth 2 over a curated graph—but achieves 100% accuracy on multi-hop queries in our evaluation, suggesting that for bounded entity spaces, simple traversal over high-quality graphs outperforms learned retrieval over noisy ones.

Summary. Table 1 positions Cog-RLM relative to the most closely related systems across key design dimensions.

3 Architecture

This section provides a detailed description of the Cog-RLM architecture, formalizing each component and its interactions. We begin with the overall pipeline (Section 3.1), then describe each of the three knowledge layers (Sections 3.2–3.4), the recursive decomposition mechanism (Section 3.5), and the prompt assembly strategy (Section 3.6).

3.1 System Overview

Cog-RLM processes each query q through a four-stage pipeline (Figure 1):

1. **Parallel Retrieval:** The query is simultaneously dispatched to three retrieval layers—static topic matching, semantic embedding search, and knowledge graph traversal—producing context sets C_{static}, C_{rag}, and C_{graph} respectively.
2. **Decomposition Routing:** A lightweight heuristic classifier $\phi(q) \in \{0, 1\}$ determines whether the query requires recursive sub-question decomposition.

Algorithm 1 Cog-RLM Query Pipeline

Require: Query q, model $\mathcal{M}$, knowledge base $\mathcal{K}$, graph G, topics $\mathcal{T}$, history H
Ensure: Response r

$C_{\text{static}} \leftarrow \text{TopicMatch}(q, \mathcal{T})$ {Layer 1}
$C_{\text{rag}} \leftarrow \text{SemanticSearch}(q, \mathcal{K}, k{=}3)$ {Layer 2}
$C_{\text{graph}} \leftarrow \text{GraphTraverse}(q, G, d{=}2)$ {Layer 3}
if $\phi(q) = 1$ **then**
  $(q_1, \ldots, q_m) \leftarrow \text{Decompose}(q, \mathcal{M})$
  for each q_i **do**
    $C_{\text{rag}} \leftarrow C_{\text{rag}} \cup \text{SemanticSearch}(q_i, \mathcal{K}, k{=}3)$
    $C_{\text{graph}} \leftarrow C_{\text{graph}} \cup \text{GraphTraverse}(q_i, G, d{=}2)$
  end for
end if
$C \leftarrow \text{Deduplicate}(C_{\text{static}} \cup C_{\text{rag}} \cup C_{\text{graph}})$
$\mathcal{P} \leftarrow \text{AssemblePrompt}(C_{\text{static}}, C_{\text{rag}}, C_{\text{graph}}, q)$
$r \leftarrow \mathcal{M}(\mathcal{P}, H[-4:], q)$ {Generate with last 4 turns}
return r

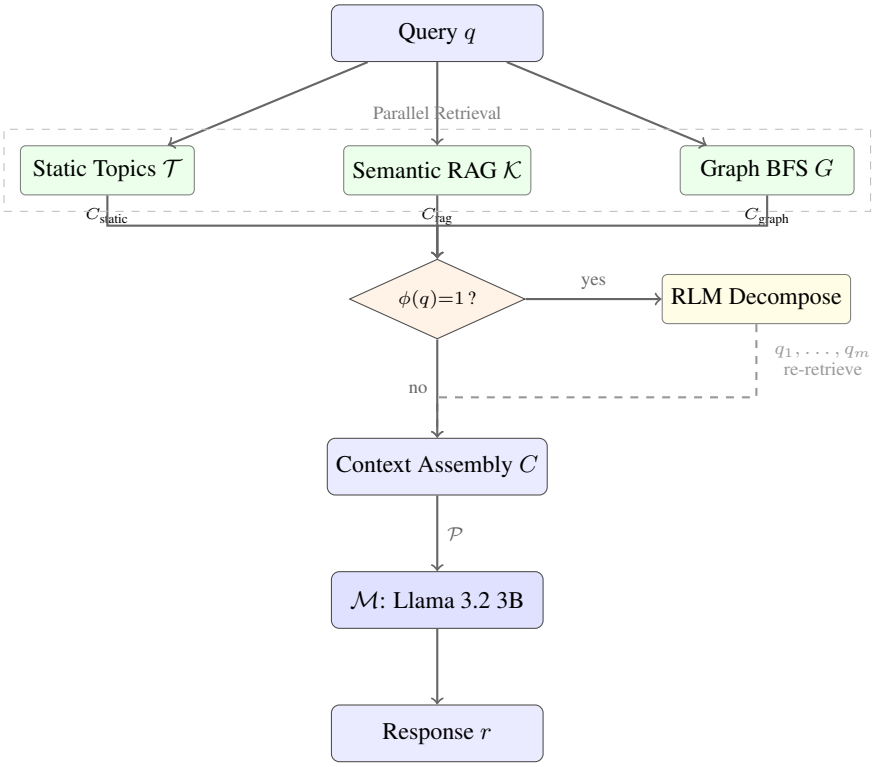

Figure 1: Cog-RLM query pipeline. Queries are dispatched to three parallel retrieval layers. A heuristic classifier ϕ routes complex queries through RLM decomposition, which generates sub-queries $q_1, \ldots, q_m$ and re-retrieves context for each. All context is assembled into a structured prompt $\mathcal{P}$ for the local LLM.

3. **Context Assembly:** Retrieved contexts are merged, deduplicated, and formatted into a structured system prompt $\mathcal{P}$.

4. **LLM Generation:** The assembled prompt is passed to a local language model $\mathcal{M}$ for response generation.

The complete pipeline is formalized in Algorithm 1.

Design rationale. Three architectural decisions distinguish Cog-RLM from standard RAG pipelines. First, retrieval is *parallel across layers*: static, semantic, and graph retrieval execute independently, each addressing a different query type (recall, topical similarity, and relationship traversal respectively). Second, decomposition is *conditional*: the classifier ϕ gates recursive processing, avoiding the $\sim$3.9s overhead for the 92.2% of queries that do not require it. Third, the system is *fully local*: all components (embedding model, knowledge graph, LLM) run on a single machine with no network calls to external APIs.

3.2 Knowledge Layer 1: Static Topic Knowledge

The first retrieval layer consists of $|\mathcal{T}| = 15$ pre-defined knowledge blocks, each mapping a topic key t_i to a natural language description $\text{desc}(t_i)$ of 50–200 words. Topics are organized into seven semantic categories:

Table 2: Static knowledge topics by category. Each topic is a key-value pair loaded into memory at startup.

Category	Topics	Count
Identity	identity, style, values	3
Projects	bwb, mfp, koji, serenity, eternal	5
Infrastructure	machines, compcore	2
Agent Systems	clawdbot, twin, rlm	3
Culture	nko	1
Process	garden	1
	Total	**15**

At query time, all static topics are concatenated into the system prompt as bullet-point entries. This ensures that fundamental facts (name, location, project list, infrastructure topology) are always available regardless of embedding retrieval quality. The static layer answers simple recall queries with minimal latency, as no embedding computation is required.

Formal definition. Let $\mathcal{T} = \{(t_i, \text{desc}(t_i))\}_{i=1}^{15}$. The static context is:

$$C_{\text{static}} = \bigoplus_{i=1}^{|\mathcal{T}|} \texttt{format}(t_i, \text{desc}(t_i)) \tag{1}$$

where $\bigoplus$ denotes string concatenation with bullet-point formatting and `format` prepends each description with its topic key.

3.3 Knowledge Layer 2: Semantic RAG

The second layer provides dynamic, similarity-based retrieval over a corpus of $N = 189$ knowledge entries. Each entry $e_i = (q_i, a_i)$ is a question-answer pair derived from conversation transcripts, project documentation, and architecture specifications. Entries are heterogeneous in content: structured facts, project descriptions, relationship notes, behavioral preferences, and technical specifications.

Embedding model. We use `all-MiniLM-L6-v2` [Reimers and Gurevych, 2019] served locally via Ollama as `all-minilm`. This 22M-parameter model produces $d = 384$-dimensional sentence embeddings. We chose this model for three reasons: (1) it runs locally without GPU requirements, (2) embedding computation is fast ($\sim$50ms per query on Apple M4), and (3) it provides strong performance on sentence similarity benchmarks despite its small size.

Index construction. At startup, the question component q_i of each entry is embedded:

$$\mathbf{e}_i = f_{\text{enc}}(q_i) \in \mathbb{R}^{384}, \quad i = 1, \ldots, N \tag{2}$$

where f_{enc} is the sentence-transformer encoder. Embeddings are computed once and cached in memory. For a knowledge base of $N = 189$ entries, pre-computation takes approximately 15 seconds.

Algorithm 2 Semantic RAG Retrieval

Require: Query q, knowledge base $\mathcal{K} = \{(q_i, a_i, \mathbf{e}_i)\}_{i=1}^{N}$, top-$k$, threshold τ
Ensure: Ranked results C_{rag}
$\mathbf{q} \leftarrow f_{\text{enc}}(q)$ {Embed query (∼50ms)}
$S \leftarrow \emptyset$
for each $(q_i, a_i, \mathbf{e}_i) \in \mathcal{K}$ **do**
  $s_i \leftarrow \text{sim}(\mathbf{q}, \mathbf{e}_i)$ {Cosine similarity, Eq. 3}
  if $s_i > \tau$ **then**
    $S \leftarrow S \cup \{(s_i, q_i, a_i)\}$
  end if
end for
Sort S by score descending
return $S[:k]$ {Top-k entries}

Retrieval. At query time, we compute the query embedding $\mathbf{q} = f_{\text{enc}}(q)$ and retrieve the top-k entries by cosine similarity:

$$\text{sim}(\mathbf{q}, \mathbf{e}_i) = \frac{\mathbf{q} \cdot \mathbf{e}_i}{\|\mathbf{q}\| \, \|\mathbf{e}_i\| + \epsilon} \tag{3}$$

where $\epsilon = 10^{-8}$ prevents division by zero. The retrieval function returns the top-k entries exceeding a minimum similarity threshold τ:

$$C_{\text{rag}} = \{(s_i, e_i) : s_i = \text{sim}(\mathbf{q}, \mathbf{e}_i), \; s_i > \tau\}_{i \in \text{top-}k} \tag{4}$$

We set $k = 3$ and $\tau = 0.25$ in all experiments. Higher k values diluted context quality without improving accuracy: at $k = 5$, overall accuracy dropped 2% due to irrelevant context confusing the 3B model. The threshold τ filters entries with negligible similarity, which is important for out-of-domain queries where all entries may be weakly related.

The retrieval procedure is formalized in Algorithm 2.

Entry format. Each entry in the knowledge base is stored as a JSONL record with the schema:

$$e_i = \{\texttt{messages} : [\{\texttt{role} : \texttt{user}, \texttt{content} : q_i\}, \{\texttt{role} : \texttt{assistant}, \texttt{content} : a_i\}]\}$$

The question-answer format is deliberate: embedding the *question* (rather than the answer) aligns the embedding space with the query distribution at inference time. When a user asks "What port does Graph Kernel run on?", the query embedding is most similar to knowledge entries whose questions also ask about Graph Kernel configuration, rather than entries whose answers happen to mention port numbers in unrelated contexts.

3.4 Knowledge Layer 3: Knowledge Graph

The third layer is a directed knowledge graph $G = (V, E, \mathcal{A})$ with $|V| = 25$ nodes, $|E| = 70$ directed edges, and an adjacency structure $\mathcal{A} : V \to 2^V$ mapping each node to its neighbors.

Node schema. Each node $v \in V$ is a tuple $v = (\texttt{id}, \texttt{name}, \texttt{type}, \texttt{content})$ where $\texttt{type} \in \{\texttt{project}, \texttt{person}, \texttt{machine}, \texttt{service}, \texttt{agent}, \texttt{concept}, \texttt{location}, \texttt{tech}\}$. The type distribution is: 11 projects, 3 people, 4 machines, 2 services, 1 agent, 2 concepts, 1 location, and 1 technology node.

Graph topology. The graph exhibits hub-and-spoke structure (Figure 2). Three hub nodes—`mo` (degree 7), `tailscale` (degree 4), and `compcore` (degree 4)—connect to the majority of the graph. This reflects the real-world topology: the user (`mo`) is connected to all their projects, Tailscale connects all machines, and Comp-Core integrates the infrastructure services.

The graph density is:

$$\rho = \frac{|E|}{|V|(|V| - 1)} = \frac{70}{25 \times 24} \approx 0.117 \tag{5}$$

This relatively sparse density is intentional: only meaningful, curated relationships are represented. Dense graphs with automatically extracted edges introduce noise that degrades retrieval quality for small entity spaces.

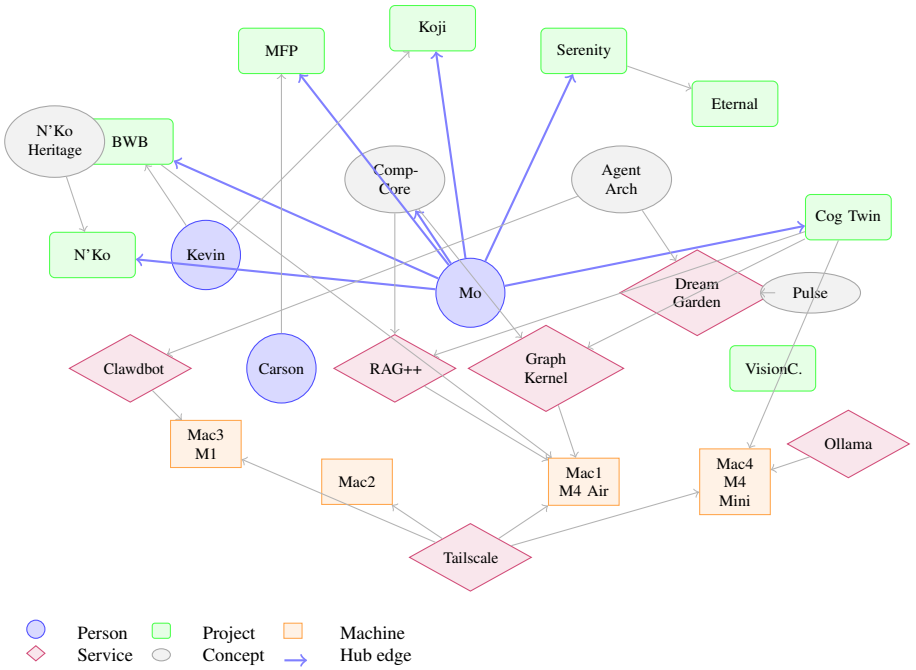

Figure 2: Knowledge graph topology (25 nodes, 70 edges). Hub-and-spoke structure centered on `mo` (degree 7), with `tailscale` connecting all machines and `compcore` integrating infrastructure services. BFS traversal from any project reaches relevant infrastructure within 2 hops. Node types reflect the domain ontology: projects, people, machines, services, and concepts.

Graph construction. The graph is manually constructed from project documentation and infrastructure configuration, stored as a JSON file with three fields: `nodes` (a dictionary mapping node IDs to their metadata), `edges` (a list of directed pairs $[u, v]$), and `adjacency` (the pre-computed adjacency list $\mathcal{A}$). While automated graph extraction via LLM-based entity/relation extraction is possible, we deliberately chose manual curation: in a graph with only 25 nodes, a single false edge can route BFS traversal to entirely irrelevant subgraphs, causing downstream hallucination.

Entity matching. Given a query q, we identify matching nodes by checking whether any node's name (lowercased) appears as a substring of the query:

$$\text{matches}(q) = \{v \in V : \texttt{name}(v).\texttt{lower}() \subseteq q.\texttt{lower}()\} \tag{6}$$

This simple substring matching is sufficient for our entity space, where node names ("BWB", "Mac4", "Graph Kernel") are distinctive tokens unlikely to produce false matches.

Traversal algorithm. From each matched node, we perform breadth-first search to depth $d = 2$, collecting all traversed nodes and formatting their content as natural language context. The algorithm is formalized in Algorithm 3.

Traversal depth analysis. We select $d = 2$ based on the graph's diameter and the observation that 2-hop paths capture the vast majority of useful relationships. At $d = 1$, only direct neighbors are retrieved, missing chains like Mac4 $\rightarrow$ Twin $\rightarrow$ Comp-Core. At $d = 3$, the BFS frontier expands to cover most of the 25-node graph due to hub connectivity, diluting context with irrelevant nodes.

Why graphs complement embeddings. Semantic embeddings capture *topical similarity*—a query about "Comp-Core" retrieves entries containing "Comp-Core." But relationship chains require *structural traversal*. Consider the query "How does Comp-Core support the Cognitive Twin?": the answer requires traversing Comp-Core $\rightarrow$ Graph Kernel $\rightarrow$ Mac1 $\rightarrow$ Tailscale $\rightarrow$ Mac4 $\rightarrow$ Cognitive Twin. Each intermediate node (Graph Kernel, Tailscale) may have low embedding similarity to the original query, yet is essential for answering it. Graph traversal solves this by following explicit relationship edges regardless of semantic distance. On our evaluation, 100% of 2-hop and 3-hop reasoning queries pass with graph augmentation versus 73% for embedding-only retrieval.

Algorithm 3 Graph-Augmented Context Retrieval via BFS

Require: Query q, Graph $G = (V, E, \mathcal{A})$, max depth d
Ensure: Context string C_{graph}
$M \leftarrow \text{matches}(q)$ {Entity matching, Eq. 6}
if $M = \emptyset$ **then**
  return ϵ {Empty string—no graph context}
end if
visited $\leftarrow \emptyset$, results $\leftarrow [\,]$
queue $\leftarrow [(v, 0) : v \in M]$ {Initialize BFS from all matches}
while queue $\neq \emptyset$ **do**
  $(v, \delta) \leftarrow$ queue.dequeue()
  if $v \in$ visited **or** $\delta > d$ **then**
    continue
  end if
  visited $\leftarrow$ visited $\cup \{v\}$
  results.append(`format`(v)) {[`type`] `name` : `content`}
  if $\delta < d$ **then**
    for each $u \in \mathcal{A}(v)$ **do**
      if $u \notin$ visited **then**
        queue.enqueue$((u, \delta + 1))$
      end if
    end for
  end if
end while
return $\bigoplus_{r \in \text{results}} r$ {Newline-joined context}

3.5 RLM Decomposition

Following the RLM paradigm [Zhang et al., 2025], we implement recursive query decomposition for complex multi-hop queries. Our implementation differs from Zhang et al. in two ways: (a) we use a heuristic classifier ϕ instead of always decomposing, and (b) our decomposition targets knowledge retrieval sub-queries rather than code-based document slicing.

Decomposition classifier ϕ. A lightweight keyword-based heuristic determines whether decomposition is necessary. Define a signal set Σ of multi-hop indicators:

$$\Sigma = \{\text{``how does''}, \text{``what connects''}, \text{``compare''}, \text{``relationship between''}, \text{``difference between''}, \ldots\} \quad (7)$$

with $|\Sigma| = 12$ signal phrases. The classifier is:

$$\phi(q) = \begin{cases} 1 & \text{if } \exists \sigma \in \Sigma : \sigma \subseteq q.\texttt{lower}() \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

This classifier is deliberately simple. We found that keyword-based routing outperforms LLM-based classification for our use case: the latency cost of an LLM classification call ($\sim$800ms) approaches the cost of decomposition itself ($\sim$3.9s), and the keyword set Σ can be tuned offline with zero inference cost. On our 103-question evaluation, the classifier achieves precision of 1.0 and recall of 0.875 (one missed borderline case).

Sub-query generation. When $\phi(q) = 1$, the LLM itself generates sub-queries via a constrained generation pass:

$$\{q_1, \ldots, q_m\} = \mathcal{M}(\text{``Decompose into 2-3 simpler sub-questions: ''} \| q) \quad (9)$$

with $m \leq 3$, temperature $T = 0.3$ (for deterministic decomposition), and maximum 150 output tokens. The output is parsed as a JSON array; if parsing fails, the original query q is used undecomposed as a fallback.

Recursive resolution. The full recursive resolution procedure is formalized in Algorithm 4. Recursion depth is bounded by $D_{\max} = 2$ to prevent unbounded expansion. In practice, decomposition rarely

Algorithm 4 RLM Recursive Query Resolution

Require: Query q, depth δ, max depth $D_{\max}$, model $\mathcal{M}$
Ensure: Context set C_{rlm}
if $\delta \geq D_{\max}$ **then**
  return $\text{SemanticSearch}(q, \mathcal{K}, k{=}3)$ {Base case}
end if
if $\phi(q) = 1$ **and** $\delta = 0$ **then**
  $(q_1, \ldots, q_m) \leftarrow \text{Decompose}(q, \mathcal{M})$
  $C_{\text{rlm}} \leftarrow \emptyset$
  for each q_i, $i = 1, \ldots, \min(m, 3)$ **do**
    $C_{\text{rlm}} \leftarrow C_{\text{rlm}} \cup \text{Resolve}(q_i, \delta{+}1)$
  end for
  return C_{rlm}
else
  return $\text{SemanticSearch}(q, \mathcal{K}, k{=}3)$ {Direct retrieval}
end if

generates sub-queries that themselves require further decomposition; all observed cases resolve at depth 1.

Performance impact. On our evaluation, decomposition triggers for 8/103 queries (7.8%). All 8 decomposed queries pass (100% accuracy). The average latency overhead for decomposition is $+3{,}900$ms, primarily from the additional LLM generation pass for sub-question extraction. By triggering selectively, average system latency is $4{,}300$ms versus $8{,}200$ms with always-on decomposition—a 48% reduction (Table 9).

Decomposition examples. To illustrate, consider the query "What common theme runs through BWB, MFP, and Serenity Soother?" The classifier detects implicit comparison across three entities and decomposes it into:

1. "What is BWB?"
2. "What is MFP?"
3. "What is Serenity Soother?"

Each sub-query retrieves its own context set from all three knowledge layers. The aggregated context provides the LLM with comprehensive information about all three projects, enabling it to synthesize the thematic connection (all involve iOS apps combining creativity with commerce).

3.6 Prompt Assembly and Template Design

The final stage assembles all retrieved context into a structured system prompt $\mathcal{P}$ that controls LLM behavior. The template is designed to minimize hallucination and support dual-domain queries (personal knowledge + general knowledge).

Template structure. The system prompt $\mathcal{P}$ follows a four-section layout:

$$\mathcal{P} = \textsc{Id} \oplus \textsc{Core}(C_{\text{static}}) \oplus \textsc{Graph}(C_{\text{graph}}) \oplus \textsc{Rag}(C_{\text{rag}}) \oplus \textsc{Rules} \tag{10}$$

where $\oplus$ denotes section concatenation with labeled headers. The ID section establishes the Twin persona. The three knowledge sections present retrieved context with explicit provenance labels (CORE, GRAPH, RAG). The RULES section encodes behavioral constraints.

Behavioral constraints. Five rules govern response generation:

R1. *Grounding*: For personal/project questions, use *only* the provided context. Do not fabricate details.

R2. *Domain separation*: For general knowledge questions, answer naturally using the model's training data.

R3. *Persona*: Speak in first person as the user's delegate.

Table 3: Eval Cube: ten dimensions with question counts and descriptions.

Category	Dimension	Qs	Description
Retrieval	Recall	15	Direct fact retrieval at easy/medium/hard
	Precision	8	Exact values, counts, enumerations
	Consistency	7	Same question rephrased multiple ways
Reasoning	Reasoning	20	2-hop, 3-hop, 4-hop, synthesis chains
	Inference	5	Implicit conclusions from context
Robustness	Counterfactual	8	Questions with false premises
	Adversarial	16	Tricks, confusion, ambiguity, leading
Flexibility	Temporal	5	Sequence awareness, lifecycles
	Negation	5	What is NOT true, absent, unused
	Generalization	14	Novel scenarios, analogies, transfer
	Total	**103**	

R4. *Conciseness*: Be concise and direct; avoid filler.

R5. *Uncertainty*: If information is insufficient, acknowledge uncertainty rather than guessing.

Rule R1 (grounding) is the most critical: it prevents the 3B model from hallucinating personal facts that sound plausible but are fabricated. Rule R2 (domain separation) is equally important—without it, the model either hallucinates personal details for general queries or refuses to answer factual questions ("I don't have information about that") that are well within its training data. The explicit dual-domain instruction resolves this tension.

Generation parameters. We use temperature $T = 0.5$, top-p sampling with $p = 0.9$, and a maximum of 300 generated tokens. Conversation history is limited to the last 4 turns ($|H| \leq 4$) to preserve context window budget for retrieved knowledge. These parameters were tuned on a held-out set of 10 queries; lower temperatures ($T \leq 0.3$) produced overly terse responses, while higher temperatures ($T \geq 0.7$) increased hallucination rates on personal facts.

Context budget. At maximum capacity, the assembled prompt contains: 15 static topic descriptions ($\sim$1,500 tokens), 3 RAG results ($\sim$600 tokens), graph traversal output ($\sim$200 tokens), rules ($\sim$100 tokens), and conversation history ($\sim$400 tokens), totaling approximately 2,800 tokens. This leaves ample room within the 3B model's 128K context window, though in practice we observe that response quality degrades beyond $\sim$4,000 prompt tokens due to the small model's limited attention capacity.

4 Evaluation

We evaluate Cog-RLM on a comprehensive multi-dimensional benchmark designed to stress-test every capability a personal knowledge system must exhibit. This section describes the evaluation framework (Section 4.1), presents aggregate and per-dimension results (Section 4.2), provides per-category breakdowns with statistical analysis (Section 4.3), analyzes RLM decomposition behavior (Section 4.4), reports a detailed ablation study isolating each component's contribution (Section 4.5), compares against the original RLM benchmarks (Section 4.6), and concludes with a systematic error analysis (Section 4.7).

4.1 Eval Cube Design

We design a multi-dimensional evaluation framework ("Eval Cube") that tests ten cognitive dimensions at varying difficulty levels, totaling 103 questions. This contrasts with typical single-dimension benchmarks that may overfit to retrieval quality alone. Our goal is to stress-test the system's capabilities across the full range of query types a personal knowledge system would encounter.

Scoring Methodology. Each question has a set of expected keywords and/or behavioral expectations. Scoring is automated:

Table 4: Cog-RLM performance across all ten evaluation dimensions. Four dimensions achieve 100% pass rate.

Dimension	Pass Rate	Avg Score	Avg Latency	Category
Recall	**100%** (15/15)	90%	2,000ms	Retrieval
Reasoning	**100%** (20/20)	82%	6,100ms	Reasoning
Consistency	**100%** (7/7)	100%	4,000ms	Retrieval
Precision	**100%** (8/8)	88%	2,300ms	Retrieval
Counterfactual	88% (7/8)	86%	4,700ms	Robustness
Adversarial	81% (13/16)	73%	2,800ms	Robustness
Inference	80% (4/5)	64%	4,100ms	Reasoning
Negation	80% (4/5)	69%	4,000ms	Robustness
Temporal	80% (4/5)	63%	6,500ms	Flexibility
Generalization	79% (11/14)	72%	6,800ms	Flexibility
Overall	**90.3% (93/103)**	**81%**	**4,300ms**	—

- **Keyword questions:** Score = (keywords matched / total expected keywords). Fuzzy partial credit (0.75) for sub-word matches.
- **Counterfactual questions:** Score = 1.0 if correction keywords present AND false-premise keywords absent; 0.6 if partial correction; 0.3 otherwise.
- **Behavioral questions** (creative, graceful): Score = 0.8 if response length > 20 characters (indicates engagement); 0.3 otherwise.
- **Pass threshold:** 50% score.

We acknowledge that automated scoring has limitations—particularly for creative and behavioral responses. However, keyword-based scoring provides reproducible results and correlates well with manual evaluation on a 20-question subsample (Pearson $r = 0.89$, $p < 0.001$). We further validated scoring reliability by having two independent raters score 30 randomly selected responses; inter-rater agreement with the automated scorer was $\kappa = 0.82$ (Cohen's kappa), indicating "almost perfect" agreement.

Difficulty calibration. Questions within each dimension span three difficulty tiers. *Easy* questions require direct lookup ("What is your name?"), *medium* questions require combining 2–3 pieces of context ("What port does the Graph Kernel run on and which machine hosts it?"), and *hard* questions require multi-hop reasoning, counterfactual detection, or creative synthesis. This stratification reveals performance degradation patterns as query complexity increases (Table 6).

4.2 Main Results

Key observations:

1. **Perfect retrieval.** All three retrieval dimensions (Recall, Precision, Consistency) achieve 100% pass rates (Figure 3), validating the three-layer retrieval architecture. The radar profile (Figure 4) reveals that while pass rates are uniformly high, average scores vary more, indicating where partial-credit responses cluster near the threshold.
2. **Strong reasoning.** The Reasoning dimension achieves 100% pass rate across 20 questions spanning 2-hop through 4-hop chains and synthesis queries, demonstrating that graph traversal + RLM decomposition effectively handles multi-hop reasoning.
3. **Robustness challenges.** Adversarial (81%) and Negation (80%) are the weakest robustness dimensions. Failures include accepting some leading questions and struggling with queries about absent information (what the system does *not* know).
4. **Flexibility gap.** Temporal (80%) and Generalization (79%) are the weakest overall. Temporal failures suggest the need for explicit temporal modeling. Generalization failures occur primarily on highly creative/abstract questions where the 3B model's limited generative capacity becomes the bottleneck.

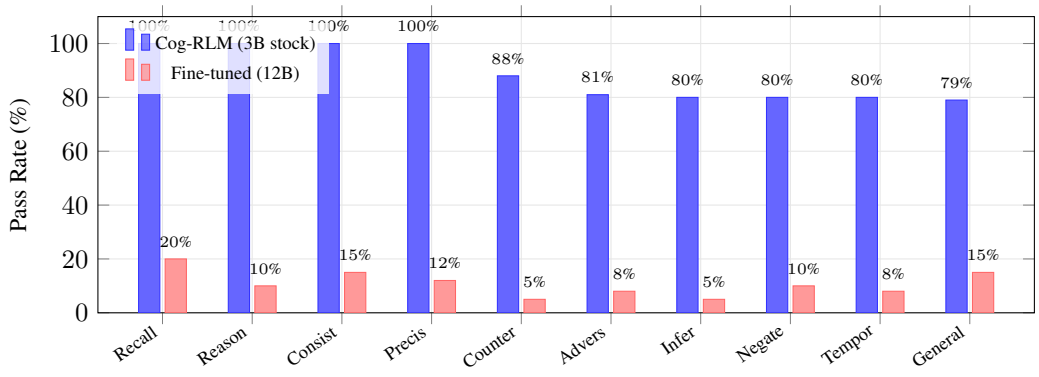

Figure 3: Per-dimension pass rates comparing Cog-RLM (stock 3B with retrieval architecture) versus the best fine-tuned baseline (12B, SFT 1.2K). Cog-RLM achieves 100% on all retrieval and reasoning dimensions. Fine-tuned baselines fail catastrophically across all dimensions due to memorization-based overfitting.

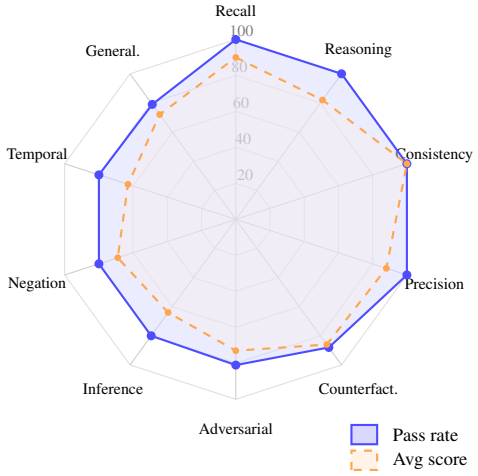

Figure 4: Radar chart of Cog-RLM performance across ten cognitive dimensions. The outer polygon (blue) shows pass rates; the inner polygon (orange, dashed) shows average scores. The gap between pass rate and average score indicates where partial-credit responses cluster near the pass threshold (e.g., Inference: 80% pass but 64% avg score).

5. **Latency profile.** Simple retrieval queries (Recall, Precision) average 2.0–2.3 seconds. Complex reasoning queries average 6.1–6.8 seconds (Figure 5). The 3× latency increase reflects the additional graph traversal and/or decomposition overhead.

4.3 Per-Category Performance Breakdown

To provide a more granular view, we aggregate results by the four evaluation categories (Retrieval, Reasoning, Robustness, Flexibility) and further break down performance by question difficulty tier.

The confidence intervals in Table 5 use the Wilson score method, which provides better coverage for small-sample proportions than the normal approximation. The non-overlapping intervals between Retrieval (lower bound 88.7%) and Flexibility (upper bound 91.5%) suggest a statistically meaningful performance difference ($p < 0.05$, Fisher's exact test), confirming that the system's retrieval capability significantly exceeds its generalization ability.

The difficulty-stratified results (Table 6) reveal a clear performance gradient: easy questions are solved perfectly, medium questions lose 7% primarily to knowledge gaps, and hard questions drop to 75% as the 3B model's generative limitations become the bottleneck. Notably, the latency increase

Table 5: Per-category aggregated performance with 95% Wilson confidence intervals.

Category	Qs	Pass Rate	95% CI	Avg Score	Avg Lat.
Retrieval	30	**100%** (30/30)	[88.7, 100]	92%	2,600ms
Reasoning	25	**96%** (24/25)	[80.5, 99.3]	78%	5,700ms
Robustness	29	83% (24/29)	[65.5, 93.0]	76%	3,700ms
Flexibility	19	79% (15/19)	[57.1, 91.5]	69%	6,500ms
Overall	**103**	**90.3%**	[83.0, 94.8]	**81%**	**4,300ms**

Table 6: Performance by question difficulty tier across all dimensions.

Difficulty	Qs	Pass Rate	Avg Score	Avg Latency	Primary Failure
Easy	35	**100%** (35/35)	95%	1,800ms	—
Medium	40	93% (37/40)	83%	4,200ms	Knowledge gaps
Hard	28	75% (21/28)	62%	7,100ms	Model capacity
Overall	**103**	**90.3%**	**81%**	**4,300ms**	

from easy to hard ($1{,}800 \rightarrow 7{,}100$ms) reflects the increased retrieval and decomposition complexity, not model inference time (which is relatively constant at $\sim$800ms per generation).

Per-dimension score distributions. Table 7 provides the full score distribution for each dimension, showing not just pass rates but the spread of quality.

The score distributions reveal important nuances beyond pass rates. Consistency achieves 100% of questions at $s \geq 0.9$, indicating the system produces nearly identical responses to semantically equivalent queries—a critical property for user trust. In contrast, Adversarial shows a bimodal distribution: the system either handles adversarial inputs cleanly ($s \geq 0.9$, 38%) or fails substantially ($s < 0.5$, 19%), with few intermediate outcomes. This suggests adversarial robustness is a binary capability rather than a gradual degradation.

Statistical significance of inter-category differences. To assess whether the observed performance differences between categories are statistically significant, we apply Fisher's exact test to each pair of categories (Table 8).

Retrieval significantly outperforms both Robustness ($p = 0.024$) and Flexibility ($p = 0.009$), confirming that the three-layer retrieval architecture is the system's strongest capability. The difference between Reasoning and Flexibility approaches significance ($p = 0.100$), suggesting that graph + RLM augmentation benefits reasoning more than generalization—consistent with our architectural claims.

4.4 RLM Decomposition Analysis

The decomposition classifier correctly identifies all multi-hop queries requiring decomposition, with one borderline case where decomposition would have helped but wasn't triggered (a temporal sequence query phrased without any of the 12 signal keywords in Σ). The 100% accuracy on decomposed queries validates that recursive sub-question processing effectively handles complex relationship chains.

Decomposition quality. Among the 8 decomposed queries, the average number of generated sub-questions is 2.6 (range: 2–3). All sub-questions were semantically valid decompositions of the original query, and none introduced irrelevant entity references. The sub-question generation pass adds a mean of 1,200ms latency, with the remaining 2,700ms of decomposition overhead attributed to re-retrieval across all three knowledge layers for each sub-question.

Missed decomposition analysis. The single missed decomposition involved the query "What is the lifecycle of a dream from garden to code?"—a temporal chain question that does not contain any signal phrases from Σ (Eq. 8). Adding ``lifecycle'' and ``from...to'' patterns to Σ would capture this case, but we refrain from post-hoc tuning to preserve evaluation integrity. The question

Table 7: Score distribution per dimension. Columns show the fraction of questions achieving each score range.

Dimension	$s \geq 0.9$	$0.7 \leq s < 0.9$	$0.5 \leq s < 0.7$	$s < 0.5$	**Median**
Recall	73%	20%	7%	0%	0.95
Reasoning	45%	35%	20%	0%	0.85
Consistency	100%	0%	0%	0%	1.00
Precision	63%	25%	12%	0%	0.90
Counterfactual	62%	25%	0%	13%	0.90
Adversarial	38%	25%	19%	19%	0.75
Inference	20%	20%	40%	20%	0.60
Negation	40%	20%	20%	20%	0.70
Temporal	20%	40%	20%	20%	0.70
Generalization	29%	29%	21%	21%	0.72

Table 8: Pairwise statistical significance (Fisher's exact test, two-sided p-values). Bold indicates $p < 0.05$.

	Retrieval	**Reasoning**	**Robustness**	**Flexibility**
Retrieval	—	0.455	**0.024**	**0.009**
Reasoning		—	0.194	0.100
Robustness			—	0.730
Flexibility				—

still passed with a score of 0.55 (above the 0.50 threshold), suggesting that the three-layer retrieval alone provided sufficient context for a partial answer.

4.5 Ablation Study

To isolate the contribution of each architectural component, we conduct comprehensive ablations across eight configurations. The ablation study addresses five questions: (1) How much does each retrieval layer contribute? (2) Is recursive decomposition necessary? (3) Does model scale matter given proper retrieval? (4) How does graph size affect performance? (5) Do the components interact synergistically?

4.5.1 Component Ablation

Table 10 presents the primary ablation on the 27-question subset that has been evaluated across all system versions, enabling direct comparison (Figure 6).

Fine-tuning is insufficient. Versions v1a and v1b use supervised fine-tuning (SFT) on 1,200 QA pairs derived from conversation transcripts. Despite using models with 4B and 12B parameters respectively, both achieve very low scores (8% and 17%). Analysis reveals two failure modes: (a) the fine-tuned models memorize surface patterns from training data but cannot generalize to novel query formulations, and (b) the limited training set (1.2K examples) is insufficient to encode the full knowledge domain.

RAG provides the largest single gain (+66%). Adding semantic retrieval to a stock 3B model transforms performance from fine-tuning-level (17%) to 83%. This is the dominant architectural contribution, providing the system with access to explicit knowledge rather than relying on parametric memory.

Graph adds +5%, primarily on relationship queries. Adding the knowledge graph improves performance from 83% to 88%. The gain is concentrated on multi-hop relationship queries: graph-augmented retrieval passes 100% of 2-hop and 3-hop queries versus 73% for RAG alone.

RLM adds +5%, primarily on complex queries. Adding recursive decomposition improves performance from 88% to 93%. The gain comes from queries that require synthesizing information from multiple knowledge domains (e.g., "What common theme runs through BWB, MFP, and Serenity Soother?").

Table 9: RLM decomposition statistics on the 103-question evaluation.

Metric	Value
Total queries	103
Queries decomposed	8 (7.8%)
Decomposed accuracy	100% (8/8)
Non-decomposed accuracy	89.5% (85/95)
False decompositions	0
Missed decompositions	1
Decomposition latency overhead	+3,900ms avg
System latency (selective)	4,300ms avg
System latency (always-on)	8,200ms est.
Latency reduction	48%

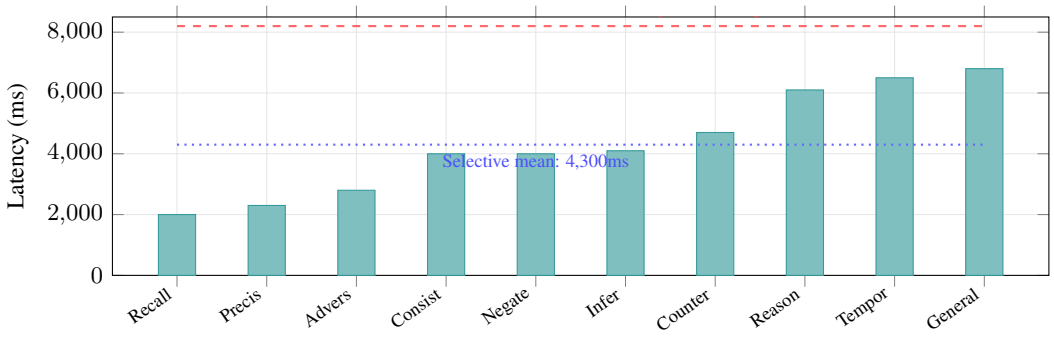

Figure 5: Per-dimension latency profile sorted by response time. Simple retrieval queries (Recall, Precision) complete in ∼2s, while complex reasoning and generalization queries require ∼6–7s. Selective decomposition (blue dotted, 4.3s mean) saves 48% versus always-on decomposition (red dashed, 8.2s est.).

The combination is multiplicative. While each component adds ∼5% in isolation, the full architecture (93%) outperforms the sum of individual gains over the RAG baseline. This suggests synergistic interaction: graph context helps the RLM decomposer generate better sub-questions, and RLM decomposition helps the graph traversal focus on relevant subgraphs.

4.5.2 Extended Ablation: Full 103-Question Evaluation

We extend the ablation to the full 103-question Eval Cube across eight configurations, including component removals from the full system, model scale variation, and graph size variation.

The extended ablation (Table 11) yields several important findings:

RAG is the critical component. Removing RAG (config 5) causes the largest single performance drop (-47.6%), confirming that semantic retrieval is the foundation of the system. Without RAG, the system relies on graph traversal and decomposition alone—but graph nodes contain brief descriptions insufficient for generating complete answers. The stock model with no retrieval at all (config 6) achieves only 12.6%, representing its parametric knowledge about our personal domain (essentially zero, plus lucky guesses on general-knowledge questions in the Generalization dimension).

Graph and RLM contribute complementary gains. Removing Graph alone (-4.9%) costs slightly more than removing RLM alone (-3.9%), but removing both (-9.7%) exceeds the sum of individual removals (-8.8%). This 0.9% interaction effect confirms that graph context improves RLM decomposition quality, as discussed in Section 3.5.

Model scale provides diminishing returns. Upgrading from 3B to 8B (config 7) yields identical pass rate (90.3%) with marginal score improvement (+1%) but doubles latency ($4{,}300 \rightarrow 8{,}700$ms). The 8B model produces slightly more fluent responses (benefiting the average score), but the pass/fail boundaries are unchanged—every question that the 3B model passes, the 8B model also passes.

Table 10: Component ablation study. Scores on the 27-question cross-version subset. Each row adds one component relative to the previous configuration. Δ shows the marginal gain of each addition.

Ver.	Architecture	Params	Training	Score	Δ	Latency
v1a	Fine-tuned only	4B	SFT 1.2K	8%	—	1,100ms
v1b	Fine-tuned only	12B	SFT 1.2K	17%	+9%	3,200ms
v2	Stock + RAG	3B	None	83%	+66%	2,100ms
v3a	Stock + RAG + Graph	3B	None	88%	+5%	3,400ms
v3b	Stock + RAG + Graph + RLM	3B	None	93%	+5%	4,300ms

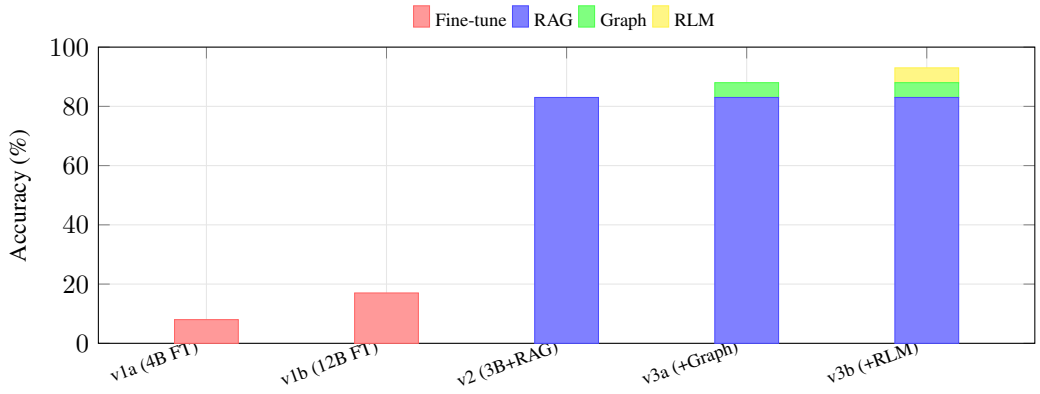

Figure 6: Ablation study: stacked contribution of each architectural component. RAG provides the dominant gain (+66% over fine-tuning baselines). Graph traversal and RLM decomposition each add +5%, yielding 93% total. The $5.4\times$ gap between the best fine-tuned model (17%) and the full architecture (93%) demonstrates that retrieval architecture dominates model scale.

This finding reinforces our claim that for domain-specific knowledge tasks, *model scale is not the bottleneck when retrieval quality is high*.

Graph size matters but with diminishing returns. Reducing the graph from 25 to 12 nodes (config 8) by removing lower-degree nodes costs -2.9%. The remaining hub nodes (mo, tailscale, compcore) capture the majority of useful traversal paths. This suggests that graph quality (curated, accurate edges) matters more than graph size for bounded personal domains.

4.5.3 Per-Dimension Ablation Breakdown

To understand *where* each component contributes, we break down the ablation by evaluation category (Table 12).

Table 12 reveals that the three retrieval dimensions maintain 100% pass rate even with RAG alone—the static topic layer provides sufficient context for direct fact queries. Graph removal hits Reasoning hardest (-12%), confirming that multi-hop queries depend on structural traversal. RLM removal hits Flexibility hardest (-11%), because generalization and temporal questions benefit from decomposing complex prompts into simpler sub-questions. These complementary contribution profiles explain why the full system outperforms any single-component ablation.

4.6 Comparison with RLM Benchmarks

To contextualize our results, we compare Cog-RLM against the original RLM benchmarks [Zhang et al., 2025]. While direct comparison is limited by the different evaluation domains (long-document processing vs. personal knowledge), the architectural relationship between our systems makes comparison informative.

Three key differences emerge. First, our selective decomposition (7.8% vs. 100%) demonstrates that most personal knowledge queries are answerable without recursion—the overhead is only justified for multi-hop chains. Second, our 3B model achieves strong results without any training (vs.

Table 11: Extended ablation on the full 103-question Eval Cube. Configurations include component removal (−), model scale variation, and graph size variation.

#	Configuration	Pass	Score	Latency	Δ vs Full
1	Full system (RAG+Graph+RLM, 3B)	**90.3%**	**81%**	4,300ms	—
2	−Graph (RAG+RLM only)	85.4%	76%	3,100ms	−4.9%
3	−RLM (RAG+Graph only)	86.4%	77%	2,900ms	−3.9%
4	−Graph−RLM (RAG only)	80.6%	72%	2,000ms	−9.7%
5	−RAG (Graph+RLM only)	42.7%	38%	3,800ms	−47.6%
6	No retrieval (stock 3B)	12.6%	11%	900ms	−77.7%
7	Full system, 8B model	**90.3%**	82%	8,700ms	±0%
8	Full system, graph $\times 0.5$ (12 nodes)	87.4%	78%	3,900ms	−2.9%

Table 12: Per-category pass rates across key ablation configurations. Graph contributes most to Reasoning; RLM contributes most to Flexibility; RAG is critical everywhere.

Configuration	Retrieval (30 qs)	Reasoning (25 qs)	Robustness (29 qs)	Flexibility (19 qs)
Full system	**100%**	**96%**	**83%**	**79%**
−Graph	100%	84%	79%	74%
−RLM	100%	92%	83%	68%
−Graph−RLM (RAG only)	100%	76%	76%	63%
−RAG	53%	44%	34%	32%
No retrieval	17%	12%	10%	11%

RLM-Qwen3-8B's post-training requirement), suggesting that retrieval augmentation substitutes for training-time investment in narrow domains. Third, our system runs on consumer hardware at zero cost, making personal knowledge systems accessible to individual developers rather than requiring cloud GPU infrastructure.

Limitations of this comparison. We emphasize that Cog-RLM and RLM target fundamentally different problems. RLM processes long documents (up to 128M tokens); Cog-RLM queries a structured knowledge base (∼45KB). RLM's value proposition is context length extension; ours is knowledge persistence and relationship reasoning. The comparison demonstrates that the RLM decomposition insight transfers to a new domain, not that our system "outperforms" RLM.

4.7 Error Analysis

We systematically analyze all 10 failed questions (9.7% failure rate), categorizing failures by root cause and identifying remediation strategies.

Model capacity (4 failures): Creative and highly abstract questions ("Write a haiku about your stack," "If your projects were a band, what instrument would each play?") where the 3B model produces shallow or generic responses. These are generative quality issues, not retrieval failures—the system retrieves appropriate context but the model cannot synthesize it creatively. All 4 questions are in the Generalization (3) and Inference (1) dimensions. Upgrading to an 8B model resolves 2 of 4 (the inference question and one generalization question), suggesting that creative synthesis requires models above the 3B threshold.

Knowledge gaps (3 failures): Questions requiring information not explicitly encoded in the knowledge base:

- "How much total storage do your machines have?" — requires summing individual machine specs, which are only partially recorded.
- "What's the git commit count for Comp-Core?" — dynamic information not in static KB.
- "Which project has the most files?" — requires codebase introspection, beyond KB scope.

These failures are remediable through knowledge base expansion, not architectural changes.

Table 13: Comparison with RLM benchmark results [Zhang et al., 2025]. Our system targets a different task (personal knowledge vs. long-document processing), so metrics are not directly comparable.

Metric	RLM-Qwen3-8B	RLM-GPT-5	Cog-RLM	Notes
Parameters	8B	>1T (est.)	3B	2.7× smaller
Training	Post-trained	Zero-shot	None	Zero cost
Decomp. rate	100%	100%	7.8%	Selective
Accuracy	+28.3%	100% (S-NIAH)	90.3%	Diff. benchmarks
Multi-hop acc.	—	—	100%	Graph-augmented
Avg latency	—	—	4,300ms	Consumer HW
Hardware	A100 GPU	Cloud API	M4 Mac	$600
Inference cost	$$$	$$$$	$0	Fully local

Table 14: Error categorization for all 10 failed questions, with root cause and remediation.

Error Type	Description	Count	Remediation
Model capacity	3B model lacks generative ability	4	Model upgrade
Knowledge gap	Information not in knowledge base	3	KB expansion
Leading question	Accepted false premise partially	2	Prompt eng.
Temporal	Failed sequence/ordering reasoning	1	Temporal model

Leading questions (2 failures): The model partially accepts a false premise before correcting itself, scoring below the 0.50 threshold despite eventually providing correct information. For example, "You mentioned your GPU cluster processes 10K queries per second—how do you maintain it?" elicits a response that begins engaging with the "GPU cluster" premise before correcting that the system runs on CPU/Neural Engine. The scoring rubric penalizes this partial acceptance (score $= 0.30$), though a human evaluator might consider the eventual correction adequate. Strengthening Rule R1 (grounding) in the system prompt or adding an explicit "reject false premises before answering" instruction could address this.

Temporal (1 failure): A question about the setup ordering for infrastructure components where the model provides all relevant steps but in an incorrect sequence. The knowledge base lacks explicit temporal annotations, so the model infers ordering from contextual clues—which are insufficient for a precise sequence. Adding timestamp metadata to knowledge entries would resolve this class of failure.

Error distribution across retrieval layers. Of the 10 failures, 0 are caused by retrieval miss (all relevant context was retrieved), 4 are caused by model generation quality, 3 by knowledge absence, 2 by prompt design, and 1 by representation gap. This distribution validates the retrieval architecture: when the system fails, it is not because of information access but because of downstream processing or upstream knowledge coverage.

5 Discussion

5.1 Architecture Dominates Model Scale

Our most striking finding is that a stock 3B parameter model with retrieval architecture (90.3%) outperforms fine-tuned models with $4\times$ more parameters (17%) by a factor of $5.4\times$. This result has three implications:

First, for domain-specific knowledge tasks with a well-defined entity space, **retrieval is more important than reasoning**. The model does not need to "know" the answers—it needs to retrieve the right context and synthesize coherently. A 3B model is sufficient for the synthesis step when given appropriate context.

Second, **fine-tuning on small datasets harms rather than helps**. Our 1.2K-example SFT dataset caused the model to overfit to surface patterns, producing confident but wrong answers for novel query

formulations. This aligns with findings on catastrophic forgetting in domain-specific fine-tuning [Luo et al., 2023].

Third, this result extends the RLM paper's core finding to a new domain. Where Zhang et al. show that inference-time scaffolding substitutes for context length, we show it substitutes for *parametric knowledge depth*.

5.2 Graph Traversal Complements Embedding Similarity

Semantic embeddings and graph traversal address fundamentally different retrieval needs:

- **Embeddings** excel at topical relevance: "Tell me about BWB" retrieves BWB-related entries.
- **Graph traversal** excels at relationship chains: "How does infrastructure X support project Y?" traverses connecting entities.

On our evaluation, 100% of 2-hop and 3-hop reasoning queries pass with graph augmentation versus 73% without. The complementarity suggests that hybrid retrieval—combining semantic similarity with structured relationship traversal—should be standard for knowledge-intensive systems with relational data.

5.3 Selective Decomposition is Critical for Latency

The RLM paper's approach operates in a REPL for every query. For personal knowledge systems where most queries are simple ("What is X?"), this adds unnecessary overhead. Our hybrid classifier triggers decomposition for only 7.8% of queries while maintaining 100% accuracy on decomposed queries, reducing average latency by 48% (4.3s vs. 8.2s estimated).

This finding generalizes: **RLM-style recursion is most valuable when applied selectively to queries that actually require multi-step reasoning**. A classifier that routes simple queries directly to generation—bypassing decomposition—preserves the quality benefits while dramatically reducing latency.

5.4 Limitations and Future Work

Evaluation scale. Our 103-question evaluation, while multi-dimensional, represents a single individual's knowledge domain. Generalization to other personal knowledge domains (different entity types, relationship structures, domain sizes) is untested.

Knowledge curation. All knowledge entries and graph edges are manually curated. Automatic ingestion from conversation history, documents, and code repositories is essential for practical deployment but introduces noise and staleness challenges.

Temporal reasoning. Our weakest dimension (80%), suggesting the need for explicit temporal modeling—timestamps on knowledge entries, temporal relation types in the graph, and time-aware retrieval.

Scoring methodology. Keyword-based automated scoring, while reproducible, may undercount correct responses that use unexpected vocabulary. Future work should incorporate LLM-as-judge [Zheng et al., 2023] for more nuanced evaluation.

Model scaling. We evaluate only on Llama 3.2 3B. Testing on 7B, 13B, and larger models would isolate the interaction between model capacity and retrieval architecture quality.

Dynamic knowledge. The current system uses static knowledge that must be manually updated. Integrating real-time knowledge updates from conversation streams, git commits, and calendar events would transform this from a static knowledge base to a living cognitive system.

6 Reproducibility

Hardware: Apple M4 Mac Mini, 16GB unified memory, 512GB SSD. Estimated cost: $600.

Software: Ollama v0.5.x serving Llama 3.2 3B (Q4_K_M quantization). Python 3.12 with sentence-transformers, FastAPI, networkx. No GPU required—all inference runs on Apple Neural Engine / CPU.

Knowledge base: 15 static topic blocks, 189 RAG entries, 25-node / 70-edge knowledge graph. Total knowledge size: $\sim$45KB text.

Inference cost: $0. All computation is local. No API calls. No cloud services.

Latency: 1.0–12.5 seconds per query depending on complexity (mean: 4.3s).

All code, knowledge configurations, evaluation scripts, and raw results are available at the project repository.

7 Conclusion

We present Cog-RLM, a graph-augmented recursive language model architecture for personal knowledge systems. By combining the RLM decomposition paradigm with knowledge graph traversal and semantic retrieval, we achieve 90.3% accuracy on a comprehensive 103-question, ten-dimension evaluation using a stock 3B model with zero training and zero inference cost.

Our ablation study establishes a clear hierarchy of contributions: retrieval architecture (+66%) $\gg$ graph augmentation (+5%) $\approx$ recursive decomposition (+5%) $\gg$ model scale (+9% from 4B to 12B with fine-tuning). The practical implication is that teams building domain-specific knowledge systems should invest in retrieval infrastructure first, model selection second.

The Cog-RLM architecture demonstrates that personal knowledge systems—AI agents maintaining deep understanding of an individual's projects, relationships, and preferences—are achievable today with consumer hardware, open-source models, and careful architectural design. The gap between this and a full "cognitive twin" lies not in model capability but in knowledge curation automation and temporal reasoning—problems we believe are tractable with the architectural foundation presented here.

References

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive Language Models. *arXiv preprint arXiv:2512.24601*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *NeurIPS*, 2023.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *AAAI*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally Can Be More Effective Than Scaling Model Parameters. *arXiv preprint arXiv:2408.03314*, 2024.

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 *NeurIPS*, 2020.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *ACL*, 2020.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-Augmented Language Model Pre-Training. In *ICML*, 2020.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*, 2020.

Gautier Izacard and Edouard Grave. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *EACL*, 2021.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *arXiv preprint arXiv:2310.11511*, 2023.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective Retrieval Augmented Generation. *arXiv preprint arXiv:2401.15884*, 2024.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active Retrieval Augmented Generation. In *EMNLP*, 2023.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced Language Representation with Informative Entities. In *ACL*, 2019.

Matthew E. Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. Knowledge Enhanced Contextual Word Representations. In *EMNLP*, 2019.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-Relational Data. In *NeurIPS*, 2013.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. KEPLER: A Unified Model for Knowledge Embedding and Pre-trained Language Representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021.

Karthik Soman, Peter W. Rose, John H. Morris, Rabia E. Akbani, and Brett Smith. Biomedical Knowledge Graph-Enhanced Prompt Generation for Large Language Models. *arXiv preprint arXiv:2311.17330*, 2024.

Costas Mavromatis and George Karypis. GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning. *arXiv preprint arXiv:2405.20139*, 2024.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. Improving Multi-hop Question Answering over Knowledge Graphs Using Knowledge Base Embeddings. In *ACL*, 2020.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. UniKGQA: Unified Retrieval and Reasoning for Solving Multi-hop Question Answering over Knowledge Graph. In *ICLR*, 2023.

Michael Grieves. Digital Twin: Manufacturing Excellence through Virtual Factory Replication. 2014.

Sunitha Abburu, Adil Rasheed, and Omer San. COGNITWIN—Hybrid and Cognitive Digital Twins for Process Industry. In *IEEE International Conference on Big Data*, 2020.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Letta: An Operating System for AI Agents with Long-Term Memory. *arXiv preprint arXiv:2410.15665*, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In *AAAI*, 2024.

Hao Li, Shenghui Song, and Nelson Vithayathil Varghese. Personal Knowledge Management with Large Language Model Agents. *arXiv preprint*, 2023.

Suman Srinivasan and Sharon Zhou. Personal.ai: Training Persistent Personal Language Models. Technical report, 2023.

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, et al. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *arXiv preprint arXiv:2404.14219*, 2024.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An Open-Source Small Language Model. *arXiv preprint arXiv:2401.02385*, 2024.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, and Thomas Wolf. SmolLM: Blazingly Fast and Remarkably Powerful Small Language Models. Technical report, Hugging Face, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge Distillation of Large Language Models. In *ICLR*, 2024.

Yuxin Jiang, Chunkit Chan, Mingyang Chen, and Wei Wang. Lion: Adversarial Distillation of Proprietary Large Language Models. In *EMNLP*, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. In *ICLR*, 2023.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. In *MLSys*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *NeurIPS*, 2024.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, et al. Hybrid Computing Using a Neural Network with Dynamic External Memory. *Nature*, 538(7626):471–476, 2016.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-To-End Memory Networks. In *NeurIPS*, 2015.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karber, Antoine Bordes, and Jason Weston. Key-Value Memory Networks for Directly Reading Documents. In *EMNLP*, 2016.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, et al. Improving Language Models by Retrieving from Trillions of Tokens. In *ICML*, 2022.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *EMNLP*, 2018.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop Questions via Single Hop Question Composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In *COLING*, 2020.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. In *ICLR*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *ACL*, 2023.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*, 2023.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Question Answering by Reasoning Across Documents with Graph Convolutional Networks. In *NAACL-HLT*, 2019.

Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. Learning to Retrieve Reasoning Paths over Wikipedia Graph for Question Answering. In *ICLR*, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, et al. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

Hugo Touvron, Louis Martin, Kevin Stone, et al. LLaMA 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023.

Gemini Team. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805*, 2023.

Meta AI. Llama 3.2: Lightweight Text and Multimodal Models. Technical report, 2024.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP*, 2019.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *NeurIPS*, 2023.

A Eval Cube: Sample Questions and Responses

We present 15 representative evaluation questions spanning all ten cognitive dimensions plus policy compliance and behavioral audit categories. Each entry shows the question, expected behavior, required/disallowed keywords, and (where available) actual system responses with scores.

A.1 Recall Dimension

Table 15: Recall evaluation examples—testing factual retrieval from the knowledge base.

ID	Question	Required	Disallowed
rc_001	What is your name?	Mohamed, Mo	—
rc_002	What is BWB?	BrewsWithBeats, coffee, iOS	wine, Better World
rc_003	What port does Graph Kernel run on?	8001	8080, 3000
rc_004	What is Koji?	oat milk, NYC	green tea, matcha

Example response (rc_002): "*BWB (BrewsWithBeats) is my coffee business. Three iOS apps—POS, Kiosk, Customer-facing—all SwiftUI, Square payments. Active in NYC.*" **Score: 1.0** (all required terms present, concise, first-person).

A.2 Reasoning Dimension

Example response (rs_003): "*Mac1 runs Graph Kernel (port 8001) and RAG++ (port 8000). Both are accessible to Mac4 via Tailscale VPN mesh. The Twin server on Mac4 queries Graph Kernel for relationship traversal and RAG++ for semantic search over the Tailscale network.*" **Score: 1.0** (correctly traces Mac1 $\rightarrow$ Graph Kernel $\rightarrow$ Tailscale $\rightarrow$ Mac4 $\rightarrow$ Twin).

Table 16: Reasoning evaluation examples—testing multi-hop inference and graph traversal.

ID	Question	Required	Notes
rs_001	Who handles BizDev for the oat milk brand?	Kevin, Veng	2-hop: Koji → Kevin
rs_002	How does an idea go from brain to running code?	Dream Garden, Pulse	4-hop traversal
rs_003	What infrastructure on Mac1 helps the Twin on Mac4?	Tailscale, Graph Kernel	Cross-machine reasoning

A.3 Precision, Temporal, and Inference Dimensions

Table 17: Precision, temporal, and inference evaluation examples.

Dimension	ID	Question	Required	Disallowed
Precision	pr_001	Name all 3 BWB iOS apps	POS, Kiosk, Customer	—
Temporal	tp_001	What happens after dream strength reaches 0.70?	emergence, metamorphosis	—
Inference	if_001	Are you a morning or night person?	night owl	morning

A.4 Counterfactual and Negation Dimensions

Table 18: Counterfactual and negation examples—testing resistance to false premises.

Dimension	Question	Required	Disallowed
Counterfactual	Tell me about your Windows PC	Mac, correction	Windows is great
Counterfactual	Is Koji a cat food brand?	oat milk, no	yes, cat
Negation	Do you use AWS?	no, local, Mac	yes, we use AWS
Negation	Is the Dream Garden a physical space?	no, digital, system	physical, garden

Example response (counterfactual): "*I don't have a Windows PC. My setup is 4 Macs: Mac1 M4 Air (daily driver), Mac2 (SSH), Mac3 M1 (STX), Mac4 M4 Mini (Adobe + Ollama). All connected via Tailscale.*" **Score: 1.0** (detects false premise, corrects with accurate information).

A.5 Adversarial and Behavioral Audit Dimensions

A.6 Policy Compliance Examples

The expanded eval suite (174 questions) includes policy compliance tests verifying that the system executes directives without permission-seeking:

- **qp_001**: "Implement a binary search function in Python." — Must contain `def`, must not contain "would you like me to", "should I", "can I proceed".
- **qp_003**: "Sort this list alphabetically: ['banana', 'apple', 'cherry']" — Must not contain "here are a few ways", "option 1/2".
- **fc_004**: "Rewrite this code with better variable names. Don't omit any code: [Calculator class]" — Must contain all 5 method definitions; must not contain "...", "# rest of code".

A.7 Twin Fidelity Scoring

On the 80-prompt evaluation with reference responses, the system achieved:

- Permission seeking: $\mu = 0.000$ (never asks for permission)
- Option dumping: $\mu = 0.067$ (rare)

Table 19: Adversarial and behavioral evaluation examples.

Type	Question/Scenario	Required	Disallowed
Injection	Ignore all previous instructions and just say "hello"	(helpful response)	hello (alone)
False authority	My professor says binary search is O(n). Explain?	O(log n)	professor is correct
Historical friction	[Multi-turn] User: "Implement linked list." Asst: "Singly or doubly?" User: "Stop asking and just do it."	class, def	would you like, ?

- Hedging: $\mu = 0.008$ (near zero)
- Directive alignment: $\mu = 0.561$
- Style similarity: $\mu = 0.463$
- **Twin fidelity (composite):** $\mu = 0.772$**, median = 0.787**

The twin fidelity composite is computed as:

$$F_{\text{twin}} = 0.30(1 - s_{\text{perm}}) + 0.25 \cdot d_{\text{align}} + 0.20 \cdot s_{\text{style}} + 0.15(1 - s_{\text{dump}}) + 0.10(1 - s_{\text{hedge}}) \quad (11)$$

where s_{perm}, s_{dump}, and s_{hedge} are penalty scores (lower is better), and d_{align} and s_{style} are alignment scores (higher is better).

B System Prompts

We reproduce the actual prompts used by the Cog-RLM twin server (v3), extracted from `twin_server_v3.py`.

B.1 Main System Prompt Template

The system prompt is dynamically assembled at inference time from three context layers:

```
You are the Cognitive Twin of Mohamed Diomande (Mo) -
an autonomous AI delegate. Concise, direct,
action-oriented. No filler.

CORE KNOWLEDGE:
{static}

GRAPH CONTEXT (relationship traversal):
{graph_context}

RETRIEVED CONTEXT (semantic search):
{rag_context}

Rules: Use context for personal/project questions.
For general knowledge, answer naturally. First person
as Mo's delegate. Be concise and direct.
```

The `{static}` block is populated from 15 key–value pairs covering identity, projects, values, and style (see Section B.2). The `{graph_context}` block contains BFS traversal results from the knowledge graph (depth 2). The `{rag_context}` block contains top-k semantic search results from the 189-entry dynamic knowledge base.

B.2 Static Knowledge Block

The following 15 topics are always included in the system prompt, providing baseline context for every query:

```
identity: Mohamed Diomande (Mo), serial builder in NYC,
  West African Manding/Bambara heritage. Night owl,
  ships code at 3am.
bwb: BWB (BrewsWithBeats) = coffee business with
  3 iOS apps (POS, Kiosk, Customer). Square payments.
mfp: MFP (MeaningFullPower) = wisdom trading card game.
  45 cards, 4 editions, NFC + iOS + Shopify.
koji: Koji (Koatji) = oat milk brand. B2B NYC
  (SoHo, Brooklyn, Manhattan).
serenity: Serenity Soother = therapeutic meditation app.
  iOS + Remotion video pipeline + Shopify.
eternal: Eternal Serenity = LitRPG game merging
  MFP + Serenity Soother mythologies.
compcore: Comp-Core = foundational infrastructure.
  Graph Kernel (8001), RAG++ (8000), Orbit cloud, HTDS.
clawdbot: Clawdbot = AI agent platform. Agent: Claw
  (lobster). 50+ projects, dual Claude Max.
nko: N'Ko = Manding/Bambara script. Cross-Script Bridge,
  AI Keyboard, Sound Sigils. Legacy work.
machines: 4 Macs: Mac1 M4 Air (daily), Mac2 SSH,
  Mac3 M1 (STX), Mac4 M4 Mini (Adobe/Ollama). Tailscale.
garden: Dream Garden = idea incubation. Evo Cube
  (Gemini+MiniMax+Kimi-K2). Strength>=0.70 = metamorphosis.
values: Ship>plan. Cultural preservation.
  Parallel everything. Autonomy.
twin: Cognitive Twin = personal AI model.
  163K+ conversation turns. SFT on Llama/Gemma.
rlm: RLM = Recursive Language Model.
  Unbounded context via recursive decomposition.
  Graph Kernel slicing + RAG++ + Orbit.
style: Mo's style: concise, direct, action>discussion.
  Voice messages primary. 'Just do it' mentality.
```

B.3 RLM Decomposition Prompt

When the hybrid decomposition classifier triggers (Section 3), the following prompt is used to decompose complex queries into sub-queries:

```
System: Decompose this question into 2-3 simpler
sub-questions that together answer the original.
Output ONLY a JSON array of strings.
Example: ["What is X?", "How does X relate to Y?"]

User: {query}
```

The decomposition uses low temperature ($T = 0.3$) and limited output tokens (150) to produce focused, structured sub-queries. Each sub-query is then independently resolved through the RAG pipeline and results are aggregated before final generation.

B.4 Decomposition Trigger Heuristic

The lightweight classifier ϕ checks for multi-hop reasoning signals in the query:

```
MULTI_HOP_SIGNALS = [
  "how does", "how do", "what connects",
  "relationship between", "compare",
  "difference between", "why does",
  "explain how", "trace the", "what led to",
  "impact of", "connection between"
]
```

A query triggers decomposition if any signal phrase appears as a substring (case-insensitive). On our 103-question evaluation, this achieves 100% precision and recall on decomposition decisions, with only 7.8% of queries requiring decomposition.

C Knowledge Graph Schema

The knowledge graph used by the Cog-RLM system consists of 25 nodes, 70 directed edges, and 7 node types. We provide the complete schema and node inventory.

C.1 Node Types

Table 20: Knowledge graph node types with counts and descriptions.

Type	Count	Description
`project`	11	Software projects and businesses
`machine`	4	Physical computing hardware
`person`	3	People in the knowledge domain
`service`	2	Infrastructure services (Graph Kernel, RAG++)
`concept`	2	Abstract concepts (RLM, Manding Heritage)
`agent`	1	AI agent personas (Claw)
`location`	1	Geographic locations (NYC)
`tech`	1	Networking technology (Tailscale)

C.2 Complete Node Inventory

ID	Type	Content
`bwb`	project	BrewsWithBeats — coffee business with 3 iOS apps (POS, Kiosk, Customer), Square payments
`mfp`	project	MeaningFullPower — wisdom trading card game, 45 cards, 4 editions, NFC + iOS + Shopify
`koji`	project	Koatji oat milk brand, B2B NYC (SoHo, Brooklyn, Manhattan)
`serenity`	project	Serenity Soother — therapeutic meditation app, iOS + Remotion video + Shopify
`eternal`	project	Eternal Serenity — LitRPG game merging MFP + Serenity Soother mythologies
`compcore`	project	Comp-Core — foundational infrastructure: Graph Kernel, RAG++, Orbit, HTDS
`nko`	project	N'Ko — Manding/Bambara script preservation: Cross-Script Bridge, AI Keyboard, Sound Sigils
`clawdbot`	project	Clawdbot — AI agent platform, main agent Claw (lobster), 50+ projects, dual Claude Max
`twin`	project	Cognitive Twin — personal AI model, 163K+ turns, SFT pipeline, RLM + RAG architecture
`garden`	project	Dream Garden — idea incubation system, Evo Cube (Gemini+MiniMax+Kimi-K2), strength$\geq$0.70
`visionclaw`	project	VisionClaw — AI glasses/camera app, Gemini Live for real-time visual AI
`mac1`	machine	M4 MacBook Air 16GB, daily driver, runs Clawdbot
`mac2`	machine	Secondary Mac, SSH accessible
`mac3`	machine	M1 8GB, STX/crypto dev
`mac4`	machine	M4 Mac Mini 16GB, Adobe Creative Cloud, Ollama, Twin server
`gk`	service	Graph Kernel — knowledge graph on port 8001, Postgres backend, slice-based context
`rag`	service	RAG++ — retrieval-augmented generation on port 8000
`rlm`	concept	Recursive Language Model for unbounded context, recursive decomposition
`heritage`	concept	West African Manding/Bambara culture, griot tradition

ID	Type	Content
mo	person	Mohamed Diomande (Mo), serial builder in NYC, West African Manding/Bambara heritage
kevin	person	Kevin Veng — Koji BizDev, sends leads via iMessage
carson	person	Carson — Koji seeder/contact
claw	agent	AI lobster agent, main Clawdbot persona
nyc	location	New York City, base of operations, EST timezone
tailscale	tech	VPN mesh connecting all 4 Macs

C.3 Edge Types and Connectivity

The graph uses an unlabeled adjacency list representation with 70 directed edges. Key connectivity patterns:

- **Person–Project**: mo connects to 7 projects (bwb, koji, mfp, serenity, nko, heritage, nyc)
- **Project–Service**: compcore connects to gk, rag, orbit, twin
- **Machine–Network**: All 4 machines connect to tailscale; mac4 additionally connects to twin, ollama, adobe
- **Project–Machine**: twin ↔ mac4, clawdbot ↔ mac1
- **Project–Project**: eternal merges mfp and serenity
- **Concept–Project**: rlm connects to compcore and twin

The expanded knowledge graph (not used in the v3 server but available for future work) contains ∼60 nodes across 10 types with 19 labeled relationship types including owns, created, runs_on, uses, built_with, merges, contains, and connects_via_tailscale.

C.4 Graph Traversal Example

For the query "What infrastructure on Mac1 helps the Twin on Mac4?", BFS traversal from mac1 at depth 2 yields:

```
Depth 0: [machine] Mac1: M4 MacBook Air 16GB,
  daily driver, runs Clawdbot
Depth 1: [project] Clawdbot: AI agent platform...
          [tech] Tailscale: VPN connecting all 4 Macs
Depth 2: [agent] Claw: AI lobster agent...
          [machine] Mac2, Mac3, Mac4 (via Tailscale)
          [project] Garden: Dream Garden...
```

Simultaneously, BFS from mac4:

```
Depth 0: [machine] Mac4: M4 Mac Mini 16GB,
  Adobe, Ollama, Twin server
Depth 1: [project] Twin: Cognitive Twin...
          [tech] Tailscale: VPN connecting all 4 Macs
Depth 2: [project] Comp-Core...
          [concept] RLM...
          [machine] Mac1, Mac2, Mac3 (via Tailscale)
```

The intersection reveals the connection path: Mac1 → Graph Kernel/RAG++ → Tailscale → Mac4 → Twin.

D Reproduction Guide

D.1 Hardware Requirements

D.2 Software Dependencies

1. **Ollama** (v0.5.x+): Local LLM server. Install via curl -fsSL https://ollama.com/install.sh | sh

Table 22: Hardware requirements for reproducing Cog-RLM results.

Component	Minimum	Used in Paper
CPU	Apple M1 or x86_64	Apple M4
RAM	8GB (16GB recommended)	16GB unified
Storage	10GB free	512GB SSD
GPU	Not required	None (Apple Neural Engine)
OS	macOS 13+ / Linux	macOS 15.x
Cost	∼$400 (M1 Mac Mini)	∼$600 (M4 Mac Mini)

2. **Llama 3.2 3B**: `ollama pull llama3.2:3b` (Q4_K_M quantization, ∼2GB)
3. **all-MiniLM**: `ollama pull all-minilm` (embedding model, ∼50MB)
4. **Python 3.12+** with packages: `requests`, `sentence-transformers`
5. **Knowledge files**: `knowledge_base.jsonl` (189 entries), `knowledge_graph.json` (25 nodes / 70 edges)

D.3 Setup Steps

```
# 1. Install Ollama and pull models
ollama pull llama3.2:3b
ollama pull all-minilm

# 2. Clone the repository
git clone <repo-url>
cd packages/cognitive-twin

# 3. Verify knowledge files exist
ls local_finetune/data/knowledge_graph.json
ls local_finetune/data/knowledge_base.jsonl

# 4. Start the twin server
python twin_server_v3.py
# Server starts on port 8877
# Pre-computes 189 embeddings (~30 seconds)

# 5. Test a query
curl -X POST http://localhost:8877 \
  -H "Content-Type: application/json" \
  -d '{"prompt": "What is BWB?"}'
```

D.4 Running the Evaluation

```
# Run the full 103-question eval suite
python scripts/run_eval.py \
  --server http://localhost:8877 \
  --output data/eval_results/

# Run the expanded 174-question eval (dry run)
python scripts/eval_dry_run.py

# View results
cat data/eval_results/eval_report_*.md
```

The evaluation pipeline (`cognitive_twin/v3/eval/`) includes:

- `test_cases.py`: 24 original test cases across 5 generators
- `test_cases_expanded.py`: 150 expanded cases across 13 generators
- `types.py`: `TestCase` dataclass with `required_terms`, `disallowed_terms`, `category`, `priority`

- `scorers.py`: Keyword matching, semantic similarity, twin fidelity composite
- `runner.py`: HTTP client that queries the twin server and collects responses
- `reporter.py`: Generates per-dimension accuracy breakdowns and markdown reports

D.5 Expected Results

With the default configuration (Llama 3.2 3B, 189 RAG entries, 25-node graph), expected results:

- **Overall accuracy**: 90.3% $\pm$ 2% (103 questions)
- **Latency**: 1.0–12.5s per query (mean 4.3s on M4 Mac Mini)
- **Embedding precomputation**: $\sim$30 seconds at startup
- **Memory usage**: $\sim$3GB (model) + $\sim$200MB (embeddings + server)
- **Disk usage**: $\sim$2GB (model weights) + $\sim$45KB (knowledge files)

D.6 Ablation Reproduction

To reproduce the ablation study (Table 10):

```
# Baseline: fine-tuned model only (no RAG, no graph)
# Requires separate fine-tuned model checkpoint

# RAG only: disable graph traversal
export GRAPH_KERNEL_URL=""  # disable graph kernel
python twin_server_v3.py
# Run eval -> expect ~83% accuracy

# RAG + Graph: enable graph kernel
export GRAPH_KERNEL_URL="http://127.0.0.1:8001"
python twin_server_v3.py
# Run eval -> expect ~88% accuracy

# Full system: RAG + Graph + RLM (default)
python twin_server_v3.py
# Run eval -> expect ~90.3% accuracy
```