Engineering a Production RAG-Based Agent Platform

Eduards Mukans[1,*], Guntis Barzdins[2]

[1]University of Latvia
[2]University of Latvia, IMCS

Abstract

This paper presents a production-grade Retrieval-Augmented Generation (RAG) agent platform Graip.AI Agent Labs (*https://agent-labs.graip.ai*) developed by Graip.AI (*https://graip.ai*), introducing two key innovations: index-time contextual enrichment, where LLMs augment document chunks with additional context to improve embedding quality, and multi-vector representations, where each chunk is encoded both directly and via multiple LLM-generated question embeddings to better capture diverse query formulations. The platform combines modular agent orchestration, hybrid retrieval with adaptive routing, and human-in-the-loop workflows to support enterprise use cases such as document question answering and system integration. Evaluation results from deployment show that preprocessing and representation strategies are the dominant factors in RAG effectiveness.

Keywords

Retrieval-augmented generation, large language models, vector databases, LLM agents, knowledge bases, production systems

1. Introduction

Large Language Models (LLMs) are fundamentally limited by static knowledge, susceptibility to hallucination, and inability to access private information. Retrieval-Augmented Generation (RAG) addresses these limitations by grounding LLM outputs in external knowledge retrieved at inference time [1]. Enterprise RAG adoption reached 51% in 2024 [2], yet the CRAG benchmark [3] reveals that state-of-the-art industry RAG solutions achieve only 63% hallucination-free accuracy — demonstrating both RAG's value and the distance to production reliability.

This paper presents the architecture of a production RAG-based agent platform developed by Graip.AI. The platform is built on Python, LangChain/LangGraph for orchestration, FastAPI for serving, PostgreSQL with pgvector for unified relational and vector storage, Azure Blob Storage for document persistence, Celery with Redis for asynchronous processing, and Kubernetes with Helm for deployment. It serves business use cases spanning document-grounded question answering, natural language database querying, third-party system integration (SAP, email, external web services), and human-supervised agent actions. The platform's design draws on the authors' prior experience with LLM-augmented NLP pipelines: using GPT-3 to generate synthetic training features that improved NER performance in multilingual shared tasks [4], employing GPT-4 for data augmentation in adverse drug event discovery from social media [5], building context-enriched classification pipelines with LLM-generated insights [6], and investigating vector embedding representations for dictionary modelling tasks [7]. These experiences with quality augmentation through LLM-generated features, staged ML pipeline architectures, and the interplay between vector representations and downstream task performance directly informed the platform's multi-vector embedding strategy, contextual enrichment pipeline, and modular agent design.

The system operates in four modes: (1) agent and knowledge base configuration, where administrators define agents with system prompts, model selections, and tool bindings, and provision knowledge bases as coordinated pgvector tables, BM25 columns, and blob storage containers; (2) document ingestion

Baltic DB&IS 2026 Conference Forum and Doctoral Consortium, 28 June - 1 July 2026, Tartu, Estonia
*Corresponding author.
eduards.mukans@lu.lv (E. Mukans); guntis@latnet.lv (G. Barzdins)
https://emukans.github.io (E. Mukans); https://research.lu.lv/en/persons/guntis-bÄĄrzdiÅĘÅą/ (G. Barzdins)
0009-0000-2004-5431 (E. Mukans); 0000-0002-3804-2498 (G. Barzdins)

with multi-vector indexing, implementing an asynchronous pipeline that parses, chunks, deduplicates, contextually enriches, and embeds documents into multiple vector representations per chunk; (3) query processing with adaptive routing and hybrid retrieval, classifying queries by complexity and dispatching to direct LLM response, knowledge base lookup via hybrid BM25/dense retrieval with Reciprocal Rank Fusion (RRF) and cross-encoder re-ranking, SQL query execution, or multi-hop iterative retrieval; and (4) human-in-the-loop approval, using LangGraph's `interrupt()` mechanism to pause graph execution before dangerous actions or sensitive response generation.

The paper is organised as follows. Section 2 reviews the RAG paradigm evolution. Section 3 reviews the technology stack. Sections 4–8 detail each architectural element: agent configuration (Sect. 4), document ingestion (Sect. 5), query processing (Sect. 6), agent orchestration (Sect. 7), and human-in-the-loop (HITL) workflows (Sect. 8). Section 9 describes the evaluation strategy for RAG and agent system quality. Section 10 discusses future directions and Sect. 11 concludes.

2. The RAG paradigm evolution

The foundational taxonomy of Gao et al. [8] identifies four increasingly sophisticated RAG paradigms. **Naive RAG** implements the canonical retrieve-read pipeline: documents are chunked, embedded, and retrieved via cosine similarity before concatenation into the LLM prompt. On the CRAG benchmark, this reaches 44% accuracy — a modest 10-percentage-point improvement over unaugmented LLMs [3].

Advanced RAG introduces pre-retrieval optimisation (query expansion, rewriting, Hypothetical Document Embeddings) and post-retrieval processing (re-ranking, filtering, compression). The most impactful technique is hybrid search with cross-encoder re-ranking: Pezzuti et al. [9] report MRR10 of 0.8633 on MS MARCO DEV-SMALL.

Modular RAG [10] transcends the linear pipeline to offer reconfigurable frameworks in which specialised components — search modules, memory modules, routing, and task adapters — can be freely composed.

Agentic RAG [11] embeds autonomous agents with reflection, planning, tool use, and multi-agent collaboration into the retrieval pipeline, achieving 94.5% accuracy on HotpotQA with GPT-4o-mini.

The Graip.AI platform implements elements from all four paradigms: hybrid retrieval with re-ranking (Advanced RAG), sub-graph composition and adaptive routing (Modular RAG), and self-reflective corrective retrieval with tool execution (Agentic RAG).

3. Technology stack review

Figure 1 presents the platform's architecture, showing the interaction between all core components.

LLM Providers. The platform currently relies on third-party LLM providers, consuming state-of-the-art models via API from OpenAI, Anthropic, Google (Gemini), DeepSeek (via OpenRouter), and Qwen (via OpenRouter). This multi-provider strategy enables per-agent model selection, resilience against provider outages, and avoidance of GPU infrastructure costs. The principal limitation is latency variability: generation typically takes 4,000 – 8,000ms, constituting 70 – 90% of total pipeline latency. A further concern is data sovereignty — sensitive enterprise data must traverse third-party APIs, which may be unacceptable in regulated domains. To address both limitations, the platform is planned to support self-hosted LLM deployment, enabling organisations to run open-weight models (e.g., Llama, Qwen, DeepSeek) on their own infrastructure whilst using the same LangGraph orchestration and tool-binding interfaces.

LangChain and LangGraph. LangGraph extends LangChain with stateful graph execution, providing typed state schemas, conditional edges, and native cycle support — capabilities essential for self-corrective workflows such as CRAG and IRCoT, as well as the human-in-the-loop interrupt and resume patterns described in this work. LangGraph was selected as the primary orchestration framework over open-source alternatives at the outset of platform development in 2024.

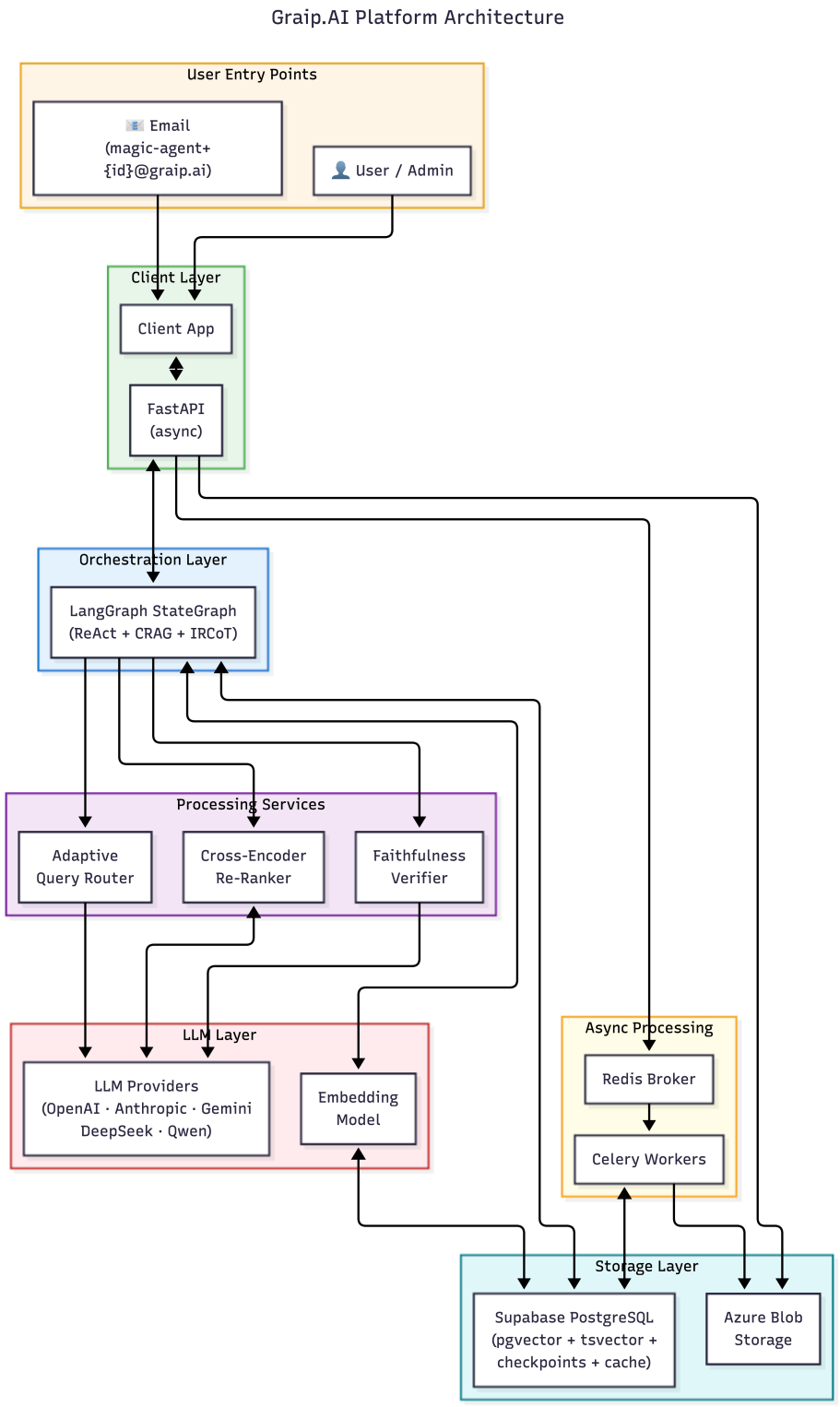

Figure 1: Graip.AI platform architecture. The diagram shows all core components and their interactions across the client, orchestration, processing, LLM, storage, and async layers.

PostgreSQL consolidates five storage concerns in a single instance: relational data, pgvector with HNSW indexing, `tsvector` columns for BM25, PostgresSaver checkpoints, and semantic cache.

FastAPI provides asynchronous request handling critical for concurrent queries during LLM inference. **Azure Blob Storage** persists original documents for provenance and re-processing. **Celery** with **Redis** handles asynchronous ingestion, ensuring uploads never block queries. **Celery Beat** enables scheduled

re-indexing. **Kubernetes with Helm** manages deployment with separate worker pools isolating query-serving from ingestion pods.

4. Agent and knowledge base configuration

Before any query can be processed, administrators configure agents and their associated knowledge bases. An *agent* is defined by a system prompt, a selected LLM model, and a set of tool bindings. A *tool* is any capability the agent may invoke: a predefined LangGraph pipeline (analogous to RPA, where nodes and flow are strictly defined), an integration with a third-party system (SAP, email dispatch, external web services), a lookup in a related knowledge base, or a text-to-SQL query against a linked database. The agent's behaviour depends entirely on the configured toolset and routing logic.

A *knowledge base* is provisioned as a coordinated set of resources: a pgvector-enabled table with HNSW indexing for dense retrieval, a `tsvector` column for BM25 sparse retrieval, and an Azure Blob Storage container for source documents. Knowledge bases are configured independently and may be shared across multiple agents. The knowledge source may be a file (PDF, image, plain text, CSV, JSON) or a publicly available URL, with parser selection, including OCR via Azure Document Intelligence, Google Cloud OCR, or AWS Textract.

The separation of agent and knowledge base configuration enables multi-tenancy (different agents share knowledge bases with tailored retrieval parameters) and flexibility (per-KB embedding models accommodate multilingual or domain-specific content). The limitation is configuration complexity: the multi-dimensional parameter space currently requires manual tuning.

5. Document ingestion pipeline

Recent benchmarks confirm that chunking strategy is among the highest-leverage decisions in RAG pipeline design. An NVIDIA evaluation across five datasets and seven strategies found that page-level chunking achieved the highest average accuracy (0.648) with the lowest standard deviation (0.107), outperforming both token-based and section-level alternatives [12]. Even within the same document category, however, optimal strategies varied: across three financial datasets the best approach differed each time — 1 024-token chunks for one (0.579), 512-token chunks for another (0.681), and page-level chunking for a third (0.520). Extreme chunk sizes (128 and 2 048 tokens) consistently underperformed the mid-range, and query complexity correlated with the preferred granularity — factoid queries favoured 256 – 512-token chunks, whilst analytical queries requiring broader context benefited from 1 024-token or page-level segments. These findings establish data preprocessing as a dominant factor in retrieval quality and motivate the multi-strategy ingestion pipeline described below. Figure 2 illustrates the pipeline.

When a user uploads documents, FastAPI stores originals in Azure Blob Storage, creates a metadata record with status `processing`, and queues a task to Redis. The Celery worker executes five stages:

Stage 1: Parsing. The objective of this stage is to transform source material of any type into semantically searchable text. The worker fetches the original resource from Azure Blob Storage and selects a conversion strategy appropriate to the content type. Structured documents (PDF, DOCX, HTML) parsed mainly with LlamaParse. Image-based and scanned documents are digitised via optical character recognition, with the platform integrating three cloud OCR providers — Azure Document Intelligence, Google Cloud OCR, and AWS Textract — whose selection is configurable per knowledge base according to language coverage, accuracy requirements, and cost constraints; a self-hosted DeepSeek-OCR-v2 integration is planned for environments where document data must not leave the organisation's infrastructure. For non-textual knowledge sources, additional extraction methods are applied: images are annotated with descriptive metadata to enable semantic search over visual content, URLs are scraped to retrieve the underlying textual content, and binary formats (e.g. CSV, JSON) are converted to structured text suitable for downstream chunking.

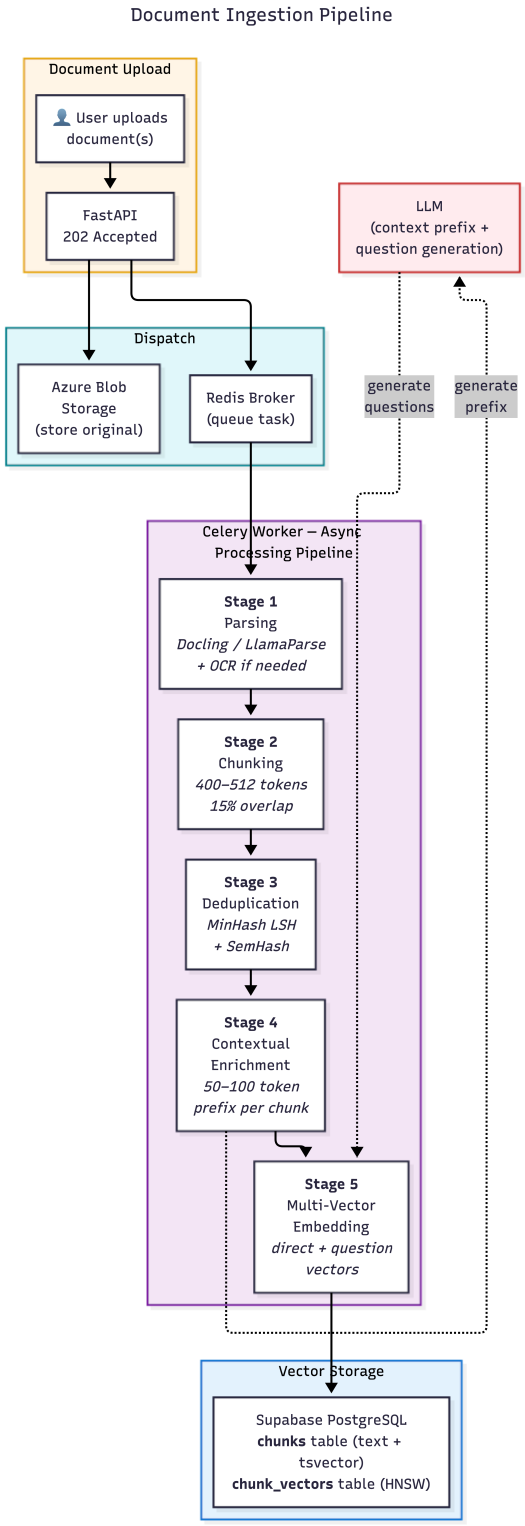

Figure 2: Document ingestion pipeline. Stages 4 – 5 require LLM calls for contextual enrichment and question generation.

Stage 2: Chunking. Recursive character splitting at 400 – 512 tokens with 15% overlap, preserving section boundaries and retaining positional metadata (page, section, offset). Page-level chunking is available for paginated financial and legal documents.

Stage 3: Deduplication. Two-tier: MinHash LSH for near-duplicate text, SemHash (model2vec) for semantic duplicates — paraphrased content that differs lexically but conveys identical information

(130K samples in 7s).

Stage 4: Contextual enrichment. Following Anthropic's approach [13], an LLM generates 50 – 100 tokens of document-level context prepended to each chunk before embedding. This LLM-driven quality augmentation at indexing time — using a large model to enrich data consumed by the retrieval pipeline — generalises a pattern the authors previously applied in classification tasks: GPT-4-generated rationales and synthetic features improved adverse drug event detection [5], and context-enriched classification pipelines demonstrated that LLM-generated insights can substantially improve downstream model performance even when the LLM itself is not used at inference time [6].

Stage 5: Multi-vector embedding. For each chunk, the system generates: (a) a *direct chunk embedding* of the contextualised text, and (b) 3 – 5 *question-based embeddings* — the LLM generates questions the chunk could answer, each embedded independently. At retrieval time, a user's question may match a generated question embedding even when it has low similarity to the declarative chunk embedding, bridging the query – document formulation gap that is acute in business contexts. This multi-representation strategy echoes findings from earlier embedding research: Mukans et al. [7] demonstrated that single-vector representations are insufficient to capture the full semantics of a concept, as word meaning requires richer contextual signals than a single embedding can encode. The multi-vector approach addresses this limitation by providing multiple complementary views of the same content. All vectors are stored in a `chunk_vectors` table with a foreign key to `chunks` and a `vector_type` column (`direct` or `question`), resolving to the source chunk regardless of which vector matched.

5.1. Embedding model selection

The embedding model is configurable per knowledge base. All LLM and embedding calls are routed through LiteLLM, a unified client that provides a consistent interface across providers. The platform primarily relies on OpenAI and Anthropic embedding models: OpenAI's `text-embedding-3-small` produces 1,536-dimensional vectors (used by default), whilst `text-embedding-3-large` scales up to 3,072 dimensions and supports Matryoshka dimensionality reduction (allowing truncation to as few as 256 dimensions) for a flexible compression – quality trade-off. Anthropic's embedding models offer dimensionalities ranging from 256 to 2,048, with a default of 1,024. Higher-dimensional representations generally yield better retrieval accuracy but require proportionally more storage and slower HNSW index construction.

The principal limitations of the ingestion pipeline are threefold: storage multiplication (4 – 6 vectors per chunk due to the multi-vector strategy), LLM call cost during ingestion (∼100 calls per 50-chunk document for contextual enrichment and question generation), and the lack of incremental re-chunking— any change to chunking or embedding parameters requires full re-processing of the affected knowledge base.

6. Query processing and retrieval architecture

Queries enter the LangGraph StateGraph through FastAPI and pass through adaptive routing, retrieval, verification, and delivery.

Adaptive routing. A router node classifies each query and dispatches to: (1) direct LLM response for simple queries; (2) knowledge base lookup via hybrid retrieval; (3) SQL query execution via template retrieval; or (4) multi-hop iterative retrieval. The possible routes depends on enabled tools in the agent configuration.

Hybrid retrieval. The knowledge base route executes BM25 sparse search and dense vector search in parallel, then concatenates the results. The dense search matches against both direct chunk embeddings and generated question embeddings, resolving all matches to the original chunk. An LLM-based relevance grading step then evaluates the combined candidate set, filtering out chunks that are not pertinent to the query and retaining only those judged relevant.

SQL query route. A dedicated knowledge base stores parameterised SQL templates as chunks. The system retrieves matching templates via multi-vector search, re-ranks against user intent, and the LLM adapts the template with query-specific parameters. Queries are validated (SELECT-only, parameterised, schema-validated) before execution.

Multi-hop retrieval. For complex questions, an IRCoT-style loop [14] generates reasoning steps, derives sub-queries, retrieves additional context via hybrid search, and accumulates results until the chain completes.

Verification. All routes conclude with an LLM faithfulness check verifying each claim against retrieved context, followed by source citation attachment. Unsupported claims trigger re-retrieval or removal.

Limitations include the end-to-end latency, the cost of LLM-based relevance grading, and the fixed BM25/dense weighting — dynamic per-query weighting [15] would improve recall but is not yet implemented.

7. Agent orchestration and tool execution

The platform's agent architecture is a LangGraph StateGraph implementing the ReAct pattern [16] — interleaving reasoning (Thought) with environment actions (Action → Observation). Figure 3 illustrates the agent's interaction with tools and knowledge bases.

The flow begins when a user sends a query to an agent thread via the web platform or by email (using magic email addresses `magic-agent+{id}@graip.ai`). The agent's behaviour is determined by its configured tools. The Corrective RAG pattern [17] governs default retrieval: a `retrieve` node fetches documents, a `grade_documents` node applies relevance grading, and conditional edges route to generation, query reformulation, or hallucination checking.

LangGraph's sub-graph composition enables modularity: each tool binding corresponds to a sub-graph invoked via conditional edges from the router node. New tools are added by registering sub-graphs without modifying core agent logic. This modular, staged pipeline architecture — where each processing stage is independently testable and replaceable — follows the same design principle demonstrated in prior NLP competition pipelines, where staged approaches combining separate retrieval, classification, and LLM augmentation components consistently outperformed monolithic models [4, 5]. The typed state schema ensures nodes operate on well-defined structures, and conditional edges provide auditable routing.

8. Human-in-the-loop workflows

For actions carrying business risk, the platform uses LangGraph's `interrupt()` mechanism to pause graph execution, persist state via PostgresSaver to PostgreSQL Database, and resume from the exact checkpoint when the user provides input. Two HITL patterns are implemented:

Dangerous action approval. A risk classifier evaluates planned tool calls. Actions classified as dangerous (database modifications, bulk communications, deletions) trigger an interrupt. The user sees the intended action, affected systems, and estimated impact, then may approve, modify parameters, or reject. All decisions are recorded in an audit log.

Data review before generation. In sensitive domains (compliance, financial reporting), the system pauses after retrieval. The user reviews retrieved data and update it accordingly. The LLM generates a response grounded exclusively in the human-curated context.

PostgresSaver stores complete state snapshots at every super-step: thread identifier, graph state as JSONB, node metadata, and pending writes. A critical requirement is **idempotency**: since nodes re-execute upon resumption, any side effects before the interrupt must be safe to repeat. The platform addresses this by performing interrupts *before* executing side effects or moving the HITL actions to a separate node.

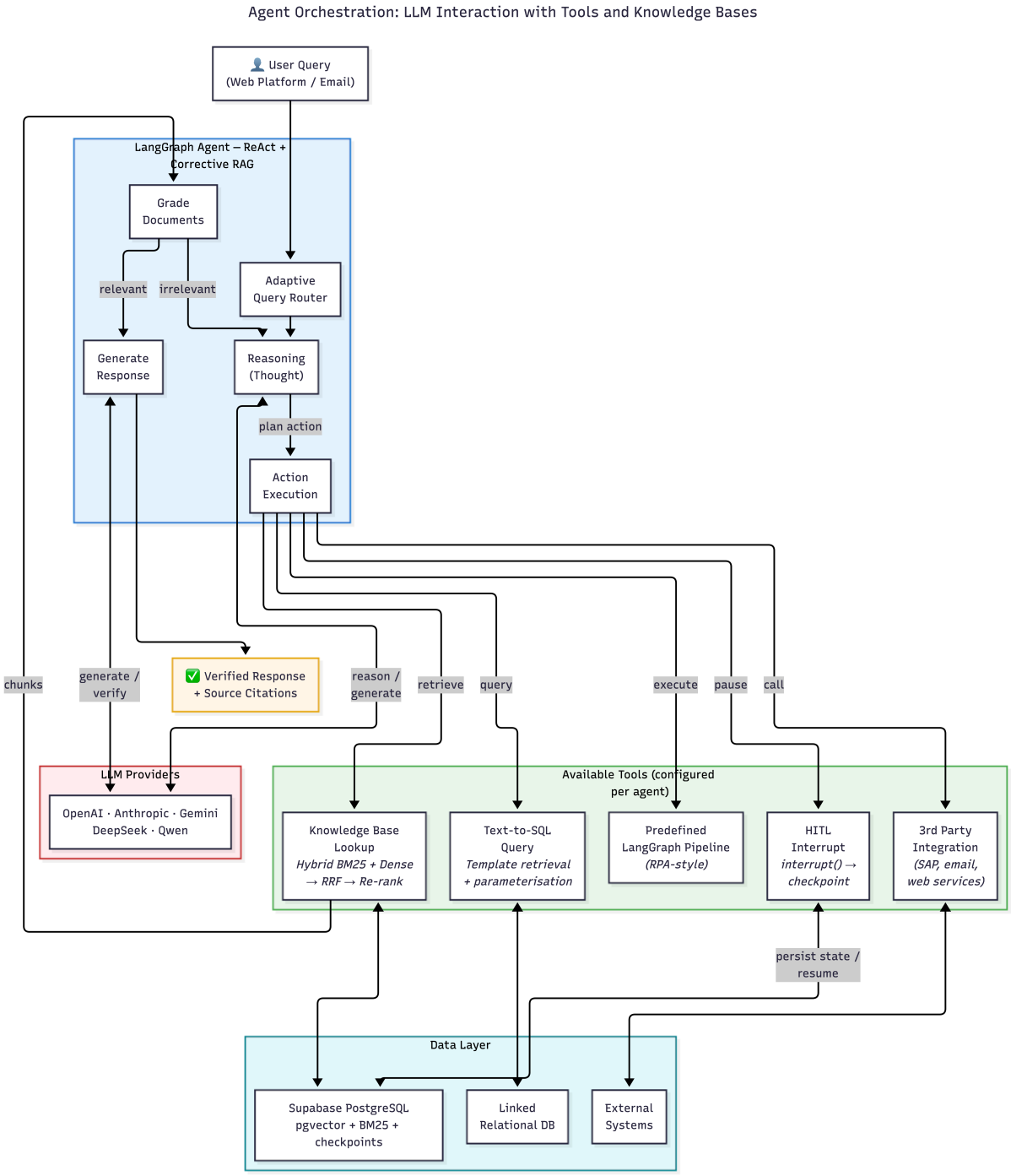

Figure 3: Agent orchestration: the LangGraph agent interleaves reasoning with tool execution via conditional edges. The corrective loop (Grade → Reason) implements the CRAG pattern.

HITL workflows address organisational trust — a deployment prerequisite in regulated industries. The implicit feedback signal provides data for future retrieval parameter tuning. Current limitations are unbounded review latency and single-reviewer approval; multi-reviewer chains are a future direction.

9. Evaluation and testing

Evaluating a production RAG platform requires a multi-layered strategy spanning RAG-specific quality metrics, automated LLM-based judgement, and traditional software engineering tests that verify pipeline correctness and stability.

9.1. RAG-specific metrics

Traditional information retrieval metrics such as nDCG, MAP, and MRR were designed for sequential human reading, yet LLMs process all retrieved documents holistically. The RGB benchmark demonstrates that LLMs heavily trust retrieved false information even when they possess correct internal knowledge [18], whilst CRUD-RAG finds that standard BLEU and ROUGE scores correlate poorly with factual consistency [19]. The platform therefore centres its quality assessment on four complementary metrics. **Faithfulness** measures factual consistency between a generated response and the retrieved context; RAGAS [20] operationalises this by decomposing answers into atomic claims and verifying each via natural language inference, achieving 95% agreement with human annotators. Scores above 0.8 indicate strong performance, and this metric is the most informative signal for detecting generation-stage hallucination. **Context Precision** quantifies the signal-to-noise ratio of retrieved passages — low scores indicate excessive irrelevant material that increases cost and introduces distractors. **Context Recall** measures retrieval completeness and is the only metric requiring ground-truth references, with scores above 0.8 indicating strong retrieval. **Answer Relevance** evaluates whether the response addresses the user's question, independent of factual accuracy.

These metrics map directly to pipeline stages: context precision and recall evaluate hybrid retrieval and re-ranking (Sect. 6), faithfulness evaluates verification and generation, and answer relevance evaluates adaptive routing and end-to-end response quality.

9.2. Frameworks and LLM-as-judge

The platform integrates several evaluation frameworks. RAGAS [20] provides reference-free evaluation for faithfulness, context precision, and answer relevance, making it practical in production where ground-truth maintenance is costly. DeepEval complements RAGAS with over 25 metrics, native Pytest integration, and 40+ red-teaming vulnerability tests; because it treats evaluations as unit tests with pass/fail thresholds, every deployment can be gated on minimum faithfulness and context precision scores. LangSmith provides per-node evaluators over LangGraph execution traces, producing node-by-node audit trails for both automated scoring and targeted human review.

LLM-based judgement substantially outperforms traditional surface-form metrics for open-ended RAG evaluation: G-Eval demonstrated that GPT-4 scoring achieves significantly higher correlation with human preferences than BLEU or ROUGE, which show low alignment on tasks requiring semantic understanding [21]. The RGB benchmark [18] further reveals that LLMs tend to trust retrieved false information even when they possess correct internal knowledge, reinforcing the platform's separation of retrieval evaluation from generation evaluation. Our approach therefore relies on LLM-based evaluation for the large majority of cases, reserving human evaluation for calibration and domain-specific validation.

9.3. Traditional pipeline and end-to-end testing

RAG metrics capture semantic quality, but a production system also requires traditional tests that verify pipeline correctness across releases. The platform employs a two-tier strategy. At the backend tier, each pipeline stage — parsing, chunking, embedding, retrieval, re-ranking, and generation — is covered by Pytest integration tests running against realistic data fixtures. LLM responses are recorded during an initial baseline run and cached locally; subsequent executions replay cached outputs, eliminating inference non-determinism and decoupling test reliability from API availability whilst exercising the full data flow through fusion, re-ranking, and post-processing. These tests verify structural contracts (chunk token budgets, RRF candidate counts, document identifier preservation) and run on every commit.

At the inference tier, end-to-end tests exercise the full user-facing path through the web UI, invoking live LLM APIs exactly as a real session would. Queries pass through adaptive routing, hybrid retrieval, generation, and faithfulness verification; assertions cover both structural properties (citation presence, JSON schema conformance, source attribution) and semantic quality via RAGAS or DeepEval scoring. Because these tests are non-deterministic and more expensive, they run as nightly or pre-release gate

checks. Together, the two tiers ensure that deterministic pipeline logic is validated continuously whilst end-to-end semantic quality is monitored at regular intervals.

10. Future directions

The deployment experience with the Graip.AI platform, combined with the trajectory of the 2023–2025 literature, suggests several directions for future development.

Self-hosted LLM and OCR deployment. A near-term priority is enabling organisations to run open-weight language models and self-hosted OCR engines (e.g., DeepSeek-OCR-v2) on their own infrastructure. This addresses data sovereignty requirements in regulated industries — where sending documents to third-party APIs may be impermissible — whilst retaining the same LangGraph orchestration interfaces and multi-vector indexing pipeline.

Agent configuration templates. The current configuration process requires administrators to manually specify system prompts, tool bindings, retrieval parameters, and re-ranking models. A library of pre-tuned templates for common business use cases — customer support, legal compliance, financial reporting, technical documentation Q&A — would encode evidence-based defaults (e.g., page-level chunking with conservative similarity thresholds for legal documents, aggressive re-ranking with smaller candidate sets for customer support) whilst remaining fully customisable.

Dynamic per-query retrieval weighting. The current hybrid retrieval pipeline uses a fixed weighting between BM25 and dense scores. Hsu et al. [15] demonstrate that a learned, query-dependent weighting function consistently outperforms fixed-weight methods; integrating such an approach would allow the system to emphasise lexical or semantic matching on a per-query basis.

Multi-modal RAG. The current pipeline processes text extracted from documents, discarding embedded images, charts, and diagrams. Extending ingestion and retrieval to handle visual content natively — via vision-language embeddings and multi-modal re-rankers — would substantially improve coverage for domains where visual information is critical, such as engineering documentation and medical imaging reports.

Collaborative multi-reviewer approval chains. The current HITL implementation supports single-reviewer approval. Extending this to sequential or parallel multi-reviewer chains would address organisational workflows where sensitive actions require sign-off from multiple designated approvers before execution.

11. Conclusion

We have presented the architecture of the Graip.AI RAG-based agent platform, detailing working principles, configuration, and production trade-offs for each core component: agent configuration, document ingestion, query processing, agent orchestration, and human-in-the-loop workflows.

Our deployment experience confirms that data preprocessing quality dominates end-to-end performance — contextual enrichment paired with multi-vector embedding effectively bridges the query – document formulation gap common in business contexts. Hybrid retrieval with cross-encoder re-ranking delivers the strongest quality-to-complexity baseline. Human-in-the-loop workflows with persistent graph state prove essential for enterprise adoption in regulated domains.

The platform's modular sub-graph composition — decomposing complex tasks into independently testable stages connected by well-defined interfaces — enables integration of new tools and retrieval strategies without modifying core agent logic, a principle informed by the authors' prior work on staged LLM-augmented NLP pipelines. Key limitations include storage multiplication from the multi-vector strategy, fixed BM25/dense weighting that does not adapt per query, and single-reviewer approval; addressing these through dynamic retrieval weighting, configuration templates, and multi-reviewer chains is the focus of ongoing work.

Acknowledgments

This work has been supported by the EU Recovery and Resilience Facility projects Language Technology Initiative (No 2.3.1.1.i.0/1/22/I/CFLA/002) and Latvian Quantum Initiative (No. 2.3.1.1.i.0/1/22/I/CFLA/001).

Declaration on Generative AI

*During the preparation of this work, the author(s) used **Claude Opus 4.6**, **Grammarly** in order to: **Grammar and spelling check**, **Paraphrase and reword**. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.*

References

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks, in: Advances in Neural Information Processing Systems, volume 33, 2020, pp. 9459–9474.

[2] Menlo Ventures, The State of Generative AI in the Enterprise, Technical Report, 2024. https://menlovc.com/2024-the-state-of-generative-ai-in-the-enterprise/.

[3] X. Yang, et al., CRAG – comprehensive RAG benchmark, in: Advances in Neural Information Processing Systems, Curran Associates, 2024, pp. 10470–10490.

[4] E. Mukans, G. Barzdins, RIGA at SemEval-2023 task 2: NER enhanced with GPT-3, in: Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023), Association for Computational Linguistics, Toronto, Canada, 2023, pp. 331–339.

[5] E. Mukans, G. Barzdins, RIGA at SMM4H-2024 task 1: Enhancing ADE discovery with GPT-4, in: Proceedings of the 9th Social Media Mining for Health Research and Applications (SMM4H 2024) Workshop and Shared Tasks, Association for Computational Linguistics, Bangkok, Thailand, 2024, pp. 23–27.

[6] E. Mukans, G. Barzdins, RIGA at SMM4H-HeaRD 2025: Context-enriched classification pipeline, in: Workshop Proceedings of the 19th International AAAI Conference on Web and Social Media (SMM4H-HeaRD 2025), 2025.

[7] E. Mukans, G. Strazds, G. Barzdins, RIGA at SemEval-2022 task 1: Scaling recurrent neural networks for CODWOE dictionary modeling, in: Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022), Association for Computational Linguistics, Seattle, USA, 2022, pp. 82–87.

[8] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, H. Wang, Retrieval-augmented generation for large language models: A survey, arXiv preprint arXiv:2312.10997 (2023).

[9] F. Pezzuti, et al., Cross-encoder re-ranking of ColBERTv2 on MS MARCO, arXiv preprint arXiv:2503.22672 (2025).

[10] Y. Gao, et al., Modular RAG: Transforming RAG systems into LEGO-like reconfigurable frameworks, arXiv preprint arXiv:2407.21059 (2024).

[11] R. Singh, et al., Agentic retrieval-augmented generation: A survey, arXiv preprint arXiv:2501.09136 (2025).

[12] NVIDIA, Finding the best chunking strategy for accurate AI responses, https://developer.nvidia.com/blog/finding-the-best-chunking-strategy-for-accurate-ai-responses/, 2024.

[13] Anthropic, Introducing contextual retrieval, Technical Blog Post, https://www.anthropic.com/engineering/contextual-retrieval, 2024.

[14] H. Trivedi, N. Balasubramanian, T. Khot, A. Sabharwal, Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023), 2023, pp. 10014–10037.

[15] T.-Y. Hsu, et al., DAT: Dynamic alpha tuning for hybrid retrieval in retrieval-augmented generation, arXiv preprint arXiv:2503.23013 (2025).
[16] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing reasoning and acting in language models, in: International Conference on Learning Representations (ICLR 2023), 2023.
[17] S.-Q. Yan, J.-C. Gu, Y. Zhu, Z.-H. Ling, Corrective retrieval augmented generation, arXiv preprint arXiv:2401.15884 (2024).
[18] J. Chen, H. Lin, X. Han, L. Sun, Benchmarking large language models in retrieval-augmented generation, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, 2024, pp. 17754–17762.
[19] Y. Lyu, et al., CRUD-RAG: A comprehensive Chinese benchmark for retrieval-augmented generation of large language models, ACM Transactions on Information Systems (2025).
[20] S. Es, J. James, L. Espinosa-Anke, S. Schockaert, RAGAS: Automated evaluation of retrieval augmented generation, in: Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2024), 2024.
[21] G. Guinet, B. Omidvar-Tehrani, A. Deoras, L. Callot, Automated evaluation of retrieval-augmented language models with task-specific exam generation, in: Proceedings of the 41st International Conference on Machine Learning (ICML 2024), 2024. ArXiv:2405.13622.