Noēsis: Bidirectional Graph-RAG with Adaptive Parallelism and Cross-Knowledge-Base Semantic Discovery*

Nicola Cogotti
Alpha Cogs
nicola.cogotti@alphacogs.co.uk

August 2026

Patent pending: Application No. 102026000023146; Application No. IT202500035167.

Abstract

Retrieval-Augmented Generation over knowledge graphs (Graph-RAG) has emerged as the dominant paradigm for grounding large language models in domain-specific corpora. However, existing systems suffer from three fundamental limitations: (1) static chunking fragments long documents, losing cross-section semantic connections; (2) ingestion pipelines do not scale adaptively, causing out-of-memory failures or underutilized hardware; and (3) multi-domain deployments require either a monolithic knowledge base that dilutes retrieval precision or manual user routing.

We present **Noēsis**, a decoupled Graph-RAG architecture that addresses all three limitations through four novel algorithms: (a) *Bidirectional Graph Traversal* with a Graph-Feedback Context Resolver that simulates human sequential reading with degrading memory, producing significantly denser graphs versus single-pass baselines; (b) an *AIMD Concurrency Controller* adapted from TCP congestion control to RAG pipeline orchestration, achieving 23× speedup with zero OOM risk; (c) *Moēsis*, a domain-aware selective quantization pipeline for Mixture-of-Experts models that achieves 6.3× prompt processing speedup on 12 GB consumer GPUs and maintains crash-free stability even under extreme memory constraints; and (d) *Mesh*, a cross-KB semantic routing system with runtime structural discovery and adaptive Natural Break thresholds that enables small on-premises models to perform multi-hop cross-domain reasoning—discovering connections not explicitly present in any single document—with quality typically associated with significantly larger cloud-based systems.

We report measured results on real-world workloads: 1 min 6 s for 13.4 MB corpus ingestion (vs. 25 min sequential), prompt processing speedup of 6.3× on consumer GPUs, and cross-KB routing latency under 2 ms.

1 Introduction

Knowledge graphs provide structured representations of domain expertise that can ground large language model (LLM) responses in verifiable facts. Graph-based Retrieval-Augmented Generation (Graph-RAG) combines the semantic richness of knowledge graphs with the generative capabilities of LLMs, enabling multi-hop reasoning over complex corpora [1, 2].

Despite recent progress, production deployment of Graph-RAG systems faces three critical bottlenecks:

Semantic Fragmentation. All existing Graph-RAG systems—including Microsoft GraphRAG [1], LightRAG [2], LazyGraphRAG [5], and HippoRAG [3]—extract entities and relations from documents using *static chunking*: the document is split into fixed-size blocks, each processed independently. Limited or no shared context propagates between chunks during extraction. Consequently, concepts spanning chunk boundaries are lost or duplicated, and the resulting graph lacks the density required for reliable multi-hop traversal.

*Patent pending. Application No. 102026000023146; Application No. IT202500035167.

Rigid Parallelism. Ingestion pipelines either process documents sequentially (Microsoft GraphRAG users report indexing times of hours to days for medium corpora [1]) or use fixed parallelism that risks out-of-memory (OOM) crashes on heterogeneous hardware. No existing RAG system adapts its concurrency level in real time based on backend health.

Cross-Domain Isolation. Organizations maintaining multiple specialized knowledge bases face a dilemma: merge everything into a single KB (losing domain specialization and polluting LLM context with irrelevant information) or require users to manually select the target KB. No system discovers implicit structural connections between separate knowledge graphs at query time.

Contributions. We present Noēsis, a fully implemented and validated distributed knowledge system that goes beyond retrieval: it builds interconnected knowledge networks from heterogeneous sources and discovers emergent connections across domains that are not explicitly present in any single document—enabling on-premises reasoning with a single consumer GPU. The system solves all three problems above through:

1. A **bidirectional graph traversal** algorithm with forward-pass Graph-Feedback context resolution (n-gram scoring + recency decay) and backward-pass degree-based reconnection (§3);
2. An **AIMD concurrency controller** that transfers TCP congestion control principles to RAG pipeline orchestration with Redis-backed persistent state (§4);
3. **Moēsis**: automatic domain-aware profiling, classification, and selective quantization of Mixture-of-Experts models with runtime re-adaptation (§5);
4. **Mesh**: hierarchical fingerprint-based cross-KB routing with runtime structural discovery and adaptive Natural Break thresholds (§6).

2 System Architecture

Noēsis is a *decoupled* architecture comprising five independent components communicating via HTTP REST and Redis message queues:

- **Cortex**: LLM inference engine (llama.cpp + Moēsis optimization). Runs on dedicated GPU hardware, independently scalable.
- **Neuron**: Knowledge graph extraction library. Implements bidirectional traversal, entity deduplication, and cross-linking.
- **Synapsis**: Orchestration layer with distributed job queue, AIMD controller, and API gateway.
- **Retina**: Audio/video ingestion module (C++ VAD engine + speech-to-text).
- **Mesh**: Cross-KB semantic router operating as a shared in-process kernel (< 2 ms latency).

This separation is architecturally critical: the inference backend can be replaced (e.g., with AWS Bedrock, Claude API, or Ollama) without modifying the extraction or routing logic. The system adapts automatically to available hardware with *zero manual configuration*, scaling from consumer GPUs to multi-GPU servers. Stress tests confirm stable operation even at 6 GB VRAM with no crashes or data loss.

The extraction pipeline is *source-agnostic*: Neuron operates on any textual corpus, including PDF and DOCX documents, web pages, audio/video transcriptions, and software source code repositories. This generality enables domain-specific knowledge graphs from heterogeneous input sources without pipeline modifications.

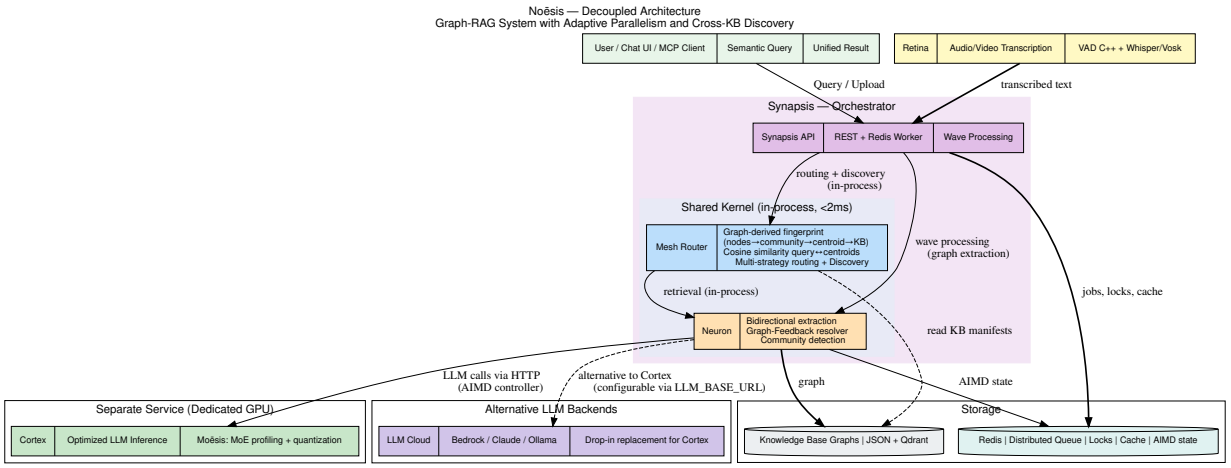


Figure 1: Noēsis system architecture. Five decoupled components communicate via REST APIs and Redis message queues, enabling independent scaling and backend replacement.

3 Bidirectional Graph Traversal

3.1 Problem Statement

When processing a long document (e.g., 300+ pages), existing systems split it into fixed-size chunks and analyze each independently. A concept introduced in Chapter 1 and referenced in Chapter 10 produces two disconnected nodes instead of a single entity with cross-document edges. Rolling summary approaches (maintaining an LLM-generated summary between chunks) add 80–120 seconds of overhead per long document and still miss long-range dependencies. Noēsis has been validated on single documents exceeding 300 pages, maintaining semantic continuity across the entire extraction.

3.2 Forward Pass with Graph-Feedback Context

Noēsis processes document slices sequentially, simulating human reading with *degrading memory*. After each slice, the **Graph-Feedback Context Resolver** selects the most relevant previously-extracted nodes to provide as context for the next slice:

$$\text{score}(n) = \text{relevance}(n) + \text{recency}(n) \tag{1}$$

where relevance(n) is computed via unigram + bigram keyword overlap between node n's label and the upcoming slice, and recency(n) provides a bounded bonus for recently-extracted nodes that decays with document distance. This mimics human working memory: recent concepts are vivid; distant ones fade unless directly relevant.

The context budget grows adaptively with document length, bounded between a minimum (to avoid starving short documents) and a maximum (to prevent context explosion), scaling linearly with the number of slices.

Document slicing uses *semantic boundaries* (headings, paragraph breaks) rather than arbitrary token offsets, ensuring each slice contains a complete conceptual unit.

3.3 Backward Pass: Degree-Based Reconnection

After the forward pass completes, a second pass identifies "early" nodes with low degree (few connections) and attempts to reconnect them to semantically related "late" nodes. This closes long-range dependencies that the forward pass could not see—analogous to a reader who, upon finishing a book, recognizes that an early character connects to a late plot event.

The backward pass activates only when the document is sufficiently long and produces enough nodes to benefit from reconnection; below these thresholds the graph is too small to benefit.

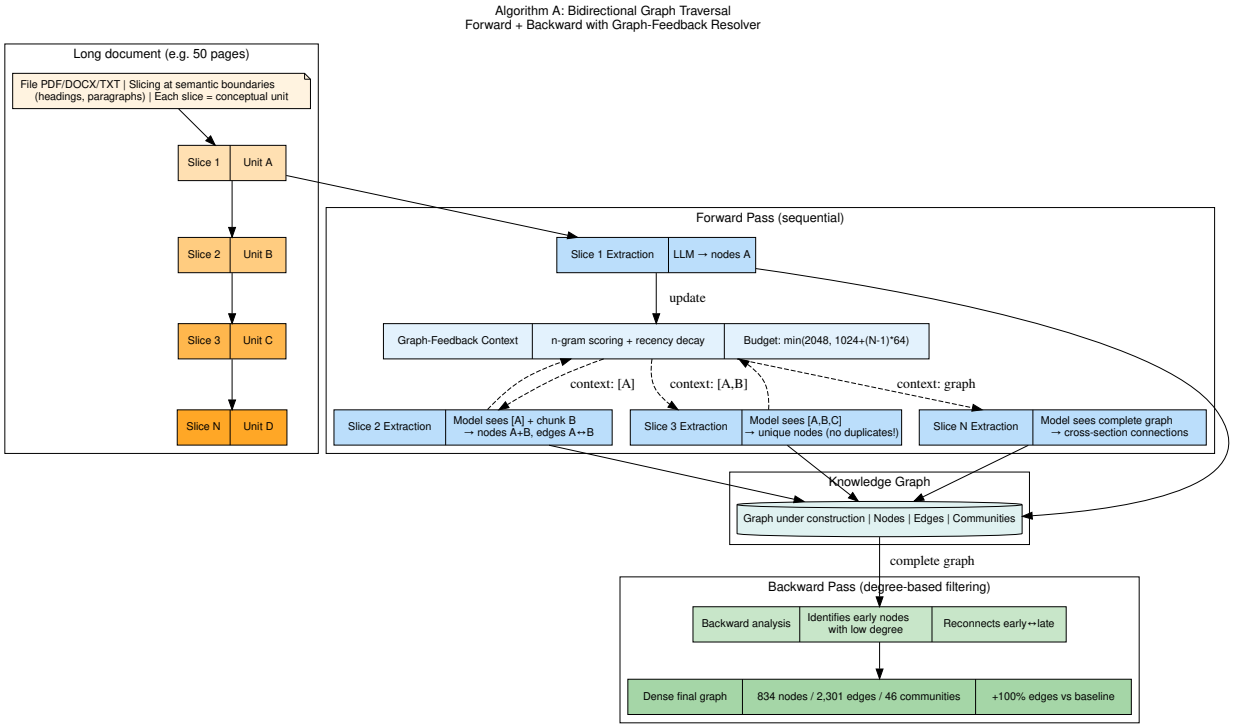


Figure 2: Bidirectional graph traversal: the forward pass builds the graph with degrading memory context, then the backward pass reconnects early low-degree nodes to semantically related late nodes.

3.4 Integrated Sub-Algorithms

Entity Deduplication (4-phase pipeline).

1. *Exact normalization* with per-source-file partitioning to avoid cross-document false merges.
2. *Locality-sensitive hashing (LSH) blocking + Jaro-Winkler* scoring with two innovations: (a) a **Shannon entropy gate** that protects low-information nodes (e.g., "Results", "Method") from automatic merging by detecting insufficient label complexity; (b) a **Leiden community boost** that increases the merge score when both nodes share a graph community.
3. *Variant pair detection*: identifies siblings differing only by numeric suffix (e.g., ASR1603 vs. ASR1604) and protects them from merge.
4. *LLM tiebreaker*: ambiguous pairs in an intermediate confidence band are resolved in batch via binary yes/no queries.

Cross-Linking: Pistol/Cannon Strategy. For small graphs, cosine similarity of node embeddings suffices ("pistol"). For larger graphs, an additional LLM pass examines an *ambiguity band* where embeddings are unreliable but the LLM can discover non-obvious causal relations ("cannon"). Example: discovering "stress" → "magnesium deficiency" → "guided exercise".

Adaptive Retry with Recursive Bisection. When the LLM produces truncated output, context overflow, or hollow responses, the system bisects the slice at a semantic boundary and retries recursively (max 3 levels = 8× expansion). This guarantees **zero data loss**: no content is silently discarded.

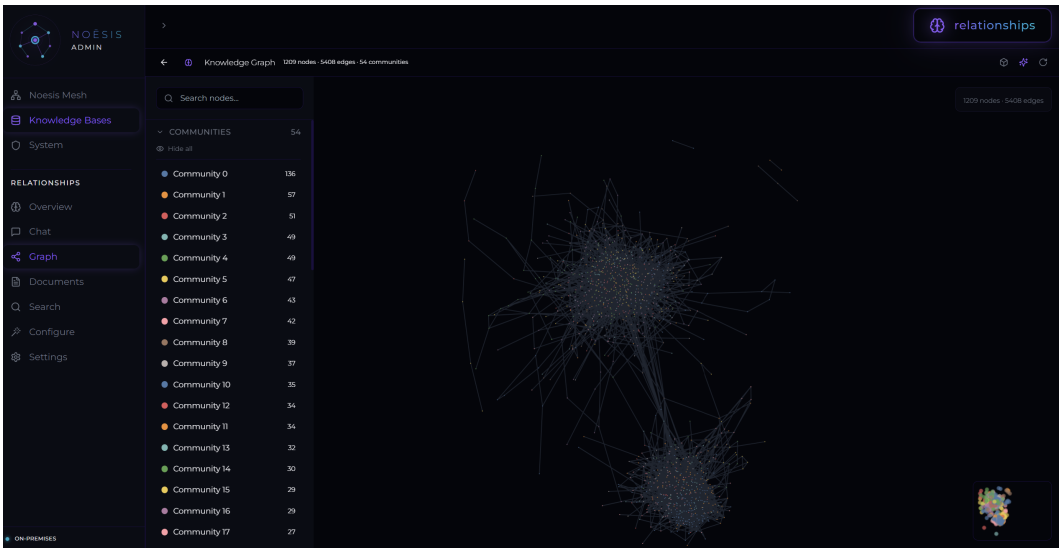


Figure 3: Intra-document cross-linking enabled by bidirectional traversal. The dense interconnection pattern between concepts from different document sections is a direct result of the backward pass reconnection—single-pass extraction produces sparse, fragmented subgraphs without these long-range edges.

Prompt Injection Protection. Since Noēsis processes user-uploaded documents (PDF, DOCX), a malicious file could inject instructions into the LLM during extraction. Each source file is wrapped in cryptographically-tagged containment delimiters that mark content as untrusted, and known LLM control sequences are neutralized before extraction to prevent instruction injection.

3.5 Measured Results

On a multilingual knowledge base (5 PDF documents):

- Baseline (single-pass, no feedback): 29 nodes, 36 edges, 4 communities.
- With bidirectional traversal: **253 nodes, 257 edges, 179 cross-links, 12 communities**.
- Full extraction run: 834 nodes, 2,301 edges, 46 communities.
- Backward pass contribution: substantial increase in cross-section edges versus forward-only baseline (magnitude varies by corpus structure).

4 AIMD Adaptive Parallel Processing

4.1 Problem Statement

Processing multiple documents in parallel accelerates ingestion but risks OOM crashes when the LLM backend cannot handle the load. Fixed parallelism either wastes GPU capacity (too conservative) or causes failures (too aggressive). No existing RAG system adapts its concurrency level at runtime based on observed backend behavior.

4.2 Algorithm Design

We adapt the Additive Increase / Multiplicative Decrease (AIMD) algorithm—well-known in TCP congestion control—to RAG pipeline orchestration. This transfer is non-trivial: TCP AIMD operates on packet-level round-trip times within a single network connection, while our controller

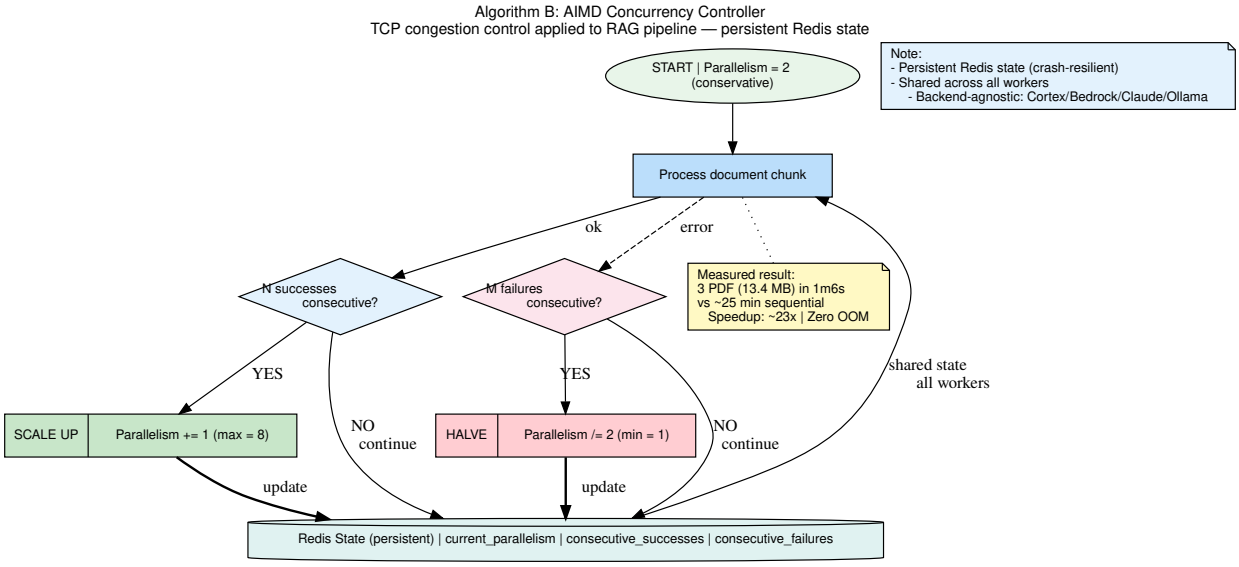


Figure 4: AIMD concurrency controller adapted from TCP congestion control to RAG pipeline orchestration. The controller increases concurrency after consecutive successes and halves it after consecutive failures, with state persisted in Redis across worker restarts.

operates on document-level processing times across a distributed system with heterogeneous backends.

The controller maintains its state in Redis (crash-resilient, shared across all worker instances):

- **Initial concurrency**: conservative start value.
- **Additive increase**: after a configurable streak of consecutive successes, $C \leftarrow C+1$ (bounded by a system-defined maximum).
- **Multiplicative decrease**: after a configurable streak of consecutive failures, $C \leftarrow \lfloor C/2 \rfloor$.

The controller is *backend-agnostic*: it adapts identically whether the inference backend is a local GPU (Cortex), a cloud API (AWS Bedrock, Claude), or a community server (Ollama). This is possible because Noēsis decouples inference from orchestration.

4.3 Hybrid Parallelism

A critical design choice: the system maintains *inter-document parallelism* (multiple documents processed simultaneously) while preserving *intra-document sequentiality* (slices within each document are processed in order). This is essential because the bidirectional traversal's degrading memory requires sequential slice processing to build correct cross-section connections.

4.4 Measured Results

- 3 PDF documents, 13.4 MB total: **1 min 6 s** with adaptive parallelism vs. ∼25 min sequential (23× **speedup**).
- Zero OOM events across all test configurations.
- Controller state survives worker crashes and restarts—no warmup period needed.

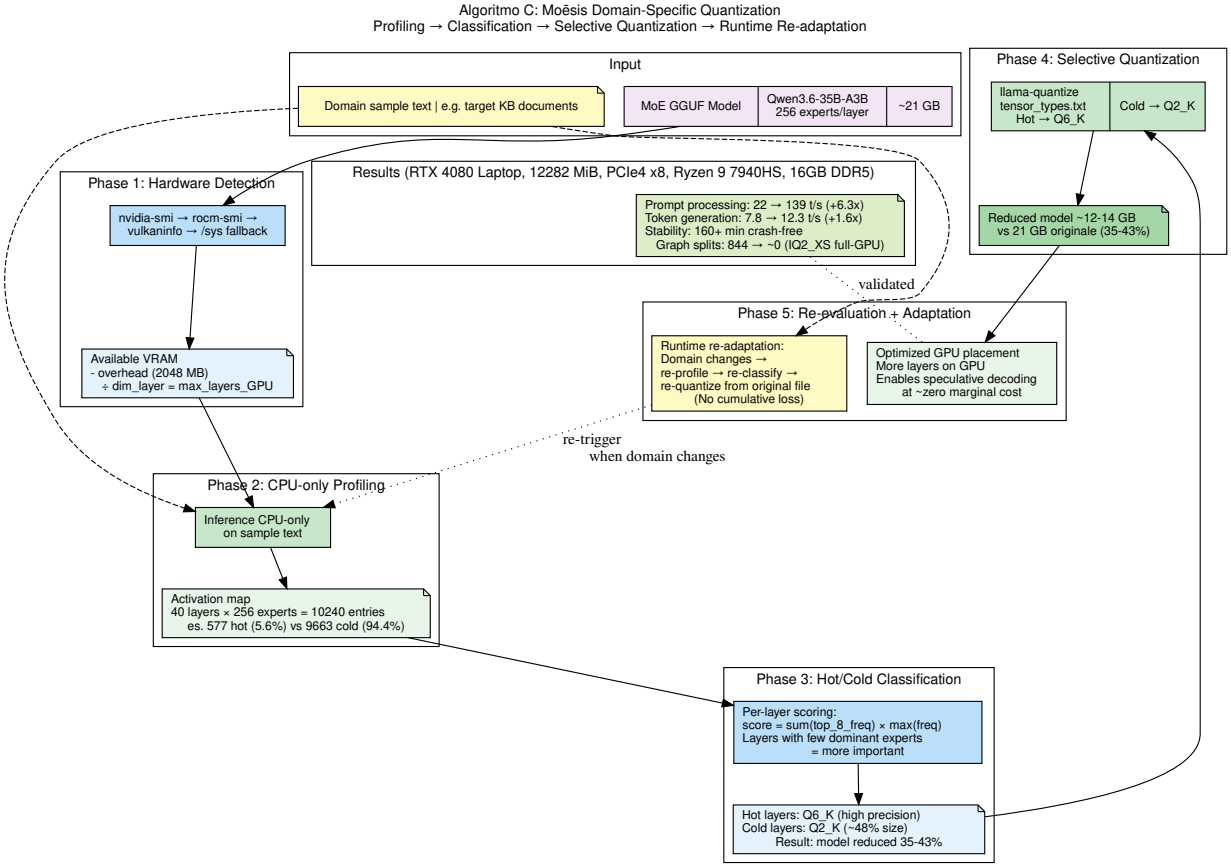


Figure 5: Moēsis three-level pipeline: (1) domain-aware profiling identifies hot/cold experts, (2) selective quantization applies different precision per layer, (3) runtime re-adaptation re-profiles when the domain changes.

5 Moēsis: Domain-Aware Selective Quantization

5.1 Problem Statement

Modern Mixture-of-Experts (MoE) models (e.g., Qwen3.6-35B-A3B with 256 experts per layer) load all experts into GPU memory even though only ∼8 are activated per token. On consumer hardware (6–12 GB VRAM), this causes immediate OOM. Existing solutions either require integration with specific inference frameworks (HybriMoE), apply uniform quantization across all layers regardless of activation patterns (imatrix, FIDDLER), or address only inference scheduling without compression (ExpertFlow, KTransformers).

5.2 Three-Level Pipeline

Level 1: Automatic Domain Profiling. Cortex runs CPU-only inference on a domain-representative text sample, recording per-layer per-expert activation frequencies. The resulting activation map captures which experts are relevant to the target domain. Typical outcome: only ∼5% of experts are "hot" (frequently activated for this domain); ∼95% are "cold".

Level 2: Selective Quantization and GPU Promotion. Each layer is classified using a scoring function that captures both the concentration of expert activations and the overall frequency of the most-used experts for the target domain. Hot layers (high concentration) are quantized at high precision (6-bit); cold layers at aggressive compression (2-bit, ∼48% size reduction). The model shrinks from 21 GB to ∼16 GB on disk (27% reduction).

The reduced model size enables a *promote-only* placement strategy: after quantization, the system re-evaluates how many layers fit entirely in GPU memory. Layers are promoted from CPU to GPU—never demoted—until VRAM is filled. This eliminates or substantially reduces the repeated CPU↔GPU data transfers that dominate inference latency on VRAM-constrained hardware. The profiling, classification, quantization, and placement steps execute automatically without manual configuration.

Level 3: Runtime Re-Adaptation. When the domain changes (e.g., from medical documents to engineering), the system re-executes profiling and re-quantizes from the *original full-precision model* (preserved as backup at first quantization). This prevents cumulative precision loss across adaptation cycles—a critical distinction from static quantization approaches.

5.3 Measured Results

Table 1: Moēsis performance on consumer hardware. The 6 GB configuration represents a laboratory stress test demonstrating system stability under extreme resource constraints, not a recommended operational configuration.

Metric	RTX 4080 Laptop (12 GB)	RTX 3060 (6 GB) [stress test]
Prompt processing	22 → 139 t/s (6.3×)	Stable (2–6 t/s)
Token generation	7.8 → 12.3 t/s (1.6×)	Stable (2–6 t/s)
Stability	Continuous	160+ min, zero crashes
Without Moēsis	Functional but slow	**Immediate OOM**

Moēsis is not merely a speed optimization—it is an *enabler*: on 6 GB VRAM, the system does not start without it. The 6 GB configuration demonstrates architectural resilience (zero crashes over 160+ minutes of continuous operation), while higher-VRAM configurations achieve substantially faster processing as more layers are promoted to GPU.

6 Mesh: Cross-KB Routing and Discovery

6.1 Problem Statement

Organizations maintaining multiple specialized knowledge bases face a fundamental tension. A single monolithic KB ("KB-GOD") pollutes the LLM context with irrelevant cross-domain information, degrading response quality. Separate KBs require users to know which domain to query and cannot discover implicit connections between domains. No existing Graph-RAG system implements automatic cross-KB routing with runtime structural discovery.

6.2 Hierarchical Fingerprint Routing

Each KB exports a semantic fingerprint derived from its graph structure:

$$\mathbf{fp}_{\mathrm{KB}} = \text{L2-norm}\left(\sum_{c \in \mathcal{C}} w_c \cdot \boldsymbol{\mu}_c\right) \tag{2}$$

where $\mathcal{C}$ is the set of Leiden communities, w_c is the community weight, and $\boldsymbol{\mu}_c$ is the mean embedding of community c's member nodes.

Query routing uses a multi-strategy approach with fallback:

1. *Primary*: cosine similarity between query embedding and KB fingerprints.
2. *Secondary*: community-level semantic overlap.

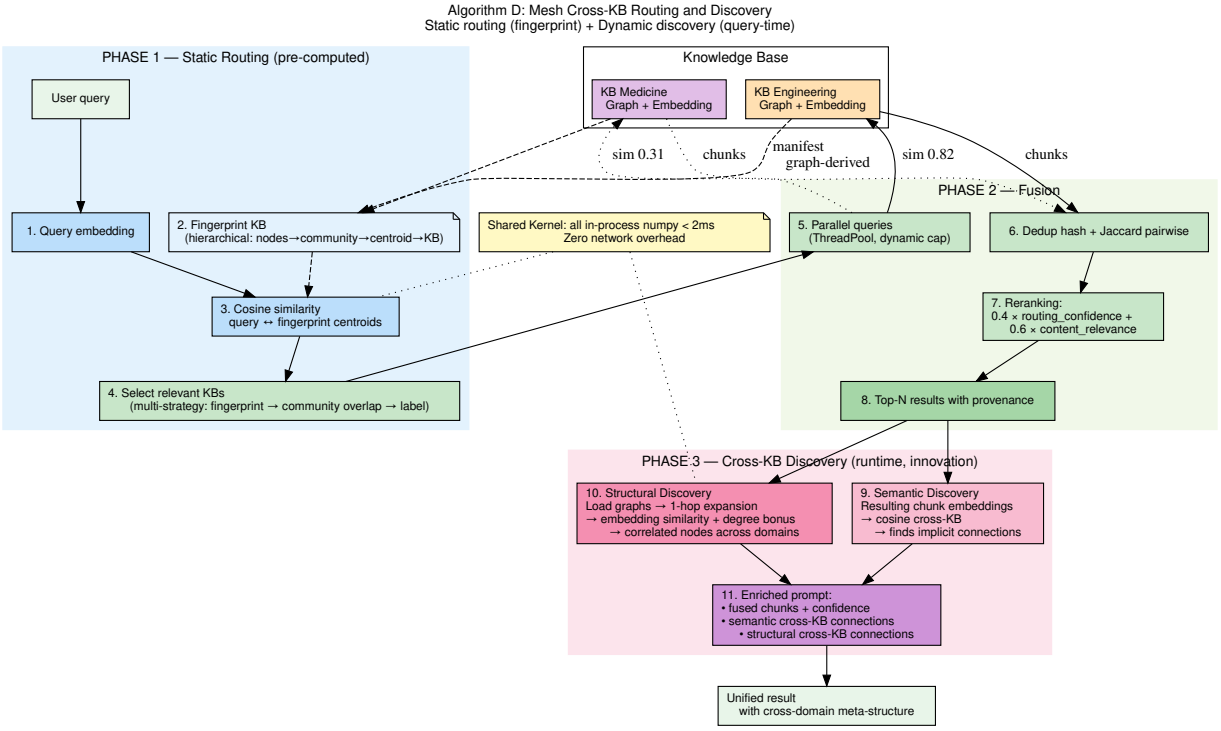


Figure 6: Mesh cross-KB routing: queries are routed to relevant knowledge bases via hierarchical fingerprint matching, then runtime structural discovery identifies emergent connections across separate graphs.

3. *Tertiary*: lexical label matching (ensures routing even without an active embedding model).

Selected KBs are queried in parallel via a thread pool. Results undergo 5-phase fusion: extraction → confidence scoring (weighted combination of routing affinity and content relevance) → Jaccard n-gram deduplication → reranking → top-k cap.

6.3 Runtime Structural Discovery

After initial retrieval, Mesh discovers *emergent* connections between KBs through two mechanisms:

Chunk-Level Semantic Discovery. Computes embeddings of retrieved chunks and identifies cross-KB pairs with high similarity that were not pre-computed in any manifest.

Node-Level Structural Discovery. Loads graphs of selected KBs, identifies query-relevant nodes, expands to 1-hop neighbors, and compares nodes across KBs using embedding similarity with a *structural bonus* based on graph role similarity (incorporating node degree and community membership).

6.4 Natural Break Adaptive Threshold

Instead of a fixed global similarity threshold, Mesh computes the optimal threshold for each specific query:

1. Compute all pairwise similarities between chunks/nodes of different KBs.
2. Sort in descending order and compute first differences (slopes).
3. Detect the "knee": the index where the slope exceeds a statistically-derived threshold based on the distribution's mean and standard deviation.

4. Fallback: if no clear knee exists, use a high percentile as threshold.

5. Safety clamp: the adaptive threshold never falls below a fixed domain-calibrated baseline.

Behavior. For semantically close KBs (e.g., two domains sharing overlapping terminology), the threshold lowers automatically, discovering real connections that a fixed threshold would filter. For distant KBs (e.g., domains with no shared vocabulary), it remains high, filtering noise.

6.5 Measured Results

Evaluation example. Query: "How can stress create hormonal dysfunctions and affect relationships? What methods mitigate negative effects?"
KBs involved: three domain-specific knowledge bases covering health, behavioral science, and interpersonal dynamics.
Model: Gemma 4 E2B (2.3B effective parameters, designed for smartphones, <2 GB RAM).

- **Without** Mesh discovery: Fragmented response with explicit disclaimers about incomplete information. Model admits inability to connect domains.

- **With** Mesh discovery + Natural Break: Organic, confident response connecting cortisol → endocrine system → estrogen/progesterone → emotional regulation → couple communication. Proposes structured strategies (individual: mindfulness; couple: empathic listening, physical connection → oxytocin). Zero disclaimers.

The emergent connection "physical contact → oxytocin → couple bonding" exists in no single document—it was discovered by Mesh at query time by comparing nodes across separate graphs. A 2.3B-parameter edge model achieves multi-hop cross-domain reasoning that the same model cannot perform without Mesh—demonstrating that the architecture, not model scale, enables emergent discovery.

Critically, Mesh is equally effective at *rejecting* irrelevant connections: a knowledge base built from a software codebase correctly produces zero cross-KB connections with semantically unrelated domains, demonstrating that the routing is selective rather than indiscriminate—avoiding the context pollution that degrades monolithic approaches.

Shared Kernel Architecture. Routing and discovery execute *within* the API process itself (in-process NumPy operations), not as external service calls. This Shared Kernel pattern eliminates network round-trips entirely, achieving routing latency <2 ms. The same in-process execution applies to runtime structural discovery—enabling real-time cross-KB reasoning without the latency penalty of microservice architectures.

7 Evaluation Summary

Table 2 summarizes the capabilities introduced by Noēsis relative to common design patterns in existing Graph-RAG systems.
Note: "Existing Approaches" reflects common patterns observed in published documentation of systems including GraphRAG, LightRAG, LazyGraphRAG, HippoRAG, and RAGFlow as of August 2026. Individual systems may address some capabilities through extensions or plugins not reflected here.

8 Related Work

Graph-RAG Systems. Microsoft GraphRAG [1] introduced community-based summarization but uses static chunking and sequential processing. LightRAG [2] reduces indexing cost but does not address cross-document continuity. LazyGraphRAG [5] eliminates pre-summarization but still

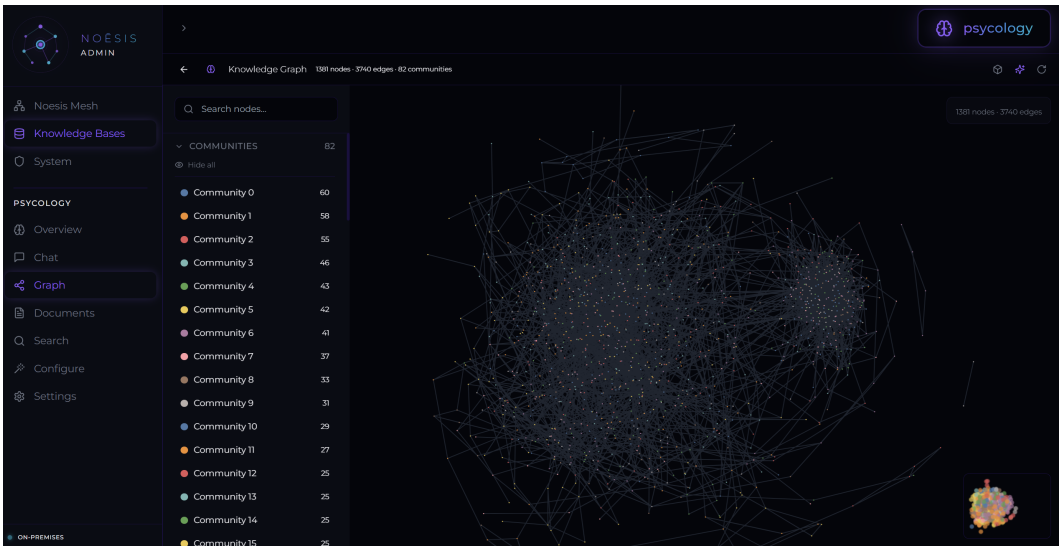


Figure 7: Two-dimensional visualization of a knowledge graph generated by Noēsis (1,381 nodes, 3,740 edges, 82 communities). Node colors indicate Leiden communities; edge thickness reflects relationship strength.

chunks independently. HippoRAG [3] models hippocampal memory for retrieval and extracts simplified triples (OpenIE), but does not build dense inter-document graph structures.

LLM Concurrency Control. CONCUR [4] applies AIMD to GPU KV-cache admission control at the serving engine level. Our work transfers AIMD to application-level pipeline orchestration with persistent cross-worker state—a fundamentally different operating context.

MoE Optimization. FIDDLER profiles expert popularity for CPU-GPU placement but applies uniform precision across all layers. HybriMoE requires integration with the kTransformers framework. ExpertFlow optimizes expert caching for inference but does not compress the model. imatrix supports domain-specific calibration data but still quantizes all layers uniformly. Moēsis uniquely combines per-layer activation profiling with selective precision assignment—hot layers retain higher precision while cold layers are aggressively compressed—enabling runtime re-adaptation as workloads evolve.

Multi-KB Routing. Adaptive-k [6] uses largest-gap detection for retrieval quantity selection—a different problem from connection discovery thresholds. No prior system implements cross-graph structural discovery at query time.

9 Conclusion

We have presented Noēsis, a fully implemented Graph-RAG system that introduces four algorithmic innovations addressing fundamental limitations of existing approaches. The bidirectional traversal with Graph-Feedback context produces knowledge graphs with density unattainable by independent chunking. The AIMD controller enables safe, adaptive parallelism across heterogeneous hardware. Moēsis makes MoE models viable on consumer GPUs through domain-aware selective quantization. And Mesh enables automatic cross-domain reasoning without knowledge base fusion, achieving emergent multi-hop discovery on small on-premises models that cannot perform such reasoning without the system.

The system has been fully implemented and extensively tested on a corpus exceeding 60 documents and 170 MB across multiple knowledge bases—including PDF, DOCX, a complete software codebase, and single documents exceeding 300 pages—processing documents in multiple languages (including non-Latin scripts) and heterogeneous source types. All reported measurements are from real-world workloads, not synthetic benchmarks.

Table 2: Capabilities of Noēsis versus common patterns in existing Graph-RAG approaches.

Capability	**Existing Approaches**	**Noēsis**
Cross-chunk context during extraction	Rare or limited	✓
Adaptive ingestion parallelism	Not observed	✓
Independently scalable inference backend	Configurable backends, not decoupled services	✓
MoE-aware selective quantization	Uniform quantization (all layers equal)	✓
Cross-KB routing (<2 ms)	Not addressed	✓
Cross-KB structural discovery (runtime)	Not addressed	✓
Adaptive similarity thresholds	Fixed thresholds	✓
Multi-pass entity deduplication	Basic or none	✓
Extraction-time prompt injection containment	Not addressed	✓
6 GB stress test → multi-GPU scaling	Typically requires >16 GB	✓

References

[1] D. Edge, H. Trinh, N. Cheng, et al. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[2] Z. Guo, L. Zhao, et al. LightRAG: Simple and Fast Retrieval-Augmented Generation. *EMNLP*, 2025.

[3] B. Gutierrez, Y. Yang, et al. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831*, 2024.

[4] CONCUR. AIMD-based Concurrency Control for LLM Batch Inference. *arXiv preprint arXiv:2601.22705*, 2026.

[5] Microsoft Research. LazyGraphRAG: Setting a new standard for quality and cost. *Microsoft Research Blog*, 2024. `https://www.microsoft.com/en-us/research/blog/lazygraphrag-setting-a-new-standard-for-quality-and-cost/`

[6] Adaptive-k. Adaptive Retrieval with Largest-Gap Selection. *arXiv preprint arXiv:2506.08479*, 2025.