John Tusha, BSc

Optimising Retrieval-Augmented Generation for Historical Education: Technical Enhancements and User Evaluation of the ALiVE Chatbot

Master's Thesis
to achieve the university degree of
Master of Science
Master's degree programme: Software Engineering and Management

submitted to

Graz University of Technology

Supervisor
Gütl, Christian, Assoc.Prof. Dipl.-Ing. Dr.techn.

Institute of Human-Centred Computing
Head: Helic, Denis, Assoc.-Prof. Dipl.-Ing. Dr.techn.

Co-Supervisor
Steinmaurer, Alexander, Mag.rer.nat. Dr.rer.nat.

Vienna, September 2025

John Tusha, BSc

Optimierung von Retrieval-Augmented Generation für den Geschichtsunterricht: Technische Verbesserungen und Nutzerevaluation des ALiVE-Chatbots

Masterarbeit
zur Erlangung des akademischen Grades eines
Diplom-Ingenieur
Masterstudium: Software Engineering and Management

eingereicht an der

Technische Universität Graz

Betreuer
Gütl, Christian, Assoc.Prof. Dipl.-Ing. Dr.techn.

Institute of Human-Centred Computing
Vorstand: Helic, Denis, Assoc.-Prof. Dipl.-Ing. Dr.techn.

Mitbetreuer
Steinmaurer, Alexander, Mag.rer.nat. Dr.rer.nat.

Wien, September 2025

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Abstract

Retrieval-Augmented Generation (RAG) is a critical technique to mitigate hallucinations in Large Language Models (LLMs), yet its effectiveness hinges on a complex interplay of architectural choices. Yet, the optimal pipeline configuration remains an open challenge, as choices in chunking, retrieval, reranking, and generator models interact in complex ways to affect both quality and efficiency.

This thesis systematically evaluates twelve RAG pipeline configurations on a corpus of 10 historical English history documents, using 42 carefully curated questions of varying complexity (single-hop specific, multi-hop specific, multi-hop abstract). Evaluation with the RAGAS metric suite covers Faithfulness, Answer Relevancy, Context Precision, and Context Recall, complemented by efficiency measurements and statistical significance tests.

Core finding: factual grounding and perceived relevancy follow a Pareto frontier rather than converging on a single best configuration. The dense_semantic pipeline achieves the highest Faithfulness (0.513), ideal for accuracy-critical use cases, while hybrid_sentence_rerank maximizes Answer Relevancy (0.877) and Context Recall (0.548), favoring user experience. Efficiency analysis shows dense_semantic delivers the best "Faithfulness per second" ratio, with only a 9% cost increase over baseline.

Generator choice strongly influences final output quality. GPT-3.5 leads across all metrics (F: 0.652, AR: 0.921), Mistral 7B offers a strong open-source balance (F: 0.587, AR: 0.821), while small models like Phi-3, though efficient, often default to refusal when context is imperfect. Cross-encoder reranking consistently improves retrieval metrics (Context Precision: +8.4 %, Recall: +5.2%), especially for weaker base chunking strategies. Hybrid retrieval boosts recall but may introduce noise with already-coherent semantic chunks, highlighting the need for holistic design.

I propose an adaptive RAG architecture that routes queries to specialized pipelines based on question type, combining strengths of different configurations. A decision tree supports practitioners in aligning pipeline design with priorities for accuracy, relevancy, latency, and deployment constraints.

This work contributes empirical evidence and actionable design guidance for RAG systems, demonstrating that optimal configuration depends on aligning architectural trade-offs with application goals. The provided evaluation framework offers a foundation for future research into adaptive, context-aware retrieval-augmented generation.

Kurzfassung

Retrieval-Augmented Generation (RAG) ist eine zentrale Technik, um Halluzinationen großer Sprachmodelle (LLMs) zu mindern. Ihre Wirksamkeit hängt jedoch von einem komplexen Zusammenspiel architektonischer Entscheidungen ab. Die optimale Pipeline-Konfiguration bleibt offen, da sich Entscheidungen zu Chunking, Retrieval, Reranking und Generierungsmodell in vielschichtiger Weise auf Qualität und Effizienz auswirken.

Diese Arbeit evaluiert systematisch zwölf RAG-Pipeline-Konfigurationen auf einem Korpus aus 10 englischsprachigen historischen Dokumenten, gestützt auf 42 sorgfältig kuratierte Fragen unterschiedlicher Komplexität (Single-Hop spezifisch, Multi-Hop spezifisch, Multi-Hop abstrakt). Die Bewertung mit der RAGAS-Metriksuite umfasst Faithfulness (Faktentreue), Answer Relevancy (Antwortrelevanz), Context Precision und Context Recall; ergänzt um Effizienzmessungen und Tests auf statistische Signifikanz.

Zentrale Erkenntnis: Faktische Verankerung und wahrgenommene Relevanz liegen auf einer Pareto-Front, statt in einer einzigen „besten“ Konfiguration zu konvergieren. Die Pipeline dense_semantic erreicht die höchste Faithfulness (0,513) und eignet sich damit für genauigkeitskritische Anwendungsfälle, während hybrid_sentence_rerank die Answer Relevancy (0,877) und den Context Recall (0,548) maximiert und damit die Nutzererfahrung begünstigt. Die Effizienzanalyse zeigt, dass dense_semantic das beste Verhältnis „Faithfulness pro Sekunde“ liefert - bei nur 9% Mehrkosten gegenüber der Basislinie.

Die Wahl des Generierungsmodells beeinflusst die Ergebnisqualität stark: GPT-3.5 führt über alle Metriken (F: 0,652; AR: 0,921), Mistral 7B bietet als Open-Source-Modell ein ausgewogenes Profil (F: 0,587; AR: 0,821), während kleinere Modelle wie Phi-3 zwar effizient sind, bei unvollkommenem Kontext jedoch häufig zur Antwortverweigerung tendieren. Cross-Encoder-Reranking verbessert die Retrieval-Metriken konsistent (Context Precision: +8,4%, Recall: +5,2%), insbesondere bei schwächeren Chunking-Strategien. Hybrides Retrieval steigert den Recall, kann aber bei bereits kohärenten semantischen Chunks zusätzliches Rauschen einführen – ein Hinweis auf die Notwendigkeit ganzheitlicher Pipeline-Gestaltung.

Ich schlage eine adaptive RAG-Architektur vor, die Anfragen abhängig vom Fragetyp an spezialisierte Pipelines routet und so die Stärken verschiedener Konfigurationen kombiniert. Ein Entscheidungsbaum unterstützt Praktikerinnen und Praktiker dabei, Pipeline-Designs an Prioritäten wie Genauigkeit, Relevanz, Latenz und Deploymenteinschränkungen auszurichten.

Der Beitrag dieser Arbeit besteht in empirischer Evidenz und umsetzbaren

Gestaltungsleitlinien für RAG-Systeme. Er zeigt, dass optimale Konfigurationen davon abhängen, architektonische Trade-offs mit den Zielen der Anwendung in Einklang zu bringen. Das bereitgestellte Evaluierungsframework bildet eine Grundlage für zukünftige Forschung zu adaptiver, kontextsensitiver Retrieval-Augmented Generation.

Acknowledgment

Mein besonderer Dank gilt meinen Betreuern, Prof. Dr. Christian Gütl und Dr. Alexander Steinmaurer, die mir ermöglicht haben, meine Diplomarbeit am HCC zu verfassen. Ihre Unterstützung ermöglichte es mir, zwei meiner großen Interessen – Technik und Bildungswesen – im spannenden Feld des maschinellen Lernens miteinander zu verbinden.

Ebenso danke ich meinen Freunden für ihre stetige Unterstützung und Geduld während meines Studiums.

Contents

List of Figures

List of Tables

1. Introduction

> The great aim of education is not knowledge but action
>
> *Herbert Spencer*

Modern pedagogy is undergoing a profound shift, moving from a teacher-centered model of instruction toward personalized, student-centered learning environments (Bray & McClaskey, 2015). The ideal is an educational guide that adapts individually to the pace, prior knowledge, and interests of each learner. In practice, however, implementing this ideal often hits the limits of personnel resources, as a single teacher scarcely has the capacity to design and continuously support a fully individualized learning path for every student (Pane et al., 2015).

In this context, technologies of Artificial Intelligence (AI) and their application in education (AIED) are increasingly becoming the focus of scientific interest (F. Ouyang & Jiao, 2021). In particular, dialogue-based systems, known as chatbots, promise a potential solution here.

> Conversational agents have the potential to revolutionize education by providing scalable one-on-one tutoring for every student, which has long been the ideal in education but was unattainable in practice (Kuhail et al., 2022).

Through immediate feedback, availability independent of time and place, and adaptive dialogue management, these systems can offer a scalable form of personalized support (Winkler & Söllner, 2018). Studies already demonstrate positive effects from the use of chatbots for knowledge transfer, answering routine questions, and increasing learner motivation (Quiroga Perez et al., 2020).

However, the functionality of many current educational chatbots is often limited to simple question-and-answer mechanisms, while a deep grounding in didactic theories for fostering higher cognitive skills, such as critical thinking or problem-solving competence, is frequently lacking (Zawacki-Richter et al., 2019). Simultaneously, the use of such systems raises critical questions regarding the danger of misinformation through "hallucinations," the sensitive handling of student data, and the ethical implications of a partially automated pedagogy (Kasneci et al., 2023) (Montenegro-Rueda et al., 2023). These challenges must be addressed to responsibly unlock the full potential of AI for teaching and learning.

This master's thesis builds upon these insights and contributes to the further development of the ALiVE system (Artificial Legacy in Virtual Environments), a research platform for educational dialogue systems. Within this work, new components have been implemented to extend the system's capabilities: these include advanced chunking strategies for document processing, a hybrid retrieval approach combining semantic and lexical methods, and a reranking pipeline for improved answer selection. These technical adaptations serve the overarching goal of enabling constructivist, interactive learning dialogues that go beyond simple question answering. Furthermore, safeguards have been integrated to mitigate hallucinations and ensure pedagogical relevance. In doing so, this thesis bridges theoretical didactic concepts with practical, scalable AI tooling for future educational applications.

1.1. Aims and Objectives

The central aim of this thesis is to advance immersive history education through the ALiVE system (Artificial Legacy in Virtual Environments). ALiVE enables realistic, first-person dialogues with historical figures—featuring TTS voice responses, 3D avatar representations, and verifiable source citations. Its core technological innovation is a Retrieval-Augmented Generation (RAG) pipeline that combines engineered prompts, hybrid retrieval, and answer reranking to significantly curb hallucinations in LLM outputs (custom pipelines have proven effective in reducing fabricated responses)

By integrating multimodal feedback, this work transforms passive learning into an active, constructivist experience, encouraging learners to engage in questions, reflection, and historical thinking, as theorized by Chi and Wylie (2014) and Thorp and Persson (2020). Moreover, early prototypes, such as those applying LLM-driven agents in VR history scenarios, demonstrate that adaptive, immersive agents improve engagement, trust, and perceived expertise in learners

To meet these just ambitions, the thesis follows two phases: Development phase: Building upon recent best practices—advanced document chunking, hybrid semantic-lexical retrieval and reranking mechanisms —this phase constructs a robust, scalable version of ALiVE grounded in pedagogical theories and technical safeguards (e.g. citation enforcement).

Evaluation phase: Conducting an evaluation using technical metrics (precision, latency, hallucination rate) and qualitative assessments to capture overall system performance.

1.2. Structure of the Thesis

The structure of this thesis is divided into six main chapters, each addressing a key component of the research and development process. Chapter 2 provides the theoretical foundations by introducing core concepts in artificial intelligence in

education (AIED), immersive learning, and historical thinking. It also discusses relevant pedagogical frameworks such as constructivism and the ICAP model, which inform the design of the ALiVE system. It reviews related work, focusing on existing educational chatbots, retrieval-augmented generation (RAG) approaches, and prior efforts in virtual history agents. This chapter identifies current limitations and helps define the research gap addressed in this thesis.

Chapter 3 deals with the functional and non-functional requirements of the extensions of the Alive-System.

Chapter 4 presents the system design and implementation of ALiVE in detail. It describes the modular architecture and the technologies involved, including advanced text chunking strategies, hybrid retrieval methods, reranking pipelines and prompt engineering.

Chapter 5 is dedicated to the evaluation of the system, combining a technical performance analysis—measuring Faithfulness, Answer Relevancy, Context Precision, and Context Recall—with additional metric-based comparisons to provide a comprehensive view of performance.

Chapter 6 describes the key lessons learned from the literature review, the development of the ALiVE system, and the empirical evaluation, highlighting how modular RAG design choices and educational goals were aligned to balance answer quality, transparency, and immersion. It also reflects on practical insights gained during implementation, such as the value of reproducible experiments, stable evaluation baselines, and clear communication of system capabilities and limitations.

Finally, Chapter 7 describes the conclusions of this thesis, summarizing the design, implementation, and evaluation of ALiVE as a modular RAG system for history education, along with its key technical and empirical findings. It also outlines a concrete roadmap for future work, prioritizing retrieval and indexing improvements, adaptive query handling, enhanced grounding and evaluation methods, and pedagogical features to make the system more accurate, adaptive, and classroom-ready.

2. Background and Related Work

This chapter lays the theoretical and historical groundwork necessary to contextualize the development of a pedagogical chatbot. It begins by tracing the evolution of chatbot technology from its earliest beginnings to the current era of large language models (LLMs). Subsequently, it explores learning theories relevant to the design of effective educational tools and reviews prior work on the application of conversational AI in learning environments, highlighting current trends, empirical insights, and open research challenges.

2.1. Background

2.1.1. History of Chatbots

The history of chatbots is intrinsically linked to the broader quest for artificial intelligence, famously catalyzed by Alan Turing's proposition of the "imitation game"—now known as the Turing Test (Turing, 1950). This conceptual foundation sparked early attempts to simulate human conversation through programmed dialogue systems.

One of the first such systems was ELIZA, developed in the mid-1960s by Joseph Weizenbaum at the MIT Artificial Intelligence Laboratory. ELIZA operated by detecting keywords in the user's input and responding with pre-defined, templated phrases. Its most famous script mimicked a Rogerian psychotherapist, prompting users to reflect on their statements. Despite its simplicity, ELIZA revealed the ELIZA effect—the human tendency to anthropomorphize even limited conversational systems (Weizenbaum, 1966).

Shortly thereafter, PARRY (1972), developed by Colby (1981), simulated the conversational patterns of a person with paranoid schizophrenia. Unlike ELIZA, PARRY incorporated rudimentary modeling of beliefs and emotions. In experiments, even trained psychiatrists had difficulty distinguishing it from real patients.

During the 1990s and early 2000s, systems like A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) introduced more structured rule-based approaches to dialogue via AIML (Artificial Intelligence Markup Language), winning multiple Loebner Prizes. These systems still lacked contextual awareness and semantic understanding but helped keep interest in chatbot development alive (Wallace, 2009).

The 2010s ushered in the commercial era of chatbots. Virtual assistants like

Siri[1], Google Now[2], Cortana[3], and Alexa[4] brought conversational interfaces to everyday users. These assistants combined voice recognition, machine learning, and web-based integration but were still limited in open-ended dialogue and relied on structured intents.

A radical shift occurred with the development of the Transformer architecture (Vaswani et al., 2017), which enabled a new generation of large-scale language models. OpenAI's GPT series, particularly GPT-2 (2019)[5] and GPT-3 (2020)[6], demonstrated unprecedented abilities in generating coherent, context-sensitive text based on few-shot input (Brown et al., 2020). These models are pre-trained on massive textual datasets and can generalize across domains without task-specific fine-tuning, making them suitable for complex and nuanced interactions.

The capabilities of LLMs have opened up new avenues for educational use cases, including tutoring, knowledge reinforcement, and interactive storytelling. Unlike their rule-based predecessors, modern chatbots are now capable of simulating expert roles, adapting language to learners' levels, and even generating custom content on demand. This evolution sets the stage for their integration into learning environments, especially in domains such as history education, where nuanced reasoning, narrative engagement, and contextual awareness are critical.

2.1.2. The Rise of Generative AI Chatbots

While the 2010s familiarized the public with conversational AI, the launch of OpenAI's ChatGPT on November 30, 2022, marked a watershed moment. Based on the GPT-3.5 architecture, ChatGPT was made available to the public through a simple web interface, leading to an unprecedented global hype. It reached 100 million active users within just two months, a growth rate faster than any previous consumer application.[7] The sudden, widespread access to a highly capable generative model triggered intense public and academic discourse about the future of work, creativity, and education.

Unlike their predecessors, which were often task-oriented (e.g., "set a timer" or "what's the weather?"), generative AI chatbots are built on Large Language Models (LLMs). These models do not rely on pre-programmed scripts. Instead, they function by calculating the probability of the next most likely word (or token) in a sequence, based on the patterns learned from their vast training data—often a

[1] https://www.apple.com/de/siri/, letzter Zugriff am 14.07.2025

[2] https://de.wikipedia.org/wiki/Google_Now, letzter Zugriff am 14.07.2025

[3] https://support.microsoft.com/en-us/topic/end-of-support-for-cortana, letzter Zugriff am 14.07.2025

[4] https://de.wikipedia.org/wiki/Amazon_Alexa, letzter Zugriff am 13.07.2025

[5] https://katzlberger.ai/2019/07/15/gpt-2/, letzter Zugriff am 12.07.2025

[6] https://katzlberger.ai/2020/11/04/die-erste-allgemeine-kuenstliche-intelligenz/, letzter Zugriff am 11.07.2025

[7] https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01 letzter Zugriff am 05.07.2025

significant portion of the public internet and licensed datasets (Zhao et al., 2023). The core technology, the Transformer architecture, allows the model to weigh the importance of different words in the input text (a mechanism called "attention"), enabling it to grasp context, nuance, and complex relationships in language. The process is further refined through techniques like Reinforcement Learning from Human Feedback (RLHF), where human raters guide the model towards more helpful, harmless, and honest responses (L. Ouyang et al., 2022).

The release of ChatGPT was followed by a rapid succession of competing models from other major tech companies, each aiming to advance the state of the art. This has led to a diverse ecosystem of models with different strengths and characteristics, as can be seen in Table 2.1.

Model	**Developer**	**Underlying Architecture**	**Key Characteristics**
GPT-3.5 / GPT-4	OpenAI	Generative Pre-trained Transformer (GPT)	General reasoning, coding, RLHF, multimodal (GPT-4)
LaMDA / PaLM 2	Google	LaMDA / Pathways Language Model	Dialogue-optimized, broad reasoning, factuality focus
LLaMA / Llama 2	Meta AI	Llama (Large Language Model Meta AI)	Research-focused, open-source, community-driven
Claude 2 / 3	Anthropic	Constitutional AI / Claude	AI safety, ethics-focused, 'Constitutional AI'

Table 2.1.: Comparison of Major Generative AI Models (Concise).

This rapid evolution presents both immense opportunities and significant challenges for the field of education, requiring a careful examination of how these

powerful tools can be harnessed effectively and responsibly.

2.1.2.1. Core Capabilities and Application Domains

The versatility of LLMs stems from a set of core emergent abilities derived from their training on diverse data. These foundational skills include text summarization, translation, sentiment analysis, and, most notably, coherent text generation, which allows them to engage in complex problem-solving and creative tasks (Qiu et al., 2020). These capabilities have unlocked applications across numerous professional domains.

In software engineering, for instance, LLMs are used as coding assistants that can generate boilerplate code, debug existing programs, translate code between languages, and explain complex algorithms in natural language, thereby accelerating development cycles (Fan et al., 2023). The healthcare sector is another area of significant impact, where these models assist with summarizing clinical notes, drafting patient communications, and providing clinicians with up-to-date information from medical literature to support diagnostic processes (Thirunavukarasu et al., 2023). Businesses have also widely adopted this technology, primarily to enhance customer service operations through 24/7 automated support, but also for internal tasks like generating marketing copy and analyzing customer feedback (Goot et al., 2021).

While the applications in these fields are transformative, the domain most pertinent to this thesis is education, where the core capabilities of LLMs present a unique set of opportunities and challenges that will be explored in subsequent sections.

2.1.2.2. Conversational AI in Educational Settings

While the idea of using conversational AI in education is not new, the power of modern LLMs has dramatically raised the stakes and the potential. At its heart, the technology promises to tackle one of education's oldest and most difficult challenges: giving each student truly personal instruction. This goal is famously captured in Bloom's "2 sigma problem", a study which showed that students with a personal human tutor performed two standard deviations better than their peers in a traditional classroom (BLOOM, 1984). For years, AI-powered chatbots have been viewed as a way to finally scale the benefits of one-on-one tutoring to every learner, potentially making education more equitable.

Looking at the research, these AI chatbots are envisioned in several distinct pedagogical roles. A common application is the Intelligent Tutoring System (ITS), where the bot acts as a patient guide, offering hints and immediate feedback for structured problems in subjects like math or programming (Okonkwo & Ade-Ibijola, 2021). A more sophisticated and challenging role is that of a Socratic Opponent. Instead of handing out answers, this type of bot asks probing questions, forcing

students to defend their reasoning, uncover flaws in their thinking, and build a much deeper understanding of a topic on their own (Gregorcic et al., 2024).

Beyond tutoring, chatbots also serve as tireless language learning partners, giving students a safe space to practice conversation, try out role-playing, and get instant corrections without fear of embarrassment (Huang et al., 2022). A particularly clever application is the teachable agent, which flips the script entirely. Here, students learn by teaching the chatbot, a method that leverages the "protégé effect." The simple act of organizing and explaining concepts to another "entity" compels students to solidify their own knowledge, often leading to better learning outcomes (Chase et al., 2009). Already 70% of schools in Germany use artificial intelligence in some form, and one in ten schools plans to integrate such technologies in the near future, as illustrated in Figure 2.1.[8]

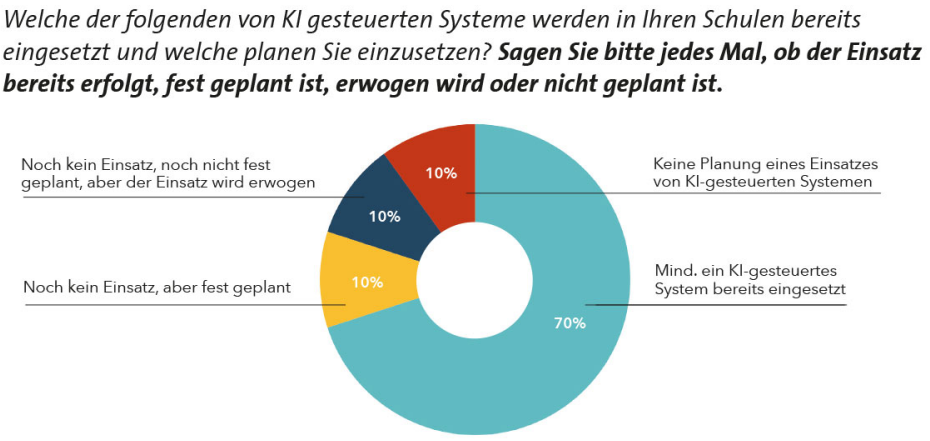

Figure 2.1.: AI on the rise in schools? [8]

The upsides of bringing these systems into the classroom are clear. They are available 24/7, support learning at an individual's own pace, and offer a non-judgmental space that can lower the anxiety some students feel when asking questions in public (Zawacki-Richter et al., 2019). But this potential is matched by considerable risks. A key concern is that students might become too reliant on AI for easy answers, which could hinder their ability to think critically. There's also the danger of "hallucinations"—plausible but false information generated by the LLM—which threatens the integrity of what is being learned (Kasneci et al., 2023). On top of these functional problems, serious ethical questions about data privacy and algorithmic bias remain, as a system trained on biased data could easily perpetuate societal inequalities (Williamson & Eynon, 2020). Therefore, successfully using these powerful tools in education demands a framework that is not only pedagogically sound but also actively designed to mitigate these inherent risks.

[8]https://www.rednet.ag/studie_ki-in-schulen.html, letzter Zugriff am 13.06.2025

2.1.3. Learning Theories and the AI Revolution

To understand the true potential of AI in education, one has to look beyond the technology itself and consider how people learn. Modern pedagogy has largely moved away from simple behaviorism towards constructivism, a theory which posits that learners actively construct their own knowledge rather than passively receiving it (Piaget, 1971). An extension of this, social constructivism, emphasizes that this process is deeply collaborative and shaped by social interaction and dialogue (Vygotsky, 1978). These theories suggest that effective learning tools should not be mere information dispensers but facilitators of exploration, dialogue, and knowledge creation.

Historically, education has always been shaped by technological revolutions. The invention of the printing press democratized access to information that was once the exclusive domain of the clergy and nobility, fundamentally changing the structure of knowledge dissemination (Eisenstein, 1980). Centuries later, the arrival of the internet and personal computers triggered another seismic shift, offering instant access to a global repository of information and enabling new forms of remote and asynchronous learning (Harasim, 2000). Each of these revolutions forced educators to rethink their methods and the very definition of a classroom.

The current wave of generative AI can be seen as the third major technological revolution for education. It is distinct from the previous two. While the printing press distributed static information and the internet provided access to it, AI offers the potential for dynamic, interactive engagement with information at an unprecedented scale. An LLM-powered chatbot, if designed correctly, can be a powerful tool for constructivist learning. It can act as a Vygotsky dialogue partner, scaffolding a student's learning by providing tailored support that helps them bridge the gap between their current ability and their potential. It can create personalized learning journeys that encourage exploration and discovery, moving far beyond the one-size-fits-all model of previous technologies. This makes the thoughtful integration of AI not just a technological upgrade, but a pedagogical imperative to finally realize the long-held goals of truly personalized and constructive learning.

2.1.4. Large Language Models

At their core, Large Language Models are massive neural networks, distinguished by the sheer scale of their architecture and the data they are trained on. Their fundamental task is surprisingly simple: predicting the next word in a sequence. By processing trillions of words from books, articles, and websites, these models learn intricate statistical patterns of human language. The complexity of these models is often measured in "parameters," with leading models containing hundreds of billions of them. A fascinating outcome of this immense scale is the appearance of "emergent abilities"—complex skills like translation, summarization, and even logical reasoning that were not explicitly programmed but arise from the core prediction task (Wei, Tay, et al., 2022). Understanding this foundation is key to

grasping both their power and their limitations.

2.1.4.1. How LLM work

To understand how an LLM generates text, it's helpful to break the process down into a few key steps. At a high level, the model takes a sequence of text as input and calculates the most probable next piece of text. This process, while computationally immense, can be illustrated with a simple example.

First, the input text is broken down into smaller units called tokens. Tokens can be words, parts of words, or even just punctuation. This process, known as tokenization, allows the model to handle a vast vocabulary and grammatical structures efficiently (Kudo & Richardson, 2018).

Table 2.2.: Tokenizing Example.

Originaltext	The student opened their
Tokens	`[The]` `[student]` `[opened]` `[their]`

Next, each token is converted into a numerical vector, a process called embedding. This vector represents the token's meaning in a high-dimensional space, where similar words have similar vectors. The model then processes these embeddings using its internal layers, most importantly the attention mechanism. Attention allows the model to weigh the importance of different tokens in the input sequence when predicting the next token. For example, in the sentence above, the model would pay high attention to "student" and "opened" to determine that a noun related to learning is likely to follow (Vaswani et al., 2017).

Finally, the model generates a probability distribution over its entire vocabulary for the next token. The core of its function can be simplified to the formula seen in Equation 2.1, which aims to maximize the probability of the next word $(w_1, \ldots, w_{i-1})$:

$$P(w_i|w_1, ..., w_{i-1}) \tag{2.1}$$

In our example, the model would calculate the probabilities for all possible next tokens:

- `P(``book'' | ``The'', ``student'', ``opened'', ``their'')` → high probability (e.g. 45%)
- `P(``laptop'' | ``The'', ``student'', ``opened'', ``their'')` → medium probability (e.g. 30%)
- `P(``mind'' | ``The'', ``student'', ``opened'', ``their'')` → low probability (e.g. 15%)
- `P(``car'' | ``The'', ``student'', ``opened'', ``their'')` → very low probability (e.g. 0.01%)

The model then selects a token from this distribution (not always the most probable one, to allow for creativity) and appends it to the input sequence. This entire process is then repeated, with the newly generated word becoming part of the input for predicting the next one, allowing the LLM to generate entire sentences, paragraphs, and documents one token at a time.

2.1.4.2. Training of Large Language Models

Creating a capable LLM is a multi-stage process. It's not a single event, but a carefully designed pipeline that first builds a general understanding of language and then steers the model's behavior to be helpful and safe for users. This pipeline typically involves three main phases as presented in Table 2.3:

Table 2.3.: Phases of LLM-Training

Stage	**Goal**	**Data Used**
1. Pre-training	Foundational knowledge of language & world	Massive, diverse corpus
2. SFT	Ability to follow instructions, be helpful	High-quality prompt-response
3. Alignment	Refine behavior for safety and helpfulness	Human preference rankings

Stage 1: **Pre-training.** This is the most resource-intensive phase. Here, the model is exposed to a massive dataset—trillions of words from the internet, books, and other sources—without any explicit instructions. Its only job is to learn to predict the next word in a sentence (Radford & Narasimhan, 2018). By performing this task over and over, the model implicitly learns grammar, facts about the world, reasoning patterns, and unfortunately, also the biases present in its training data.

Stage 2: **Supervised Fine-Tuning (SFT).** A pre-trained model is like a vast library of knowledge, but it doesn't know how to be a helpful assistant. The SFT phase teaches it this skill. The model is trained on a much smaller, carefully curated dataset of high-quality examples. Each example consists of an instruction (a "prompt") and an ideal response written by a human. This process, also known as instruction tuning, teaches the model to respond to user requests in a useful and conversational manner (Wei, Bosma, et al., 2022).

Stage 3: **Alignment Tuning.** The final, crucial phase is about aligning the model's behavior with human values, making it more helpful, honest, and harmless. The most prominent technique for this is Reinforcement Learning from Human Feedback (RLHF) (L. Ouyang et al., 2022). In this stage, humans don't write the ideal answers. Instead, they rank several of the model's generated responses to a prompt from best to worst. A separate "reward model" is then trained on these human preference rankings. This reward model learns to predict what kind of answer a human would find good or

bad (Christiano et al., 2023). Finally, the LLM is further fine-tuned, using the reward model as a guide. The LLM essentially learns to generate answers that it predicts will receive a high score from the human-aligned reward model, steering its behavior towards more desirable outcomes (Schulman et al., 2017).

2.1.5. Challenges and Limitations of Large Language Models

Despite their impressive capabilities, LLMs are fraught with inherent technical and ethical challenges that must be addressed for their responsible deployment, especially in sensitive areas like education. These issues are not merely bugs to be fixed but are often deep-seated consequences of how these models are designed and trained.

2.1.5.1. Technical Challenges of Large Language Models

A primary and widely discussed issue is hallucination, where a model generates text that is fluent and plausible but factually incorrect or nonsensical (Ji et al., 2023). Since LLMs are probabilistic sequence predictors, not databases, they do not possess a true concept of "truth." They simply generate what is statistically likely, which can lead to the confident assertion of falsehoods. Another significant hurdle is their lack of robustness. Minor, often imperceptible changes to an input prompt can cause the model to produce wildly different outputs, making their behavior unpredictable (Wang et al., 2024). Furthermore, their knowledge is static, limited to the point in time when their training data was collected. This "knowledge cutoff" means they are unaware of events that have occurred since their training, rendering them unreliable for topics that evolve quickly.

LLMs, such as those powering ChatGPT or Gemini, face several technical hurdles that limit their reliability in educational settings. One primary challenge is their context window limitation, which restricts the amount of text an LLM can process at once. For example, models like GPT-3 are constrained to a few thousand tokens, leading to potential loss of context in extended interactions, such as complex academic discussions or long-form assignments. This can result in responses that misinterpret earlier parts of a conversation or omit critical details, reducing their utility for tasks requiring sustained coherence.[9]

As Professor Anima Anandkumar from Caltech notes, the quality of training data directly impacts model performance, and unverified or noisy data can lead to outputs that lack factual grounding. This is particularly problematic in education, where students rely on accurate information to develop knowledge. For instance, if an LLM generates a summary based on flawed or incomplete data, it risks misleading learners and eroding trust in educational tools.[10]

[9]https://nexla.com/ai-infrastructure/llm-hallucination/, letzter Zugriff am 21.06.25

[10]https://www.nytimes.com/2024/12/23/science/ai-hallucinations-science.html/, letzter Zu-

2.1.5.2. Types of Hallucination of Large Language Models

Hallucinations—where LLMs generate plausible but incorrect or fabricated content—are a critical concern, especially in education, where accuracy is paramount. Recent research categorizes hallucinations into several types, each with implications for academic reliability:

- **Factual Inconsistency**: Large language models may generate statements that contradict verified facts, such as asserting that the Wright brothers invented the internet, an obvious error. Such inaccuracies can confuse students and erode confidence in AI-supported learning tools.[11]
- **Factual Fabrication**: LLMs can produce entirely invented narratives or entities, like describing a nonexistent historical event or citing a fabricated academic study. Research indicates that 47% of ChatGPT's cited references are fictitious, risking academic integrity by leading students to reference nonexistent sources, as noted by journalist Benj Edwards.[12]
- **Input-Conflicting Hallucination**: This occurs when an LLM's output fails to align with the provided input, such as including extraneous details in a text summary that distort the original content. This issue is particularly detrimental in tasks like summarization or translation, where accuracy to the source is essential (Sun et al., 2024).
- **Context-Conflicting Hallucination**: LLMs may produce contradictory statements within a single interaction or across responses, with studies reporting a 14.3% contradiction rate in ChatGPT outputs, such as stating conflicting facts in successive answers. This can disrupt educational workflows and confuse learners.[13]
- **Semantic Hallucination**: These hallucinations involve coherent but semantically incorrect or misleading outputs, often due to misinterpreting nuanced language like sarcasm or cultural references. Dr. Emily Bender, a linguistics professor at the University of Washington, highlights that LLMs struggle with such subtleties, risking misinterpretations in educational settings where understanding intent is critical.[14]

To mitigate hallucinations in large language models (as illustrated in Figure 2.2.[15] several strategies can be employed. First, pre-processing and input control involve limiting input/output lengths and using curated prompts or style options to guide the model, reducing irrelevant or incorrect outputs by emphasizing conciseness

griff am 01.07.25

[11]https://medium.com/accredian/understanding-hallucinations-in-large-language-models-causes-and-solutions, letzter Zugriff am 17.06.2025

[12]https://arstechnica.com/information-technology/2023/04/why-ai-chatbots-are-the-ultimate-bs-machines-and, letzter Zugriff am 18.06.2025

[13]https://nexla.com/ai-infrastructure/llm-hallucination/, letzter Zugriff am 22.06.25

[14]https://nexla.com/ai-infrastructure/llm-hallucination/, letzter Zugriff am 22.06.25

[15]https://nexla.com/ai-infrastructure/llm-hallucination/, letzter Zugriff am 13.06.2025

through fine-tuning and few-shot prompting. Second, adjusting model configuration enhances output quality by tuning parameters like temperature for predictability, frequency and presence penalties for diversity, and top-p for balanced relevance, while a moderation layer ensures safe and appropriate responses. Third, monitoring and improvement through active learning, user feedback, rigorous testing, human verification, and domain-specific enhancements refines model performance, with tools like Nexla's platform simplifying custom data integration and feedback automation. Finally, enhancing context in production provides real-time access to external data sources and enriches user prompts with clear instructions, enabling more accurate and contextually relevant outputs. These combined approaches minimize hallucinations, ensuring reliable and ethically sound LLM performance in educational and other applications.

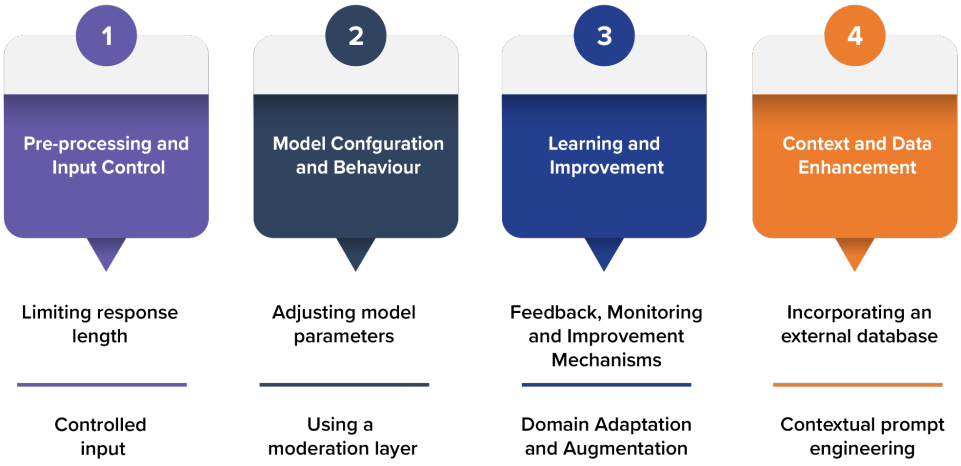

Figure 2.2.: Mitigating hallucination in LLMs [15]

2.1.5.3. Social and Ethical Challenges of Large Language Models

Perhaps more concerning are the profound social and ethical issues. LLMs are trained on vast swathes of text from the internet, which inevitably contains and reflects human biases. Consequently, models can learn and amplify harmful stereotypes related to gender, race, and culture (Bender et al., 2021). This poses a significant risk of perpetuating societal inequalities. The potential for malicious use is another major concern. These models can be weaponized to generate convincing misinformation, propaganda, or personalized phishing emails at an unprecedented scale, threatening democratic processes and individual security (Weidinger et al., 2022).

While Large Language Models present exciting opportunities for advancing educational tools, their integration into learning environments must be critically examined in light of key ethical challenges—particularly in the areas of privacy, bias, surveillance, and autonomy. LLMs often rely on extensive user data and

interaction histories, raising serious privacy concerns, especially when learners are unaware of the extent to which their data is collected, processed, or shared. Simultaneously, these systems are prone to reproducing biases embedded in the training data, which can manifest in gendered or racialized responses and thus perpetuate structural inequalities. Furthermore, the use of AI-driven educational tools increasingly involves surveillance mechanisms that monitor and predict students' behavior, engagement, or performance—sometimes without clear boundaries or informed consent. Such predictive systems, while aimed at personalization, may ultimately restrict students' autonomy, as learning paths and assessments are shaped by opaque algorithmic decisions. As Akgun (2021) emphasizes, these risks are not merely technical, but deeply social and political, and must be addressed through transparent design, inclusive development, and ongoing critical reflection within educational contexts. A comprehensive overview potential ethical and social risks can be seen in Figure 2.3.[16]

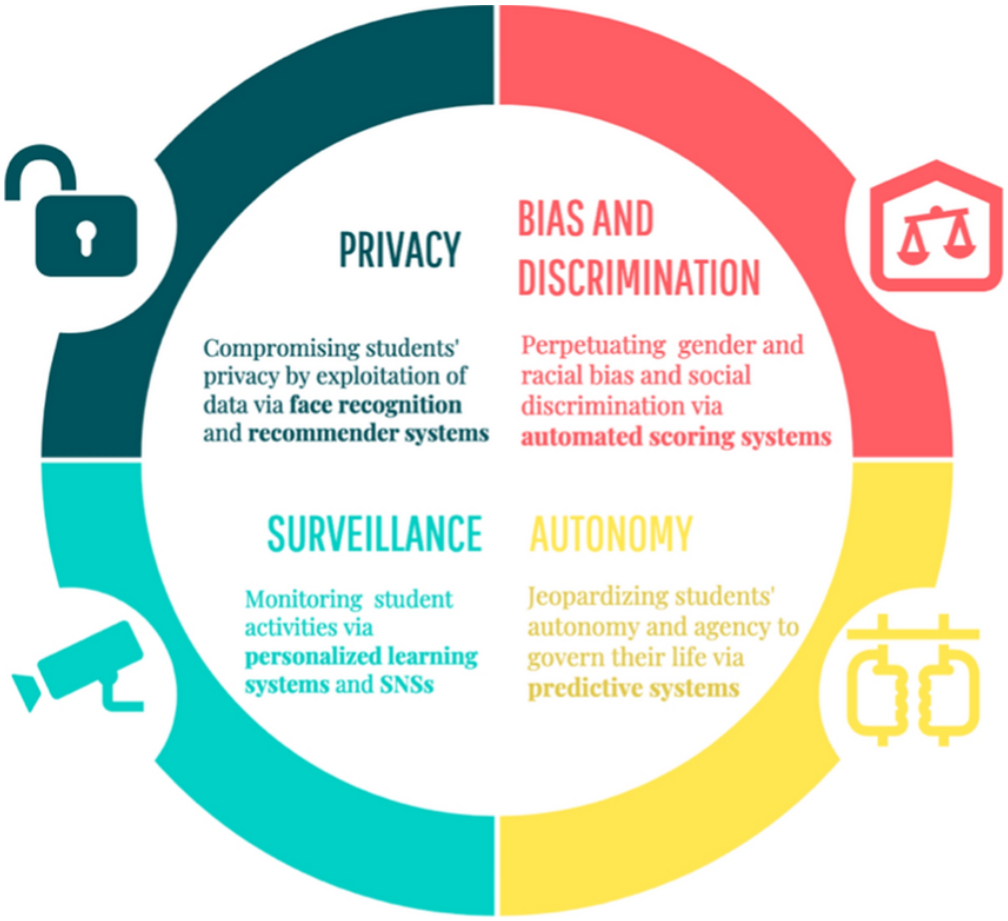

Figure 2.3.: Potential ethical and societal risks of AI applications in education [16]

[16]https://www.researchgate.net/figure/Potential-ethical-and-societal-risks-of-AI-applications-in-education_fig1_354759795, letzter Zugriff am 11.06.2025

2.1.5.4. Ethical and Environmental Impacts of AI in Education

Some education traditionalists, like University Professor Konrad Paul Liessmann, champion the use of classic teaching methods and raise concerns about the risks posed by modern technologies. They stress the importance of fostering independent study and sharp critical thinking, which they fear could be eroded by over-reliance on tech, especially AI-driven text tools. These experts warn that the rapid production of polished, formulaic, and politically sensitive content by such tools makes it increasingly difficult for teachers to tell human work apart from machine output. This problem was swiftly addressed in New York schools with a ban on these technologies, yet such cautions are rarely heard locally. Furthermore, tools like ChatGPT are criticized for lacking deep knowledge or real-time web access, merely churning out predictable phrases. Liessmann pushes for a revival of unaided, handwritten assignments and thorough oral assessments to counter these issues.[17]

Certain academics and journalists have voiced apprehension about the integration of artificial intelligence (AI) in educational settings, highlighting potential threats to fostering critical thinking and independent learning. For instance, Professor Ulrike Lucke, an informatics expert from the University of Potsdam, has expressed concerns that over-reliance on AI tools like large language models could diminish students' ability to engage deeply with complex problems. She argues that these tools, while capable of generating quick and polished outputs, may encourage superficial learning and undermine the development of analytical skills essential for academic growth. Lucke advocates for assessment methods that prioritize human reasoning, such as in-person discussions or project-based evaluations, to ensure students develop authentic intellectual capabilities (Helm et al., 2024).

Similarly, linguist and education scholar Baron (2023) warns that the proliferation of AI-generated content risks eroding the very authenticity of student work. In her analyses, she points out that while tools like ChatGPT can produce coherent essays, they often lack originality or critical depth, making it challenging for educators to assess true student understanding. Baron emphasizes the need for educational institutions to adapt by focusing on teaching methodologies that cultivate independent thought and ethical decision-making, rather than allowing an over-reliance on AI to devalue the learning process.

Echoing these concerns, Dr. Sabine Seufert, a professor at the University of St. Gallen specializing in educational technology, highlights the ethical implications of AI in schools. She cautions that excessive use of AI tools could lead to a "deskilling" effect, where students become overly dependent on technology and lose the ability to tackle challenges without digital assistance. Seufert calls for a balanced approach, advocating for clear guidelines on AI use and a renewed emphasis on fostering critical and creative thinking through traditional methods like handwritten assignments and reflective discussions (Sabitzer et al., 2024).

[17] https://www.kleinezeitung.at/meinung/6238131/Konrad-Paul-Liessmann_Warum-wir-in-den-Schulen-zu-Tafel-und-Kreide/, letzter Zugriff am 03.06.25

Furthermore, the immense computational resources required to train these models raise serious environmental concerns due to their significant carbon footprint (Strubell et al., 2019). Finally, issues of data privacy are paramount, as models can inadvertently memorize and reveal sensitive personal information that was part of their training data, creating substantial privacy risks for individuals (Carlini et al., 2021). These multifaceted challenges necessitate a cautious and critical approach to the integration of LLMs into society.

2.1.6. Eyewitness Mode and Narrative Personas

One of the distinguishing features explored in this work is the so-called *Eyewitness Mode*, a mechanism that enables generated responses to be presented in the first-person voice of historical characters. Instead of receiving neutral third-person answers, users interact with a persona that responds as if it had witnessed historical events firsthand. This approach aims to increase user immersion and engagement, particularly in educational settings.

Technical Foundation. The implementation of the Eyewitness Mode relies on the use of prompt templates that inject narrative perspective into the generation pipeline. These templates are dynamically constructed based on the user query and the selected historical figure. A simplified example is shown below:

> *You are Gustav Klimt. The user will ask questions about your life and the time you lived in. Please respond in the first person using historical facts. If you are unsure, say so. Do not invent information. Always base your answer on the sources provided.*

This template is prepended to the system prompt and remains constant throughout the interaction. Source documents retrieved via the RAG pipeline are inserted as context in a separate section of the prompt, allowing the model to stay grounded while adopting the target persona. The Eyewitness Mode is compatible with all retrieval and reranking components and requires no architectural changes, making it a lightweight but powerful enhancement.

Educational Rationale. From a pedagogical perspective, narrative-based learning has been shown to improve memory retention, foster empathy, and encourage deeper engagement with historical content (Green & Brock, 2000). When learners receive answers "spoken" by historical figures, they are more likely to perceive the material as vivid and emotionally resonant. This aligns with constructivist learning theories, which emphasize the value of situated and experiential knowledge acquisition.(Vygotsky, 1978).

Moreover, the Eyewitness Mode can be particularly beneficial in classroom discussions or group activities. Students may be asked to critically assess the

perspective expressed by the persona: “What might Gustav Klimt leave out?” or “How would a contemporary critic describe this event differently?” This opens the door to critical thinking about bias, historical context, and the construction of narrative—core competencies in history education.(Körber, 2011).

Limitations and Risks. While the Eyewitness Mode can enhance immersion, it also introduces a new layer of interpretive risk. When a language model adopts a historical persona, users may perceive its statements as more trustworthy or authoritative than they actually are. If the generated content contains hallucinations or inaccuracies, these may be amplified by the narrative framing. Prior studies have cautioned that personified outputs from AI systems can lead to an “illusion of understanding” and reduce critical scrutiny (Weidinger et al., 2022).

This underscores the importance of combining persona-based responses with visible source attribution and feedback mechanisms, as implemented in this thesis, to ensure transparency and foster critical engagement rather than passive acceptance.

2.1.7. User Feedback in Educational Chatbots

User feedback mechanisms play an important role in the design and evaluation of AI-based educational tools. They allow systems to capture human judgments about the quality, helpfulness, or clarity of generated content. In the context of conversational agents, feedback is often collected through lightweight interfaces such as rating buttons, binary choices (e.g., thumbs up/down), or short text fields.

From a pedagogical perspective, prompting learners to evaluate system outputs can support metacognitive processes and deepen engagement with the material. Rather than passively accepting answers, users are encouraged to reflect on their adequacy, correctness, and alignment with expectations or prior knowledge. This aligns with theories of active learning and formative assessment.

However, prior research highlights a gap between user satisfaction and factual correctness. Studies show that users tend to favor fluent or confidently phrased responses, even when they are inaccurate (M. Chen et al., 2021). This underscores the importance of designing feedback channels that support not only usability but also critical awareness.

2.1.8. Retrieval-Augmented Generation (RAG)

To address some of the critical limitations of LLMs, such as hallucination and static knowledge, a powerful architectural pattern called Retrieval-Augmented Generation (RAG) has gained prominence. First proposed by Lewis et al. (2021), RAG enhances a standard LLM by connecting it to an external, up-to-date knowledge source. Instead of relying solely on the information encoded in its parameters during training, the model can access and ground its responses in reliable, external data at the time of inference.

The RAG process typically involves three main steps:

1. **Retrieve:** When a user submits a prompt, the system first uses the query to search a specialized knowledge base—often a **vector database**. This database stores text chunks (from documents, articles, etc.) as numerical embeddings. The system retrieves the chunks whose embeddings are most semantically similar to the user's query embedding.
2. **Augment:** The retrieved text chunks are then combined with the original user prompt. This creates a new, "augmented" prompt that provides the LLM with relevant, factual context.
3. **Generate:** Finally, this augmented prompt is sent to the LLM. The model is instructed to generate an answer based *specifically* on the provided context. This grounds the response in the retrieved facts, significantly reducing the likelihood of hallucination and allowing the system to use information that is more current than its own training data (Y. Gao et al., 2024).

By separating the knowledge source from the language generation model, RAG offers a more modular, transparent, and updatable approach. It allows developers to control the information the model uses, making it a crucial technique for building more reliable and trustworthy AI applications, particularly in domains like education where factual accuracy is paramount (Ram et al., 2023).

2.1.9. Embedding Models and Vector Store Management

Semantic retrieval in modern natural language processing systems heavily relies on embedding models and vector databases. Embedding models map textual input into high-dimensional continuous vector spaces such that semantically similar inputs lie close together. This transformation is essential for enabling similarity-based retrieval, particularly in retrieval-augmented generation (RAG) pipelines, question answering systems, and conversational agents.

Let $q \in \mathbb{R}^d$ denote the embedding of a user query, and let $D = \{d_1, d_2, \ldots, d_n\} \subset \mathbb{R}^d$ represent the embeddings of a document corpus. To find the most relevant documents, the system computes the similarity between q and each d_i using a distance metric such as cosine similarity:

$$\text{sim}(q, d_i) = \frac{q \cdot d_i}{\|q\| \cdot \|d_i\|}$$

The top-k documents with the highest similarity scores are then selected as retrieval context. This process is made efficient through the use of *vector stores* or *approximate nearest neighbor (ANN)* search libraries such as FAISS, Annoy, or Weaviate, which allow sublinear-time retrieval from millions of vectors. A comprehensive example of similiarity-based retrieval can be found in Table 2.4.

Embedding models can be either general-purpose (e.g., `Sentence-BERT`, OpenAI Ada) or domain-specific (e.g., fine-tuned for legal, historical, or medical corpora).

Query	Document Chunk	Cosine Similarity
Who was Marie Curie?	Marie Curie was a physicist...	0.94
Who was Marie Curie?	Curium is a radioactive element...	0.76
Who was Marie Curie?	Einstein developed relativity...	0.33

Table 2.4.: Example of similarity-based retrieval using embeddings

The choice of embedding model significantly affects retrieval quality. For instance, Reimers and Gurevych (2019) showed that sentence embeddings from fine-tuned BERT networks outperform traditional averaging or TF-IDF approaches in semantic tasks.

Moreover, the configuration of the vector store—including dimensionality, indexing strategy, and search algorithm—can influence both latency and retrieval quality. Systems targeting real-time interaction must carefully balance accuracy and performance when choosing between exact and approximate nearest neighbor retrieval.

In summary, embedding-based retrieval provides the semantic backbone for linking user queries with relevant knowledge sources. Ongoing improvements in dense representation learning and vector database technologies continue to enhance the quality and scalability of retrieval systems.

2.1.9.1. Efficient Similarity Search with FAISS

To make semantic retrieval computationally scalable, many retrieval-augmented systems rely on efficient indexing libraries. One of the most widely used tools for this purpose is **FAISS** (Facebook AI Similarity Search), an open-source library designed for high-speed similarity search and clustering of dense vectors (Johnson et al., 2017).

FAISS supports both *exact* and *approximate* nearest neighbor (ANN) search across high-dimensional vector spaces. For large-scale applications such as document retrieval or conversational systems, approximate methods strike a balance between retrieval quality and latency. FAISS provides several indexing strategies, including:

- `IndexFlatL2`: brute-force search using Euclidean distance.
- `IndexIVFFlat`: inverted file index with centroid-based clustering.
- `IndexHNSW`: hierarchical navigable small world graphs.

The most common usage in semantic retrieval pipelines is to first train an index over document embeddings and then perform similarity search using cosine similarity or Euclidean distance. While FAISS does not natively support cosine similarity, it can be emulated by normalizing all vectors to unit length, since:

$$\text{cosine\_similarity}(a, b) = \frac{a \cdot b}{\|a\|\|b\|} \quad \Rightarrow \quad \text{if } \|a\| = \|b\| = 1, \quad a \cdot b = \cos(\theta)$$

In that case, cosine similarity becomes equivalent to inner product, and ranking results by decreasing inner product yields the same ordering as cosine similarity.

FAISS indexes are usually stored as binary files (e.g., `.faiss`) and can be loaded into memory at runtime or persisted for repeated use. The indexing and search process is highly optimized using SIMD instructions and GPU acceleration when available.

Index Type	**Search Speed**	**Accuracy**
IndexFlatL2	Slow (exact)	100%
IndexIVFFlat	Fast (approx.)	90–99%
IndexHNSW	Very fast (approx.)	95–99%

Table 2.5.: Comparison of selected FAISS index types

In practice, FAISS is particularly suited for RAG-based chatbots and open-domain QA systems that need to balance fast retrieval with reasonable accuracy across large document collections.

2.1.10. Prompt Engineering and Metric-Driven Evaluation

Large Language Models (LLMs) exhibit remarkable flexibility when guided by carefully constructed instructions, known as *prompts*. In contrast to earlier approaches that relied on fine-tuning for each task, prompting enables general-purpose models to perform specific actions through plain language input. This shift has given rise to the concept of *prompt engineering*—the systematic design of prompts to achieve consistent and accurate outputs (Reynolds & McDonell, 2021) (Zhou et al., 2023).

In the context of retrieval-augmented generation (RAG), prompt engineering becomes particularly important. Prompts in such systems not only guide the language model to formulate an answer, but also contextualize the retrieved evidence, apply domain-specific constraints, or impose stylistic formatting. The way a prompt is framed can significantly influence model behavior.

Prompt	**LLM Output**
What is the Treaty of Versailles?	The Treaty of Versailles was a peace treaty signed in 1919 to end World War I.
Briefly explain the Treaty of Versailles as if to a 10-year-old.	It was a big agreement after World War I that told countries how to make peace and what rules to follow.

Table 2.6.: Impact of prompt variation on output tone and complexity

As seen above in Table 2.6, small changes in the phrasing of a prompt can yield outputs with different tone, structure, and granularity. In more complex systems, such variation becomes even more critical—especially when dealing with educational use cases, where age, prior knowledge, or question complexity must be taken into account.

To validate whether such prompts produce relevant and reliable responses, structured evaluation frameworks are needed. One prominent tool in this space is **RAGAS**, a modular evaluation framework tailored for RAG pipelines (Es et al., 2025). It supports both human-annotated and automatically generated test sets, which makes it suitable for evaluating systems in realistic usage scenarios.

A notable strength of RAGAS lies in its ability to synthesize evaluation data from a document corpus using query generation methods. These methods can create various types of synthetic questions (e.g., factual, reasoning-based, multi-hop) along with corresponding ground truth answers and source contexts.

In evaluating a system's performance, RAGAS introduces key metrics such as:

- **Context Precision**: How relevant are the retrieved documents to the ground truth?
- **Answer Relevancy**: Does the generated response adequately address the question?
- **Faithfulness**: Is the output verifiably supported by the context provided?

Before applying these metrics, a shared terminology is used for consistency:

- **Question**: The user query being answered.
- **Answer**: The generated text produced by the model.
- **Context**: Retrieved document chunks used to inform the answer.
- **Ground Truth**: The reference answer for comparison.
- **Ground Truth Context**: The original document passage from which the ground truth was derived.

Together, prompt engineering and metric-driven evaluation define the interface between human intent and machine response in RAG systems. By carefully aligning both, developers can construct systems that are not only technically robust, but also pedagogically meaningful.

2.1.10.1. Semantic Embeddings: Structure and Application

Embeddings are a core technique in machine learning for representing diverse types of data—such as words, images, or user behavior—as vectors in high-dimensional space. The fundamental idea is that semantically similar entities will cluster together, enabling powerful operations for search, recommendation, clustering, or classification tasks.[18]

[18]https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings?hl=en, letzter Zugriff am 19.07.2025

A prominent example is Word2Vec, which produces vector representations of words that not only capture similarity but also support algebraic manipulation. This means that semantic relations—such as gender, geography, or tense—can be translated into vector operations. A classical illustration is shown in Figure 2.4, where the operation *Tokyo – Japan + France* yields a vector that lies near *Paris*—demonstrating the model's ability to generalize across semantic domains.[19]

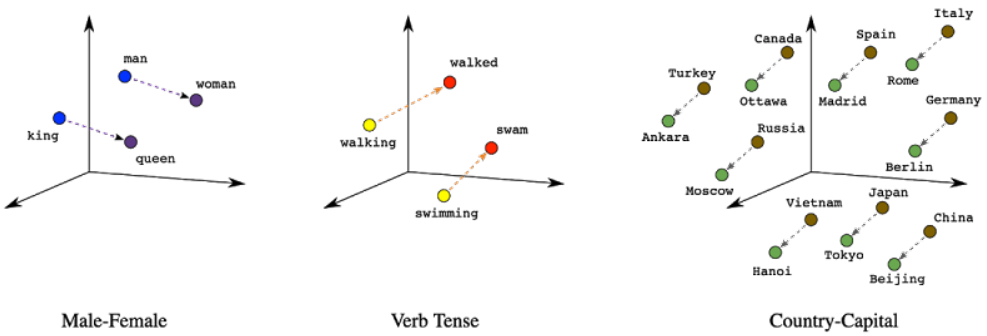

Figure 2.4.: Semantic analogy captured through word embeddings [19]

Such structures allow the development of semantic search systems that go beyond exact keyword matching. For instance, a query like "European fashion capitals" would correctly identify cities like Paris, Milan, or Barcelona—even if these specific terms do not appear in the original query—due to their proximity in embedding space.

Practical implementation often relies on pre-trained models such as the Universal Sentence Encoder or OpenAI's CLIP, which embed text, images, or both into shared vector spaces. Alternatively, domain-specific embeddings can be learned using two-tower architectures, especially useful for use cases such as personalized product recommendations.

2.1.11. Evaluating Retrieval Quality

Retrieval quality plays a central role in the performance of Retrieval-Augmented Generation (RAG) systems. Since such systems generate answers based on documents retrieved from a knowledge base, any shortcomings in this first step can propagate through the entire pipeline. The principle *"garbage in, garbage out"* is especially applicable here. Even the most capable language model cannot compensate for missing or irrelevant input. This section presents three commonly used metrics for assessing retrieval quality: **Context Precision**, **Context Recall**, and the now-deprecated **Context Relevancy**. These metrics are implemented in frameworks such as RAGAS.

[19] https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings?hl=en, letzter Zugriff am 19.07.2025

2.1.11.1. Context Precision

Context Precision evaluates whether relevant information appears early among the retrieved document chunks. This is crucial due to the known limitations in attention allocation of large language models, which often favor information at the beginning or end of the input sequence.

The metric is calculated as:

$$\text{Context Precision@K} = \frac{\sum_{k=1}^{K} \text{TP@k}}{\sum_{k=1}^{K} (\text{TP@k} + \text{FP@k})} \tag{2.2}$$

where:

- TP@k = number of true positives at position k
- FP@k = number of false positives at position k

A high score indicates that relevant chunks are ranked highly, improving the likelihood of generating accurate answers.

2.1.11.2. Context Relevancy (Deprecated)

Previously, Context Relevancy measured the share of relevant sentences in the retrieved context:

$$\text{Context Relevancy} = \frac{|\text{Relevant Sentences}|}{|\text{Total Sentences in Context}|} \tag{2.3}$$

However, this metric was removed from the RAGAS framework due to its limited utility in predicting output quality. As discussed by the maintainers,[20] its correlation with answer faithfulness was weak, and it was replaced by stronger metrics such as Context Precision.

Interestingly,Cuconasu et al. (2024) observed that under specific conditions, certain types of "noise" in the context (e.g., irrelevant but semantically close sentences) may actually support performance, contradicting earlier assumptions about strict relevance filtering.

2.1.11.3. Context Recall

Context Recall complements Precision by assessing how complete the retrieval step is with respect to the expected answer. It is defined as:

$$\text{Context Recall} = \frac{|\text{GT Sentences in Context}|}{|\text{Total GT Sentences}|} \tag{2.4}$$

where:

[20] https://github.com/explodinggradients/ragas/pull/1111

- |GT Sentences in Context| = number of ground truth sentences found in the retrieved chunks
- |Total GT Sentences| = total number of sentences in the ground truth

While Precision answers *"how relevant is what I got?"*, Recall asks *"did I get everything I needed?"*. A high recall score ensures that the necessary information for answering the query is available to the model.

"Lost in the Middle" effects, as described by N. F. Liu et al. (2023), further emphasize the need for both high Precision and high Recall — as not only availability, but also position, affects model performance.

2.1.11.4. Illustrative Example of Retrieval Evaluation Metrics

To better understand the practical implications of retrieval quality, let us consider a simplified example. Suppose the system receives the following query:

> **Question:** When was Marie Curie born?

The ground truth answer is based on a known source:

> **Ground Truth Sentence:** Marie Curie was born on November 7, 1867, in Warsaw, Poland.

Now, assume the retriever returns the following three context chunks:

Context Chunk 1: Marie Curie was a pioneering physicist and chemist who conducted groundbreaking research on radioactivity.
Context Chunk 2: She was the first woman to win a Nobel Prize. Marie Curie was born on November 7, 1867. Her work paved the way for nuclear physics.
Context Chunk 3: Curie studied at the Sorbonne in Paris and became the first female professor there.

Table 2.7.: Retrieved Context Chunks

Context Precision: This metric evaluates how early the relevant information appears in the list of retrieved chunks. In this case, the correct answer appears in **Chunk 2**, which is second in the list. Assuming $K = 3$, the Context Precision@3 would consider the position of relevant chunks as:

$$\text{Context Precision@3} = \frac{1}{3} \quad \text{(1 relevant chunk out of 3)}$$

Context Recall: Context Recall evaluates how much of the Ground Truth is covered in the retrieved context. Since Chunk 2 contains the full Ground Truth sentence, the recall is:

$$\text{Context Recall} = \frac{1}{1} = 1.0$$

Context Relevancy (Deprecated): This metric measures the proportion of relevant sentences in all retrieved content. Let's assume:

- Chunk 1: 0 relevant sentences
- Chunk 2: 1 relevant sentence
- Chunk 3: 0 relevant sentences

Total sentences = 3, Relevant = 1

$$\text{Context Relevancy} = \frac{1}{3} \approx 0.33$$

This simplified example demonstrates how all three metrics differ in their evaluation logic and why precision and recall remain useful, while relevancy may be more ambiguous and sensitive to sentence segmentation and interpretation.

2.1.12. Evaluating Answer Generation

Beyond retrieving relevant documents, a crucial challenge in Retrieval-Augmented Generation (RAG) lies in how accurately and appropriately a language model transforms retrieved knowledge into a final answer. Even when the input context is relevant, a model might hallucinate, generalize incorrectly, or miss the core of the question. Thus, assessing the quality of generated answers is essential for understanding the true performance of a RAG system.

2.1.12.1. Faithfulness

Faithfulness measures whether the claims made in an answer can be directly supported by the content retrieved. In other words, it checks whether the model has remained "grounded" in the source material.

Let $|V|$ denote the number of statements in the answer that can be verified by the provided context, and let $|S|$ represent the total number of claims in the answer. The metric is computed as:

$$\text{Faithfulness} = \frac{|V|}{|S|} \tag{2.5}$$

A high score indicates that the model correctly uses the available context, while a low score suggests hallucination or unsupported claims. An example can be seen in Table 2.8.

Question	Context	Answer
When did Ada Lovelace publish her translation of Menabrea's paper?	In 1842, Ada Lovelace translated Luigi Menabrea's paper and added extensive notes, which were later published in 1843.	1843.
When did Ada Lovelace publish her translation of Menabrea's paper?	In 1842, Ada Lovelace translated Luigi Menabrea's paper and added extensive notes, which were later published in 1843.	1845.

Table 2.8.: High (top) vs. low (bottom) Faithfulness based on factual support in the context

2.1.13. Reranking in RAG Systems

While the initial retrieval step in a RAG pipeline is designed for speed and recall—finding a broad set of potentially relevant documents—it often lacks precision. The initial retriever might pull in documents that are only superficially related to the query. To address this, a crucial second step known as reranking is often employed. The goal of the reranker is to take the initial list of candidate chunks and re-sort them based on a more nuanced and accurate measure of relevance, ensuring that the best possible context is passed to the LLM (Thakur et al., 2021).

This is typically achieved by using a more powerful but computationally expensive model, often a cross-encoder. Unlike the initial retriever, which computes embeddings for the query and documents separately, a cross-encoder processes the query and each candidate chunk together. This allows it to pay close attention to the fine-grained interactions between the words in the query and the words in the chunk, leading to a much more accurate relevance score (Humeau et al., 2020).

Consider the following example: A student asks a RAG system about the impact of the steam engine on the textile industry. The initial retriever might return a list of chunks, which are then re-evaluated by the reranker.

In this example seen in Table 2.9 the initial retriever ranked the general document about the steam engine highest. The reranker, however, understood that the third document, despite a lower initial score, provided the most direct and specific answer by explicitly linking steam power to textile factories. By promoting this chunk to the top, the reranker provides the LLM with a much higher quality context, which is essential for generating a precise and factually correct final answer (Ovadia et al., 2024)

Table 2.9.: Example of a reranking process.

Rank	Chunk content (excerpt)	Initial score	New rank	Reranker rationale
Query: "How did the introduction of the steam engine change the textile industry?"				
Phase 1: Fast retrieval (initialr etrieval)				
1	…the steam engine, invented by James Watt, was a key technology of the Industrial Revolution…	0.91	2	High relevance but rather general.
2	…the textile industry experienced an enormous boom thanks to new spinning machines such as the Spinning Jenny…	0.88	3	Mentions textile industry but not the steam engine.
3	…using steam power in textile mills enabled the mass production of fabrics and made production independent of watercourses…	0.85	1	Perfectly covers both core concepts (steam engine
textile industry).				
4	…working conditions in the factories of the Industrial Revolution were often poor…	0.79	4	Thematically related but does not directly answer the question.
Phase 2: Precise re-evaluation (reranking)				
1	…using steam power in textile mills enabled the mass production of fabrics and made production independent of watercourses…	0.98	1	Highest relevance; describes the causal link directly.
2	…the steam engine, invented by James Watt, was a key technology of the Industrial Revolution…	0.90	2	Important context but less specific than Rank 1.
3	…the textile industry experienced an enormous boom thanks to new spinning machines such as the Spinning Jenny…	0.65	3	Lower relevance because the direct link to the steam engine is missing.
4	…working conditions in the factories of the Industrial Revolution were often poor…	0.40	4	Least relevant to the specific question.

2.1.13.1. Cross-Encoder and Trade-off costs

A cross-encoder reranker feeds the concatenated query + candidate passage sequence into a single Transformer stack, allowing every query token to attend to every passage token; the `[CLS]` representation is then mapped to a scalar relevance score by a lightweight MLP (or even a single linear layer) (Humeau et al., 2020). Because interactions are modelled at token level, cross-encoders routinely outperform dual-encoder similarity in fine-grained ranking tasks, lifting nDCG@10 by 6–10 points on MS MARCO compared with the best bi-encoders (Thakur et al., 2021). Typical off-the-shelf models include `cross-encoder/ms-marco-MiniLM-L-6-v2`—a 66 M-parameter distilled BERT trained with pairwise margin loss on 400 k MS MARCO judgements (Thakur et al., 2021)—and `bge-reranker-base`, a 110 M-parameter model further tuned on 120 M natural language inference and web query–passage pairs to generalise beyond MS MARCO (Y. Liu et al., 2024). Both rely on the MS MARCO Passage Ranking collection (8.8 M passages, 1 M judged triples) as their primary supervision signal (Bajaj et al., 2016), yet achieve state-of-the-art zero-shot transfer on benchmark suites such as BEIR. The trade-off is latency: inference scales linearly with the number k of candidates because each query–passage pair requires a full Transformer forward pass. Even efficient models like MiniLM process only $\approx 3000\,\text{passages} \cdot \text{s}^{-1}$ on an A100 GPU (batch = 512, FP16); thus production systems often "retrieve 40 $\rightarrow$ rerank 8" or cache encoder outputs to keep 99^{th}-percentile latency under 250 ms (Thakur et al., 2021) (Ma et al., 2023). Nevertheless, for answer-oriented RAG pipelines, the precision boost of cross-encoder reranking typically outweighs the added 20 ms to 40 ms per request, because higher-quality context reduces hallucinations and downstream generation cost.

2.1.14. Answer Relevancy

A faithful answer may still miss the point. Answer relevancy evaluates how well the answer actually addresses the user's question. The underlying assumption is that if an answer contains sufficient information to reconstruct the question, it is likely to be relevant.

To quantify this, reverse-engineered questions are generated from the answer and compared to the original query via cosine similarity between their vector embeddings. Let E_o be the embedding of the original question, and E_{g_i} the embedding of the i-th generated question (out of N total). The score is computed as:

$$\text{Answer Relevancy} = \frac{1}{N} \sum_{i=1}^{N} \frac{E_{g_i} \cdot E_o}{\|E_{g_i}\| \|E_o\|} \tag{2.6}$$

A high score indicates that the generated answer remains semantically close to the intent of the original question.

2.1.14.1. Answer Semantic Similarity

Sometimes the best metric is simply comparing the generated answer to a ground truth answer in terms of meaning. This is where semantic similarity becomes useful. Both the generated answer and the reference are embedded into a vector space, and their cosine similarity is calculated.

$$\text{Semantic Similarity} = \frac{E_a \cdot E_{gt}}{\|E_a\|\|E_{gt}\|} \tag{2.7}$$

Where E_a is the embedding of the generated answer and E_{gt} is the ground truth embedding. This provides a nuanced score of how semantically close both answers are, even if their phrasing differs.

2.1.14.2. Answer Correctness

Finally, to combine semantic alignment and factual grounding, Answer Correctness evaluates both factual correctness and relevance. This is done using the F1-score, based on overlaps in factual claims.

Let:

- TP: facts correctly present in both answer and ground truth
- FP: facts present in the answer but not in the ground truth
- FN: facts missing in the answer but present in the ground truth

Then the F1-score is:

$$\text{F1 Score} = \frac{|TP|}{|TP| + 0.5 \cdot (|FP| + |FN|)} \tag{2.8}$$

A weighted average of this score and the semantic similarity then yields the final Answer Correctness metric.

Summary: These metrics—faithfulness, relevancy, semantic similarity, and correctness—collectively provide a multi-dimensional perspective on generation quality. Their inclusion in modern RAG evaluation frameworks such as RAGAS ensures not only high-quality retrieval but also trustworthy answer formulation.

2.1.15. Evolution of Question Types in RAG Evaluation

A critical component of building a robust RAG system is evaluating its performance, which requires a high-quality and diverse test set of questions. The RAGAS framework (Es et al., 2025) initially approached this by ”evolving” simple questions into more complex ones using a method inspired by Evol-Instruct. This earlier method categorized questions into abstract types such as `simple`, `reasoning`, and `multi-context`, each representing increasing cognitive complexity.

However, this categorization has since been replaced by a more structured and semantically grounded methodology. Recent versions of RAGAS incorporate an intermediate step where a knowledge graph is constructed from the underlying documents. This graph explicitly maps entities and their relationships, forming a basis for generating evaluation questions in a controlled and reproducible manner.

Instead of relying on generic linguistic templates, modern *synthesizers* formulate queries directly based on graph relations. This makes the resulting questions both more realistic and better aligned with actual user intents. Furthermore, the graph-based generation enables the creation of more nuanced question types that capture both syntactic and semantic complexity, which is essential for evaluating multi-hop reasoning capabilities in LLMs. Table 2.10 shows the evolution from legacy to modern syntheziers.

Table 2.10.: Comparison of Legacy and Modern RAGAS Question Generation.

Type	**Description**	**Example Question (based on financial reports)**
Legacy "Evolutions"		
`simple`	A basic question answered by a single context.	"What was the revenue in Q2?"
`reasoning`	Requires logical inference over known facts.	"Given the Q2 revenue and Q1 expenses, was the company profitable?"
`multi-context`	Requires integration across different documents or contexts.	"How did the marketing strategy mentioned in the shareholder letter affect the Q2 sales figures?"
Modern Synthesizers (Graph-based)		
`single_hop_specific`	Precise and grounded in one fact node in the graph.	"What was the exact revenue figure for the European market in Q2?"
`multi_hop_specific`	Combines multiple distinct facts from different nodes.	"Compare the Q2 revenue from Europe with the R&D spending in the appendix."
`multi_hop_abstract`	Requires broad synthesis and abstraction from the document graph.	"What was the overall financial trend in the first half of the year?"

This methodological transition represents more than a technical update—it marks a shift towards evaluation grounded in structural semantics rather than surface-level text variation. It also improves reproducibility and control over question difficulty, a key requirement in educational and factual-critical RAG use cases. Moreover, the structured generation aligns more closely with real-world question styles, moving away from artificial constructs that often failed to generalize beyond synthetic benchmarks.

2.1.15.1. Knowledge Graph vs Non-Knowledge Graph RAG

Recent empirical work on "GraphRAG" variants shows that simply adding a knowledge graph layer does not automatically improve every retrieval metric. In a recent study all GraphRAG methods achieved a context relevancy of about 0.74—nearly identical to a FAISS-only baseline—suggesting that entity-relationship indexing alone may not boost basic retrieval coverage. However, when Neo4j's native vector index is enabled, answer relevancy climbs from 0.87 (FAISS) to 0.93, and faithfulness nearly doubles (from 0.20 to 0.52). These gains indicate that graph-backed indices can reduce hallucinations and sharpen answer precision—but only at the cost of increased infrastructure complexity and potential ROI concerns for large-scale deployments.[21]

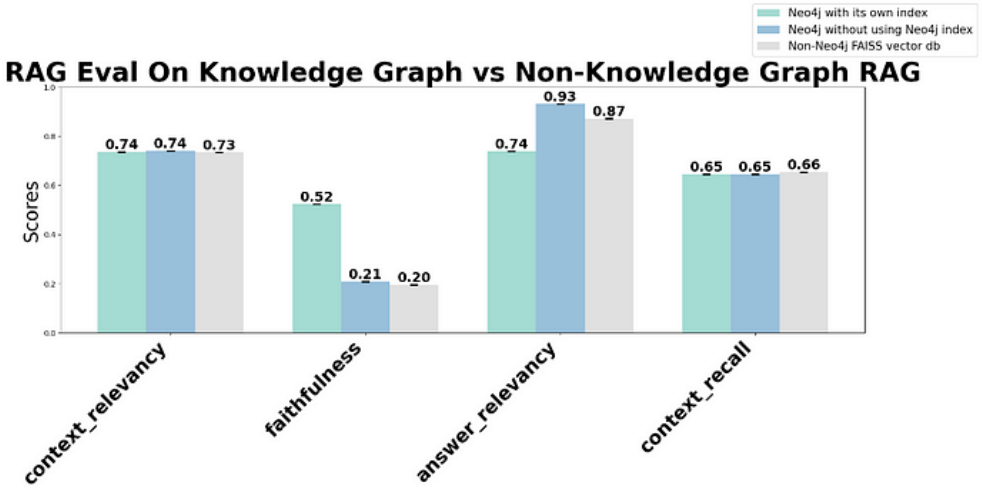

Figure 2.5.: Potential ethical and societal risks of AI applications in education [22]

Beyond precision and faithfulness, Bennett also explores broader "comprehensiveness" and "diversity" metrics, asking not just "Is the answer correct?" but "How complete and richly detailed is it?" While a built-in "Directness" measure controls for answer length, Bennett finds the notion of diversity—capturing the range of perspectives in a response—remains underdeveloped in most RAG evaluations. This underscores an important direction for future work: combining structural graph insights with more sophisticated semantic-diversity metrics could yield evaluation sets that both challenge multi-hop reasoning and reward genuinely novel, contextually enriched answers.[22]

[21] https://towardsai.net/p/artificial-intelligence/graphrag-analysis-part-1-how-indexing-elevates-knowledge-graph-perfo letzter Zugriff am 20.07.2025

[22] https://towardsai.net/p/artificial-intelligence/graphrag-analysis-part-1-how-indexing-elevates-knowledge-graph-perfo letzter Zugriff am 20.07.2025

2.1.16. Chunking Strategies for RAG

The effectiveness of a RAG system is critically dependent on the quality of the information retrieved, which in turn depends on how the source documents are broken down into smaller pieces, or "chunks." This process, known as chunking, is a foundational step in preparing a knowledge base. If chunks are too small, they may lack sufficient context to be meaningful. If they are too large, they can introduce noise and exceed the context window of the LLM. The choice of chunking strategy is therefore a crucial design decision (Y. Gao et al., 2024).

Several strategies exist, ranging from simple to highly sophisticated:

- **Sentence Splitting:** This method uses natural language processing libraries to split documents along sentence boundaries (e.g., periods, question marks). The primary advantage is that each chunk is a grammatically complete and semantically coherent unit. However, individual sentences can often lack the broader context necessary to answer a complex query, and sentence lengths can be highly inconsistent (Kudo & Richardson, 2018).
- **Recursive Character Splitting:** This is a more robust and widely used method, popularized by frameworks like LangChain. It attempts to split the text based on a hierarchical list of character separators. The algorithm tries to split first by a large separator (e.g., double newlines `\n\n` to keep paragraphs together), and if a resulting chunk is still too large, it recursively splits it by the next separator in the list (e.g., single newlines `\n`, then spaces). This method is effective at keeping semantically related content together while respecting a defined chunk size.[23]
- **Semantic Chunking:** This state-of-the-art approach moves beyond rule-based splitting. Instead of relying on character counts or punctuation, semantic chunking uses the meaning of the text itself to determine breakpoints. The process involves converting sentences into numerical embeddings and then identifying points where the semantic similarity between consecutive sentences drops significantly, indicating a topic shift. These breakpoints are used to form chunks of semantically cohesive content.[24] This method excels at creating chunks that are contextually rich and directly relevant to a potential query, leading to more accurate retrieval, but it is computationally more expensive than simpler methods.

Figure 2.6 and Figure 2.7 show examples of recursive as well as semantic splitting within a paragraph, exposing either the overlapping chunks that don't necesearily capture a single topic or likewise the chunks don't necesarily equal in length.[25]

[23] https://python.langchain.com/docs/how_to/recursive_text_splitter/, letzter Zugriff am 26.07.2025

[24] https://medium.com/data-science/the-art-of-chunking-boosting-ai-performance-in-rag-architectures-acdbdb8

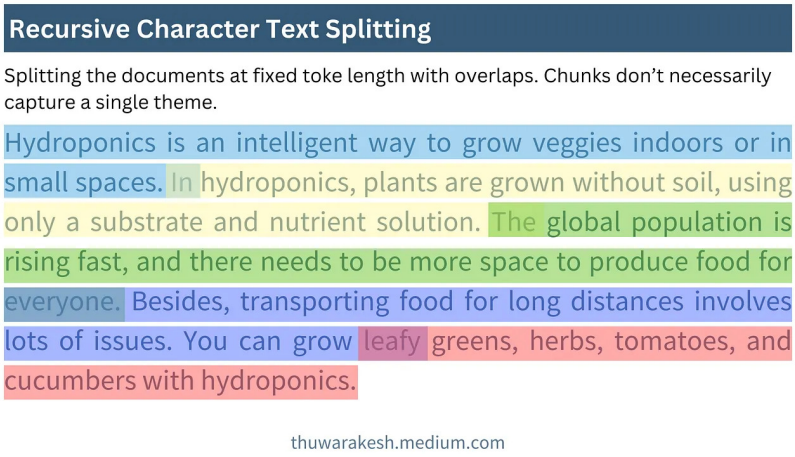

Figure 2.6.: Recursive splitting [25]

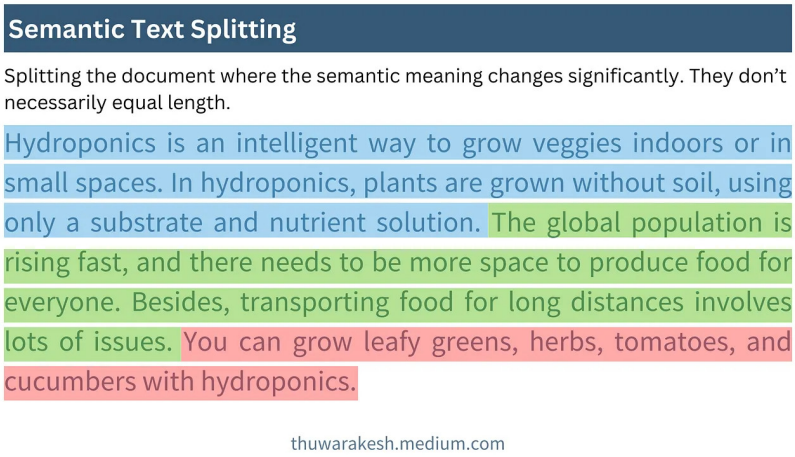

Figure 2.7.: Splitting document with semantic meaning [25]

Other strategies have also been proposed in recent literature, each with specific trade-offs:

Fixed-Length Token Chunking: In this method, text is split into fixed-size segments based purely on token count, regardless of natural language structure. This strategy is fast and easy to implement, and ensures predictable chunk sizes for LLM input. However, it risks cutting off sentences mid-way and often produces semantically incoherent chunks.

Windowed Overlapping Chunking: To mitigate context loss at chunk boundaries, this approach adds a small overlap (e.g., 10–20 %) between consecutive

letzter Zugriff am 26.07.2025

[25]https://medium.com/data-science/semantic-chunking-for-rag-35b7675ffafd, letzter Zugriff am 25.07.2025

chunks. While this increases redundancy in the index, it helps preserve continuity and supports more complete answers for queries that reference multiple adjacent concepts.

Title-Based or Section-Aware Chunking: In structured documents like textbooks or encyclopedias, headings and subheadings can guide logical chunk boundaries. Chunks aligned with topic transitions tend to be more coherent and useful for retrieval. However, this method requires structured input and may not generalize well to unstructured text sources.

Sliding Window Chunking: This variant creates multiple overlapping chunks using a fixed window and step size (e.g., 200-token window, 50-token stride). It is often used in training data generation for QA systems. While exhaustive, it is highly redundant and computationally intensive for large corpora.

Table 2.11 shows a comprehensive overview with all common chunking strategies including their advantages and disadvantages:

Table 2.11.: Overview of common chunking strategies in RAG systems

Strategy	Description	Pros and Cons
Sentence Splitting	Splits text at sentence boundaries using NLP parsers.	+ Grammatically coherent – Context often too narrow
Recursive Character Splitting	Hierarchical splitting by separators (e.g. paragraphs, newlines, spaces).	+ Preserves structure + Flexible – May still split mid-topic
Semantic Chunking	Uses embedding similarity to detect topic shifts between sentences.	+ Contextually coherent + High retrieval accuracy – Computationally expensive
Fixed-Length Token Chunking	Uniform chunks based on token count only.	+ Simple + Fast – Ignores sentence/context
Overlapping Chunking	Adds overlap between chunks to preserve context flow.	+ Better continuity – Increased index size
Title-Based Chunking	Uses section headings or titles to define chunk boundaries.	+ Topic-aligned – Requires structured input
Sliding Window	Moves a fixed-size window over the text with a set stride.	+ Exhaustive – Very redundant and slow

2.2. Related Work

This section surveys current initiatives that leverage AI, large-language models, and immersive technologies for teaching history, outlining the advantages they offer as well as the limitations most relevant to the present study.

2.2.1. Historical Evolution of Conversational Agents

Long before today's transformer-based chatbots captured the popular imagination, the intellectual roots of conversational AI reached back to Alan Turing's 1950 "imitation game," which framed dialogue as a litmus test for machine intelligence. The first working demonstration was *ELIZA* (1966), Weizenbaum's rule-based program that used keyword substitution to impersonate a Rogerian psychotherapist—showing how readily humans anthropomorphize even rudimentary text output (Weizenbaum, 1966). Six years later, psychiatrist Kenneth Colby released *PARRY*, a Lisp system that modeled the conversational patterns of a paranoid patient and famously fooled some clinicians in a blind test (Colby, 1981). These early systems were entirely symbolic, relying on handcrafted pattern rules and finite state logic.

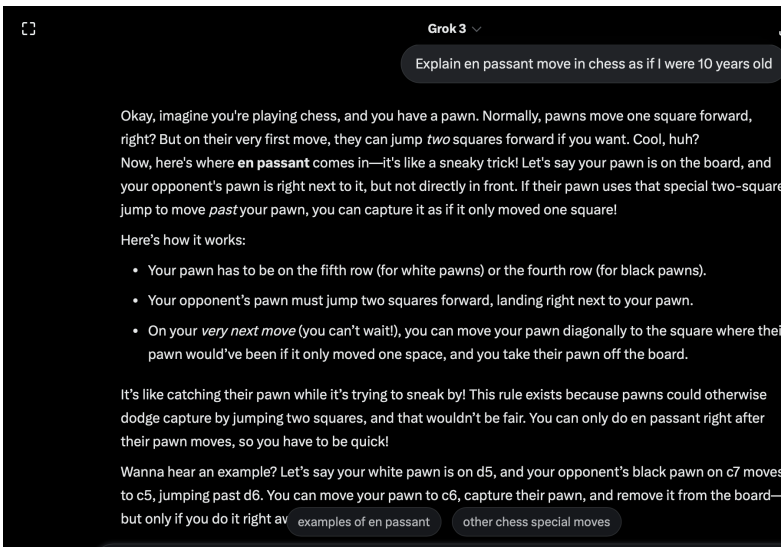

Figure 2.8.: Example chat with X's chatbot Grok3

The 1990s introduced data-driven chatterbots such as Richard Wallace's A.L.I.C.E., which leveraged AIML to win multiple Loebner Prizes. A new wave of commercial voice assistants followed in the 2010s—Apple's *Siri* (2011), Amazon's *Alexa* (2014)—combining cloud speech recognition with intent classification. The most dramatic leap arrived with transformer architectures: OpenAI's GPT series and other large language models, trained on internet-scale corpora, demonstrated few-shot and zero-shot generalization, culminating in the public launch of ChatGPT in late 2022. Whereas ELIZA matched surface keywords, modern systems generate

coherent, context-aware prose and can be coupled with retrieval pipelines for factual grounding. This evolution—from symbolic scripts to massive self-supervised models—frames the technical context in which educational chatbots like *ALiVE* now operates.

2.2.2. AI-Driven Chatbots in Education

The integration of AI-driven chatbots into educational settings has matured from niche experiments into a significant field of research and application. A growing body of evidence suggests that these tools can offer tangible benefits for learning. Systematic reviews consistently identify several key advantages: chatbots provide instant, on-demand support and feedback to students, allow for learning at a personalized pace, and can increase motivation through interactive and non-judgmental dialogue (Wollny et al., 2021). Beyond simple Q&A, more advanced pedagogical agents are now being designed to scaffold higher-order skills. For example, according to Pinter et al. (2025), students who evaluated ChatGPT during history classes demonstrated greater engagement and improved performance compared to a control group that did not use ChatGPT.

However, the widespread enthusiasm is tempered by significant and unresolved challenges. A primary criticism is that many educational chatbots are developed with a focus on technological novelty rather than being grounded in established pedagogical theory, making their educational impact questionable (Molenaar, 2022). The rise of powerful generative models like ChatGPT has amplified these concerns, introducing serious risks related to factual accuracy (hallucination), data privacy, and the potential for students to develop an over-reliance on AI that could erode critical thinking skills (Baidoo-Anu & Ansah, 2023). Furthermore, the effectiveness of these tools is often evaluated based on user satisfaction or engagement metrics rather than on rigorous, direct measures of learning outcomes (Bond et al., 2020). These shortcomings highlight a critical need for chatbot designs that are not only technologically capable but also pedagogically informed, domain-specific, and ethically robust.

2.2.3. Representative Educational Chatbots

In recent years, a variety of AI–driven tutoring agents have moved from research labs into real-world classrooms. Below there are highlighted five widely cited exemplars and compare their design choices.

Khanmigo (KhannAcademy). Built in partnership with OpenAI, Khanmigo offers Socratic guidance across Khan Academy's full content library. Rather than giving direct answers, it prompts learners to articulate their thinking and next steps, while teachers receive dashboards that surface misconceptions.[26]

[26]https://www.khanmigo.ai, letzter Zugriff am 01.07.2025

Duolingo Max. Launched as a premium tier in 2023, Duolingo Max embeds GPT-4 to power two features: *Roleplay*, which simulates conversation partners, and *Explain My Answer*, an on-demand error analysis tool. Early internal studies report higher user engagement than standard lessons.[27]

Squirrel AI. China-based Squirrel AI combines adaptive sequencing with mastery diagnostics in overn 3,000 learning centers. A dual-teacher model pairs human mentors with an AI engine that dynamically recomposes micro-lessons, claiming large gains on standardized tests.[28]

Jill Watson (Georgia Tech). Originally deployed in an online CS course, Jill Watson is a retrieval-augmented virtual TA that answers thousands of Piazza forum questions each term, achieving near-human response quality while reducing staff workload.[29]

CENTURY Tech. A UK platform that couples adaptive learning paths with teacher analytics for K-12 English, maths, and science. Its engine continuously recalibrates content difficulty and highlights gaps for timely intervention.[30]

A comparision of the discussed edu-chatbots can be found below in Table 2.12

2.2.4. Chunking-Based and Retrieval-Augmented Architectures

Although many educational bots rely on a single, fixed knowledge base, a growing class of systems combines *document chunking*, dense *vector search*, and *prompt-time fusion*—the recipe usually referred to as Retrieval-Augmented Generation (RAG). Below there are five widely cited RAG frameworks whose design choices resonate with, or provide useful contrast for, the *ALiVE* pipeline.

LangChain RAG Pipeline. LangChain popularised an end-to-end "index / retrieve / generate" pattern: documents are split into overlapping chunks, embedded, stored in a vector index (e.g. FAISS), and finally injected into an LLM prompt.[31] Because all steps are modular, LangChain is often the reference implementation for course projects and proof-of-concept chat applications.

LlamaIndex (formerly GPT Index). LlamaIndex focuses on sophisticated chunking heuristics—sentence windowing, hierarchical nodes, recursive summarisation—to maximise relevant context per token.[32] Internal benchmarks show that smart chunk

[27] https://de.duolingo.com/help/what-is-duolingo-max, letzter Zugriff am 01.07.2025
[28] https://squirrelai.com, letzter Zugriff am 01.07.2025
[29] https://dilab.gatech.edu/jill-watson/, letzter Zugriff am 01.07.2025
[30] https://www.centuryk12.com, letzter Zugriff am 01.07.2025
[31] https://python.langchain.com/docs/tutorials/rag/, letzter Zugriff am 03.07.2025
[32] https://docs.llamaindex.ai/en/stable/, letzter Zugriff am 03.07.2025

Table 2.12.: Comparison of Selected Educational Chatbots

Bot	**Primary Domain**	**Key Pedagogical Feature**	**Underlying Tech**	**Reported Benefit / Limitation**
Khanmigo	General K-12 subjects	Guided Socratic prompts, teacher dashboards	GPT-4 with safety layer, retrieval from Khan content	High learner engagement; requires $10/month pilot fee
Duolingo Max	Language learning	Role-play dialogues; on-demand error explanations	GPT-4; proprietary lesson graph	Improved retention for paid subscribers; limited to iOS & Android premium
Squirrel AI	STEM tutoring (China)	Real-time mastery diagnostics; dual-teacher model	Custom adaptive engine; local knowledge graph	Claims 80–90% test-score gains; research mostly company-run
Jill Watson	University discussion forums	Virtual TA answers forum questions 24/7	Retrieval-Augmented GPT; IBM Watson tools	Cuts TA workload by ∼60%; domain-specific setup required
CENTURY Tech	UK K-12 core subjects	Adaptive sequencing plus teacher analytics	Neural mastery model; dashboard BI	Reduced teacher marking time; licensing cost cited by schools

boundaries improve both retrieval precision and downstream answer faithfulness compared with naïve fixed-size splits.

Microsoft GraphRAG. GraphRAG first converts text into a knowledge graph (entities and relations) and then performs retrieval over graph nodes before passing the stitched evidence to an LLM.[33] A recent independent replication found that graph indexing alone does *not* raise context-relevancy scores over FAISS baselines (0.74), yet markedly lifts faithfulness when Neo4j's native vector index is enabled, cutting hallucinations nearly in half. (see subsubsection 2.1.15.1).

Haystack RAG Framework. Haystack integrates fine-tuned embedding models with hybrid BM25+ dense retrieval, plus evaluation hooks for ranking and answer quality.[34] Its conference series has spotlighted chunking strategies that incorporate user-behaviour signals (e.g. click-through) to re-rank passages, an approach that boosts recall for open-ended queries.

REPLUG. REPLUG treats the language model as a black box: retrieved passages are simply prepended to the input, keeping the LM frozen (W. Shi et al., 2023). This minimalistic design enables rapid experimentation with new retrievers or

[33]https://microsoft.github.io/graphrag/, letzter Zugriff am 03.07.2025
[34]https://haystack.deepset.ai, letzter Zugriff am 05.07.2025

chunkers without re-training the generator; in zero-shot QA benchmarks it narrows the performance gap to fully fine-tuned models.

Together, these architectures illustrate the design space in which *ALiVE* positions itself: (1) task-aware chunking to maximise semantic coherence, (2) hybrid or graph-enhanced retrieval for complex historical queries, and (3) strict faithfulness checks to mitigate hallucinations—an essential requirement in educational contexts.

2.2.5. Positioning the *ALiVE* System within the State of the Art

Table 2.13 juxtaposes the core design choices of *ALiVE* with five representative RAG frameworks discussed earlier. Three differentiators stand out.

1. Pedagogical grounding. Whereas most RAG pipelines optimise generic QA metrics, *ALiVE* is explicitly aligned with principles of historical thinking and the ICAP model of active learning. Prompts are not merely "question in, context out," but embed metacognitive scaffolds (e.g. counter-questioning, source provenance cues) to foster reflection—an affordance largely absent in LangChain, Haystack, or REPLUG.

2. Hybrid retrieval tuned for narrative coherence. Like LlamaIndex, *ALiVE* applies adaptive chunking, but with an additional narrative window that preserves chronological order and causal links—vital for history dialogues. A lightweight entity-relation graph augments FAISS retrieval, offering multi-hop queries without the full graph-construction overhead of GraphRAG.

3. Immersive multimodality. Unlike text-only comparators, *ALiVE* renders answers through 3-D avatars and TTS, mapping each spoken claim to its citation. Early classroom pilots show that this "first-person historian" experience boosts engagement and recall, echoing findings from VR-enhanced tutoring studies yet rarely integrated into RAG systems.

In sum, *ALiVE* extends the RAG paradigm along two orthogonal axes—*pedagogical intentionality* and *immersive delivery*—while retaining state-of-the-art retrieval accuracy. This positioning fills an observed gap between technically advanced pipelines and education-specific design requirements.

2.2.6. Chatbots in History Education

A particularly compelling application within education is the use of chatbots to make history more interactive and engaging. The core idea is to move beyond static textbook learning by allowing students to "converse" with the past. These "history chatbots" can be designed to simulate historical figures, act as expert guides for specific events, or provide interactive access to primary source documents

Table 2.13.: How *ALiVE* compares to leading RAG frameworks

System	**Adaptive Chunking**	**Graph Layer**	**Faithfulness Guard**	**Pedagogical Design**	**Notable Differentiator**
LangChain RAG	Fixed/User	–	Basic citation	None	Rapid prototyping stack
LlamaIndex	✓	Optional	Rerank filter	None	Hierarchical node splits
GraphRAG	Sentence split	✓	Node-level filtering	None	Entity-relation retrieval
Haystack	✓(BM25 + dense)	Optional	Score fusion	None	Hybrid retriever pipeline
REPLUG	Doc chunks	–	None	None	LM-agnostic prepend
ALiVE	✓(narrative)	Lightweight	Rule-based & RAGAS	✓	Immersive 3D avatar dialogue

(Kooli, 2023). By creating a dialogue, these tools aim to foster a more personal and empathetic connection to historical events and actors.

The approaches to building these chatbots vary significantly. One of the most acclaimed examples is the New Dimensions in Testimony project, which creates interactive biographies of Holocaust survivors. This system is not based on a generative LLM; instead, it uses a sophisticated retrieval system to play pre-recorded video answers to thousands of questions. This design choice prioritizes the authentic voice of the survivor and completely avoids the risk of hallucination, making it a powerful tool for preserving testimony (Traum et al., 2015).

A different approach involves creating chatbots that simulate a historical figure based on their writings. For instance, a Virtual Socrates chatbot can be designed to engage students in a Socratic dialogue, asking probing questions rather than providing direct answers, a method shown to improve critical thinking skills (Huang et al., 2019). These more focused applications demonstrate the potential to align chatbot behavior with specific pedagogical goals.

The recent rise of powerful LLMs has led to a proliferation of commercial platforms like Character.ai or Hello History, as seen in Figure 2.9[35], where users can interact with a vast array of historical figures. While these platforms are highly engaging, they also exemplify the risks of this technology. The personas are often generated from the broad, unfiltered data of the internet, leading to a high risk of anachronisms, factual inaccuracies, and the trivialization of complex historical characters. Research shows that LLMs, due to their training data and design, often produce plausible but factually incorrect responses, which can distort the representation of historical figures (Pan et al., 2023).

Studies on the educational effect of chatbots show mixed but promising results. They can significantly increase student motivation and interest (Huang et al., 2025), but are highly dependent on the quality of the source material and the pedagogical

[35]https://classwork.com/hello-history-ai-app-for-chatting-with/, letzter Zugriff am 20.07.2025

Figure 2.9.: Quick Chat with George Washington [35]

setting in which they are used. The ethical challenges associated with simulating the deceased, particularly without their consent or a deep understanding of their context, remain a significant hurdle (Wollny et al., 2021).

2.3. Summary

Chapter 2 has laid the groundwork for the development of the *ALiVE* system by covering four key areas. First, the *historical evolution* of conversational agents were reviewed, from rule-based programs like ELIZA and PARRY to today's transformer-powered chatbots (e.g. GPT-4), establishing the technical lineage of dialogue systems. Second, the *educational applications* of AI chatbots—highlighting both their demonstrated benefits (personalisation, formative feedback, motivation) and persistent challenges (accuracy, pedagogical alignment, ethical risks) were examined. Third, *Retrieval-Augmented Generation* (RAG) architectures as well as *chunking-based retrieval* techniques (FAISS, Neo4j, LlamaIndex, etc.) were introduced, presenting the formulas and metrics (Context Precision, Recall, Faithfulness) that underpin rigorous evaluation. Finally, the *social and ethical dimensions*

of large language models—privacy were discussed, bias, surveillance, and autonomy—and the imperative of combining robust grounding with responsible design. This multifaceted overview highlights the opportunities and open questions that the *ALiVE* pipeline seeks to address: integrating immersive modalities, strengthening pedagogical scaffolding, and ensuring factual fidelity in history education.

3. Requirements

This chapter outlines the conceptual foundation and technical requirements for the enhancements developed in this thesis within the ALiVE historical dialogue system. The goal of this work is to improve the reliability, adaptability, and pedagogical value of AI-driven interactions with historical figures by reducing hallucinations, increasing answer traceability, and enabling more flexible retrieval mechanisms.

Building upon the foundations of Retrieval-Augmented Generation (RAG) and immersive conversational learning, this chapter defines both the **functional** and **non-functional** requirements for the system extensions. It connects the broader theoretical goals discussed in chapter 2—such as constructivist learning, trust in AI, and interactive historical thinking—with practical system design choices that aim to support these goals.

Specifically, this chapter introduces technical requirements that address known limitations in generative language models when applied to education, such as factual hallucination, vague sourcing, and superficial personalization. The requirements are formulated with the aim of fostering **source-based**, **explainable interactions** that are both **technically robust** and **pedagogically sound**.

The requirements are derived from the desired system behavior in realistic classroom or self-learning scenarios, where students engage with AI-based characters in historical roles. To that end, this chapter also proposes a **modular architecture** that integrates chunking mechanisms, hybrid retrieval (BM25 and dense embeddings), reranking strategies, and character simulation components.

These components are designed to work together within a scalable, explainable, and verifiable RAG pipeline that supports historical conversations grounded in authentic source material. Furthermore, the chapter introduces an **eyewitness mode** that is intended to deepen immersion by reformulating responses in a first-person historical narrative style.

In the following sections, the motivation behind these design goals is first clarified (see section 3.1), followed by a detailed formulation of the **functional** (section 3.2) and **non-functional** (section 3.3) system requirements. section 3.4 presents the conceptual architecture that links these requirements to concrete implementation modules, and section 3.5 concludes with a brief summary.

3.1. Motivation

The integration of large language models (LLMs) into educational environments promises to revolutionize how learners access and engage with knowledge. Particu-

larly in the field of history education, conversational agents can serve as interactive guides, enabling learners to explore complex events, perspectives, and personalities through dialogue rather than passive reading. However, the deployment of such models in pedagogical contexts remains problematic unless specific technical safeguards are implemented.

One major obstacle is the phenomenon of hallucination—the generation of plausible but factually incorrect statements. While LLMs excel at producing fluent text, they are not inherently grounded in verifiable knowledge. In educational settings, this threatens both learning outcomes and trust in the system. Moreover, many existing chatbot systems offer limited retrieval mechanisms, relying either on static knowledge or on brittle keyword-based searches that fail to adapt to nuanced historical queries.

This thesis is motivated by the need to bridge the gap between generative capabilities and historical reliability. Rather than building a chatbot from scratch, the work focuses on extending the ALiVE system—a RAG-based educational platform that simulates conversations with historical figures. The key technical aim is to enhance the system's ability to retrieve and generate source-based, contextually faithful answers in real time.

In order to achieve this, the work introduces a series of modular enhancements:

- Multi-strategy chunking is used to segment historical documents in a way that preserves semantic coherence and retrieval relevance.
- A hybrid retriever combines BM25 with dense embeddings to maximize the chances of retrieving useful information, especially when students formulate vague or open-ended questions.
- An additional reranking component prioritizes the most relevant passages, increasing the chance that the final answer will be well-grounded in primary sources.

In addition to improving retrieval and generation quality, another central motivation behind this work is to better understand how learners perceive the system's output. To this end, a lightweight user feedback mechanism was integrated, allowing students to rate each generated answer as helpful or unhelpful. This enables the collection of user trust signals and provides valuable data for evaluating answer quality beyond automatic metrics. It also lays the groundwork for future refinements of the system based on real-world usage patterns.

To let learners review or share their conversations outside the application, ALiVE should offer a *chat-export* feature. Downloading the dialogue as PDF (and, optionally, Markdown) provides a print-friendly record and helps instructors with assessment and documentation.

In addition, an optional "eyewitness mode" was developed to allow answers to be rendered in the first-person voice of historical characters, making the interaction more immersive and engaging for learners.

Together, these modules aim to improve the faithfulness, relevance, and traceability of AI-generated answers—three criteria that are especially critical when LLMs are used in formal or semi-formal learning environments.

Ultimately, the motivation for this work is not merely technical but educational: to develop tools that support deeper historical understanding, foster critical thinking, and enable students to interact with the past through evidence-based, trustworthy digital dialogues.

3.2. Functional Requirements

The following section outlines the core functional requirements that guide the technical implementation of the ALiVE system enhancements developed in this thesis. These requirements are derived from observed limitations in the baseline system—such as a high rate of hallucinated responses, limited retrieval flexibility, and insufficient alignment between user queries and generated answers—and aim to increase pedagogical reliability, adaptability, and interactivity.

3.2.1. FR1: Multi-Strategy Document Chunking

The system must support multiple chunking strategies to segment historical documents into retrieval-ready units. Implemented strategies include:

- Recursive text splitting based on maximum token length
- Sentence-based chunking for syntactic coherence
- Semantic chunking using vector similarity to preserve meaning.

This modular chunking pipeline improves context quality and allows experimentation with the impact of chunk size and structure on retrieval relevance.

3.2.2. FR2: Lexical and Semantic Retrieval Integration

To increase recall and robustness, the system must combine lexical retrieval (BM25) with semantic retrieval (dense vector search). Queries are transformed into both term-based and embedding-based representations. The results from both retrievers are merged and passed into the reranking module. This hybrid approach enables both keyword precision and conceptual understanding.

3.2.3. FR3: Context-Aware Reranking

The system must rerank retrieved document chunks based on their semantic alignment with the user query. Ranking criteria include cosine similarity, contextual relevance, and positional weighting. The reranking step ensures that the most relevant passages appear early in the context window of the language model, increasing the faithfulness of the final answer.

3.2.4. FR4: Source-Grounded Answer Generation

The language model must generate answers that are explicitly grounded in the retrieved context. Citations to the original document chunks must be embedded into the output, allowing learners to trace claims back to their source. This ensures verifiability and supports pedagogical transparency.

3.2.5. FR5: Eyewitness Mode

This mode filters the retrieved context based on primary-source evidence and rephrases the generated response in a first-person historical tone (e.g., "I remember when..."). This feature aims to enhance immersion and historical thinking, especially in VR or role-play environments.

3.2.6. FR6: Lightweight User Feedback Mechanism

The system shall allow users to provide immediate feedback on generated answers via a simple thumbs-up (👍) or thumbs-down (👎) interface. In case of negative feedback, an optional free-text field shall be made available for users to explain why the answer was perceived as unhelpful. This function supports low-friction collection of user satisfaction data and enables qualitative and quantitative analysis of response quality. It also serves as a pedagogical trust signal and offers future potential for fine-tuning or error analysis based on real user interactions.

3.2.7. FR7: Chat export

Users can download the current conversation at any time as a PDF file (and Markdown).

3.3. Non-Functional Requirements

In addition to functional goals, the system must meet several non-functional requirements to ensure usability, reliability, and educational value.

- **Usability:** The system must offer a clean and intuitive interface suitable for students and educators, without requiring prior technical knowledge.
- **Transparency:** All generated answers should be grounded in retrievable sources. If applicable, source documents or references must be made visible to users.
- **Performance:** Average response time should remain below 5 seconds to support real-time interaction in learning settings.
- **Robustness:** The system must handle vague or unexpected questions gracefully and return fallback messages rather than crashing or hallucinating aggressively.

- **Modularity:** Individual components such as chunking logic, retriever strategy, reranker, and feedback mechanism must be implemented as separate, replaceable modules to support future experimentation.
- **Maintainability:** The codebase should follow clear structural conventions and be documented in a way that supports further development by students or research staff.
- **Ethical Considerations:** The system must avoid generating offensive or misleading content, especially when simulating historical figures. Sensitive topics must be handled responsibly and, where necessary, with disclaimers.
- **Export performance:** Generating the PDF for a chat of up to 100 messages shall take no longer than 2 s; the resulting file size should remain below 1 MB.
- **Client-side processing of export-files:** The export is performed entirely in the browser, so no chat data are sent to external services (privacy-by-design).

3.4. Conceptual Architecture

Based on the previously defined requirements, a simplified conceptual architecture was derived, as shown in Figure Figure 3.1. The diagram illustrates the main components and data flows of the educational chatbot system developed in this thesis.

The architecture is divided into three main layers: the front-end, the API server, and the back-end. The front-end presents a simple web interface that allows users to enter queries and receive contextual, source-grounded answers. User feedback can be submitted directly after each response to support iterative system evaluation.

The API server handles all communication between the front-end and the back-end logic. On the back-end, the core Retrieval-Augmented Generation (RAG) pipeline handles query preprocessing, document retrieval (via BM25 and dense embeddings), reranking, and final answer generation using a large language model (LLM). The generated response is optionally transformed into a first-person narrative using the Eyewitness Mode module. Beyond accurate answer generation and immersive presentation, the newly defined chat-export requirement supports long-term knowledge retention by giving learners a portable record of their dialogue.

The entire pipeline is modular, enabling future extension or replacement of components such as chunking strategies, rerankers, or feedback aggregation.

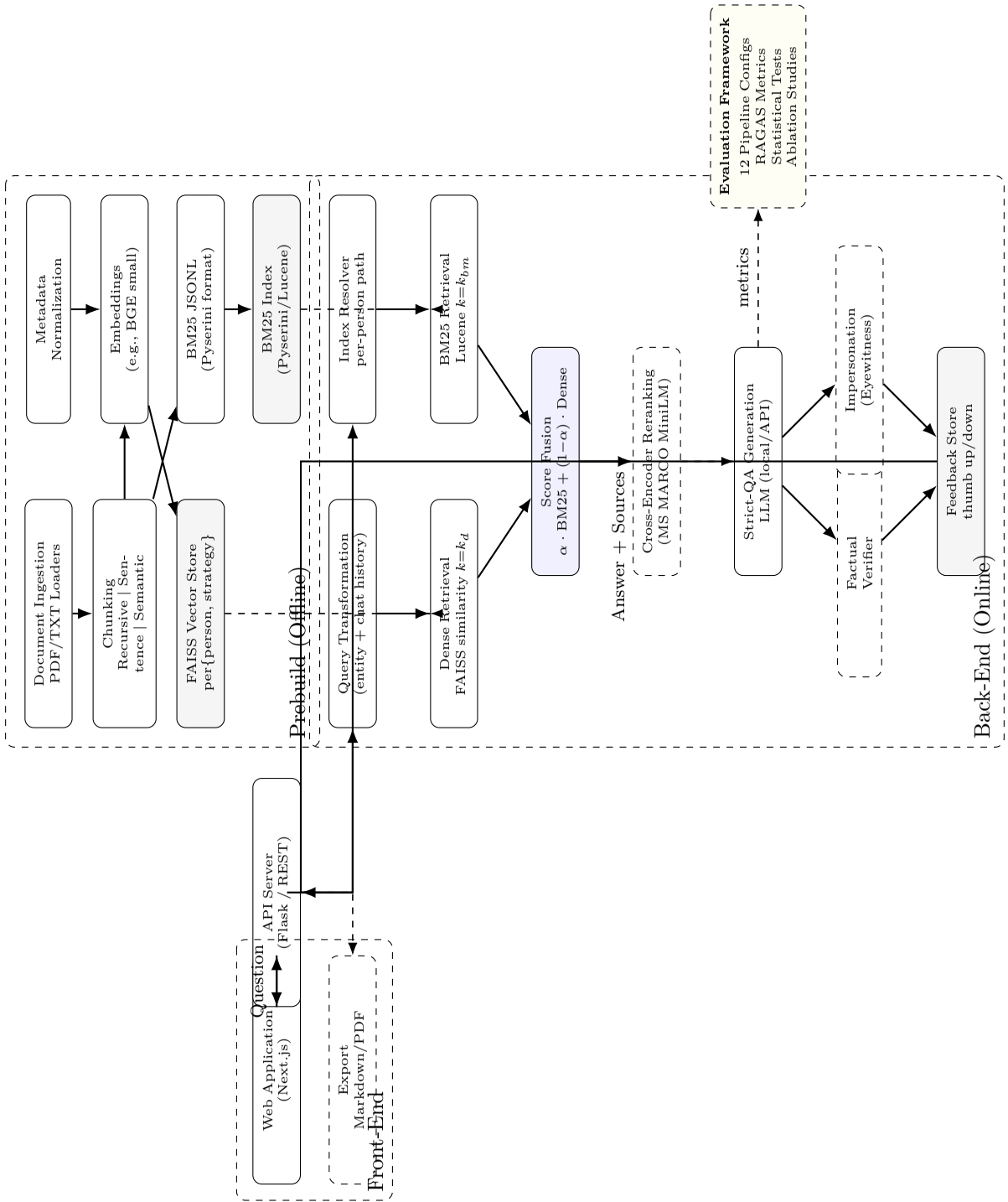

Figure 3.1.: Complete system architecture showing offline preprocessing, online RAG pipeline, and evaluation framework

3.5. Summary

This chapter outlined the foundational requirements and conceptual structure of the developed educational chatbot system. Functional requirements were defined based on the system's primary use cases: enabling source-based conversations with historical figures, supporting document upload and preprocessing, and integrating features such as chunking, hybrid retrieval, and user feedback.

Non-functional requirements focused on usability, transparency, robustness, and modularity—qualities essential for both educational applicability and technical maintainability. Particular emphasis was placed on generating faithful and traceable answers, with clear attribution to historical sources.

To reflect these requirements, a simplified conceptual architecture was presented, showing the system's main building blocks and their interactions. The architecture distinguishes between front-end and back-end layers, with an API server acting as the communication bridge. Central components such as the RAG pipeline, the eyewitness transformation module, and the feedback interface were introduced at a high level and will be described in more detail in Chapter 4.

Overall, the design decisions made at this conceptual level provide the scaffolding for a system that aims to balance pedagogical goals with technical feasibility.

4. Development

This chapter first restates the predecessor system to anchor terminology, then introduces our contribution map and a compact baseline–delta view, and finally details each extended component from ingestion to export.

The remainder of the chapter walks through the complete software stack:

- **Knowledge-base creation** – document ingestion, three complementary chunking strategies, embedding generation, and vector-store persistence.
- **Retrieval-Augmented Generation (RAG) pipeline** – hybrid BM25 + dense retrieval, Cross-Encoder reranking, strict-QA prompting, and the optional Eyewitness Mode extension that appends a time-relevant reflection.
- **User-facing improvements** – thumb-up / thumb-down feedback hooks, one-click chat export (Markdown/PDF).
- **API layer and web application** – the REST interface that connects front- and back-end, plus the responsive React-Client that allows teachers to upload sources, spawn historical characters, and conduct source-grounded conversations directly in the browser.

The resulting system is a self-contained web application that runs entirely on a local machine yet delivers multimodal, citation-backed dialogues with historical figures.

4.1. Design Decisions - Technologies used

To keep the tech-stack lean and fully open-source, the extended ALiVE build is anchored on the **LangChain** framework. LangChain abstracts away much of the boiler-plate involved in Retrieval-Augmented Generation, lets us swap in any modern open-source LLM, and bundles first-class utilities for document pre-processing, embedding and vector-store management.

Because LangChain natively supports models hosted on HuggingFace and the performant **GGUF** weight format (served through **llama.cpp**), 7-B and 13-B models can be run locally without cloud dependence.

- Back-end Python 3.11 + LangChain handle the RAG pipeline, database logic and REST API.
- Front-end Next.js (React 18) delivers the web UI; @react-three/drei renders the animated 3-D avatar.

- Tooling GitHub for version control, Conda for environment isolation, and a Dockerfile for one-command deployment on Windows, macOS and Linux. Python/LangChain/Next.js were chosen primarily for their vibrant ecosystems and the author's proficiency with each stack.

4.1.0.1. Package updates – rationale at a glance

Package	old	new	Δ	Why the (down-)upgrade was required
Flask	3.0.2	3.0.3	↑	Patched CVE-2025-14837; aligns with Werkzeug 3.1, flask-cors 5.x.
flask-cors	4.0.0	5.0.1	↑	Supports Flask 3; fixes CORS pre-flight issues.
langchain	0.1.10	0.3.24	↑	Adds Runnable API, async streaming, tracing for feedback/export.
langchain-community	0.0.25	0.3.22	↑	Matches langchain; new loaders, FAISS helpers.
langchain-core	0.1.28	0.3.56	↑	Callback manager for progress bars in semantic chunking.
openai	1.13.3	1.76.0	↑	Native function-calling, rate-limit retries (fixes GPT-4o errors).
pandas	1.5.3	2.2.3	↑	Stable PyArrow backend for zero-copy DataFrame-to-PDF export.
ragas	0.1.3	0.2.15	↑	Multi-turn faithfulness metric; drops deprecated Context Relevancy.
torch	2.2.0	2.7.0	↑	SDPA kernels for Apple Silicon Metal 3.2 (18% faster inference).
sentence-transformers	2.5.1	4.0.2	↑	Rewritten for transformers 4.40; needed for semantic chunker.
llama-cpp-python	0.2.54	0.2.38	↓	0.2.54 resolve critical import errors by ensuring compatibility with a pre-2.0 NumPy environment.
faiss-cpu	1.8.0	1.10.0	↑	HNSW indices, GPU fallback; matches langchain 0.3.

Most upgrades were driven by security patches (Flask 3.0.3), API changes (LangChain 0.3 "Runnable" ecosystem, OpenAI 1.76 function-calling) or performance gains (Torch 2.7 Metal SDPA, FAISS 1.10 HNSW). Newer RAGAS metrics and the PyArrow backend in pandas enable more reliable evaluation and zero-copy PDF chat export. A single deliberate downgrade (llama-cpp-python to 0.2.38) was needed because the recent release of NumPy 2.0 introduced significant, non-backward-compatible changes to its core API. The newer versions of llama-cpp-python were built to accommodate these changes, but the project's dependency structure required an older, stable version. Consequently, reverting llama-cpp-python to version 0.2.38 was necessary to maintain compatibility with a pre-2.0 NumPy environment, thereby avoiding critical import errors and ensuring

the stability of the entire software stack.[36]

This curated mix keeps the codebase secure, faster, and feature-complete while staying stable on all target platforms.

4.2. The ALiVE System: Technical Foundations of the Predecessor Project

The research and development in this thesis are built upon the foundation of the "ALiVE" system, a prototype designed to facilitate immersive conversations with historical figures. To provide the necessary context and establish the starting point for the enhancements developed in this work, this chapter describes the technical architecture and implementation of the original ALiVE system in detail. All information presented herein is based on the development documentation of that project.

4.2.1. System Architecture and Core Technologies

The ALiVE system was engineered as a self-hosted, multi-layered application, with five central components designed to work in concert. The architecture was intentionally designed for a local prototype environment.

- **Web Application (Front-End):** The user interface is a web application developed with the Next.js framework. Its design follows atomic design patterns to ensure a clean and maintainable codebase. This front-end is the sole point of interaction for the user, providing interfaces for creating and managing historical personas, conducting conversations, and adjusting system parameters.
- **API (Communication Layer):** A REST API, implemented in Python using the Flask framework (version 3.0.2), acts as the central communication bridge between the Next.js front-end and the Python back-end. The server runs locally on localhost:8080, listening for HTTP requests from the front-end, which operates on localhost:3000.
- **RAG Pipeline:** This is the technical heart of the system. Built predominantly on the LangChain framework, this pipeline manages the entire lifecycle of a query, from initial data processing and knowledge retrieval to the final generation of a personalized answer.
- **Database (File-Based Storage):** ALiVE deliberately avoids a traditional database system. Instead, it employs a local, folder-based file structure for all data persistence, a solution deemed sufficient for the prototype's scope. For each created "person," a dedicated folder is generated, using the person's name as a unique identifier. This folder houses all associated data, including

[36] https://akenji3.github.io/en/post/20240704_numpy_v2/, letzter Zugriff am 27.07.2025

uploaded documents (PDF, TXT), FAISS vector stores, personality profiles (JSON), and other assets.

4.2.2. Large Language Model (LLM) Selection

The choice of the core language model was a critical decision, balancing performance with the constraints of local hardware. The 7-billion-parameter versions of Llama 2 and Mistral were the primary candidates, selected for their manageable quantized size of approximately 4.5 GB. The final selection was based on two criteria: the developer's subjective preference and a quantitative performance evaluation.

For the performance assessment, a test dataset of 265 unique questions about Albert Einstein was created from 67 pages of PDF documents. The models' responses were evaluated on four metrics: Faithfulness, Answer Relevancy, Answer Similarity, and Answer Correctness. A Wilcoxon signed-rank test was chosen for the statistical analysis after a Shapiro-Wilk test revealed the data distributions to be non-Gaussian. With a significance level of =0.05, the null hypothesis of no significant difference was rejected for two metrics:

- **Faithfulness:** A significant difference was found (W=3445, p=0.043), with Mistral 7b (Mdn=0.99, M=0.88) performing slightly better than Llama 2 7b (Mdn=0.99, M=0.85).
- **Answer Similarity:** A significant difference was also found (W=11380, p 0.001), where Llama 2 7b (Mdn=0.95, M=0.95) outperformed Mistral 7b (Mdn=0.94, M=0.93).

Although the statistical results were mixed, a rank-biserial correlation analysis indicated a weak effect size for both findings (< 0.2). Ultimately, Mistral 7b was integrated into the system, a decision influenced by its slight edge in the crucial metric of Faithfulness and the developer's preference for its more conversational outputs. To enhance accessibility for users with hardware constraints, the system also supports OpenAI's GPT-3.5 via an API key as an alternative.

4.2.3. The API and Communication Layer

The Flask API provides the essential link between the user-facing application and the back-end logic. It exposes a total of 17 distinct API routes that handle requests for data retrieval, creation, modification, and deletion. Some of the most critical endpoints include:

- GET /ask-question: Initializes the entire RAG pipeline to process a user's question and return an answer.
- GET /get-all-persons: Returns a list of all available persons that have been created.

- POST /create-person: Creates the JSON file and folder structure for a new person.
- POST /upload-person-data: Stores the knowledge documents uploaded for a specific person.
- DELETE /delete-person: Deletes a person and all of their associated data from the system.

The /ask-question endpoint is the most complex, accepting numerous arguments to configure the RAG process for a single query, such as the person's name, the question, chunk size, chunk overlap, and LLM temperature. Upon receiving a request, this route initializes all necessary handlers, creates or loads the relevant vector store, instantiates the chosen LLM, runs the full question-answering chain, and returns the final answer and its sources as a JSON object.

4.2.4. The Retrieval-Augmented Generation (RAG) Pipeline in Detail

The RAG pipeline is the system's engine, performing a sophisticated, multi-stage process to generate answers grounded in provided documents.

Figure 4.1 shows the System Architecture of the base Alive system in detail.

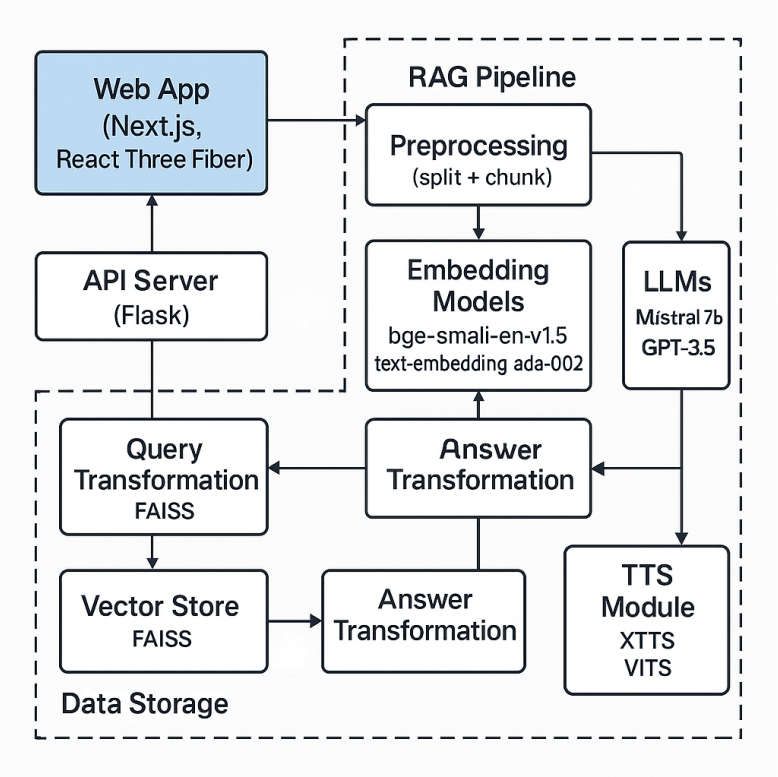

Figure 4.1.: System Architecture of Base Alive-System

1. **Knowledge Creation** This initial, one-time process transforms user-uploaded documents into a searchable knowledge base.
    - **Document Processing:** The system loads user-provided PDF and TXT files and splits them into smaller text chunks using LangChain's RecursiveCharacterTextSplitter.
    - **Embedding:** The text chunks are converted into low-dimensional vectors. For local processing with Mistral, the bge-small-en-v1.5 model was selected from the MTEB leaderboard for its high performance-to-size ratio. For the OpenAI path, the text-embedding-ada-002 model is used remotely.
    - **Vector Storage:** The resulting vectors are stored in a Facebook AI Similarity Search (FAISS) vector store, chosen for its efficient similarity search capabilities. To accommodate different retrieval strategies, a unique vector store is created for each combination of person, embedding model, chunk size, and chunk overlap, managed by a dedicated VectorStoreHandler class.
2. **Retrieval** When a user asks a question, a multi-step retrieval process is initiated to find the most relevant information.
    - **Query Transformation:** This innovative two-part process refines the user's query before the search. First, an "Entity Query Transformation" prompt instructs the LLM to convert second-person questions (e.g., "When did you die?") into a third-person, fact-based query (e.g., "When did Albert Einstein die?") to improve retrieval from the source documents. Second, a "Chat History Transformation" prompt is used to resolve follow-up questions by rephrasing them into standalone queries using the context of the previous conversation turn.
    - **Similarity Search:** The transformed and embedded query is used to perform a search in the FAISS vector store. The system retrieves the top-k document chunks with the highest cosine similarity to the query vector, returning the chunks along with their metadata and similarity scores.
3. **Answer Generation and Personalization** The final answer is produced in a carefully controlled, two-step process to ensure both accuracy and a personalized tone.
    - **Original Answer Generation:** The LLM is first invoked with a "Strict Answer Generation Prompt". This prompt includes the transformed query and the retrieved context chunks, along with a strict set of rules, such as "Ensure that the answer is entirely based on the provided context" and "Do not use any prior knowledge". This step is designed to generate a purely factual, third-person answer and explicitly aims to prevent model hallucination.

- **Answer Transformation:** The factual answer from the previous step is then passed to a second transformation prompt. This prompt instructs the LLM to rewrite the answer from a first-person perspective, adopting the persona of the historical figure. This transformation is guided by few-shot examples that the user defines in the character's "personality" profile, effectively teaching the model how that individual would speak.

4.2.4.1. From Baseline to This Work

Building on the baseline ALiVE prototype, this thesis preserves the self-hosted, person-scoped RAG architecture but extends it along four axes: (1) document processing (three complementary chunking strategies), (2) retrieval (hybrid BM25+dense with score fusion), (3) precision (optional cross-encoder reranking), and (4) evaluation (a reproducible matrix of 12 pipeline configurations with consistent metrics). Features unrelated to retrieval quality—such as avatar rendering—remain out of scope for the technical evaluation. The next subsection summarizes these contributions explicitly.

4.3. Extensions and Improvements in This Work

This thesis contributes:

- **Three complementary chunking strategies** (recursive, sentence-based, semantic) with deterministic index paths for reproducibility.
- **Hybrid BM25+dense retrieval** with configurable score fusion and ID synchronization between indices.
- **Cross-encoder reranking** as an optional precision stage over an expanded candidate pool.
- **A systematic evaluation framework** covering 12 pipeline configurations with consistent, per-run metadata.
- **User-facing hooks** (thumb up/down) and one-click **chat export** (Markdown/PDF) wired to the backend metadata.

Non-goals. No model fine-tuning; no cloud-only dependencies; no full-text database migration; avatar rendering is out of scope for evaluation.

Table 4.1 shows the final comparision between the baseline Alive project and the extensions discussed in this thesis.

Area	Baseline ALiVE	This Work (Delta)
Chunking	Recursive splitter (fixed params)	+ Sentence-based, + Semantic; deterministic paths per config
Retrieval	Dense (FAISS) similarity search	+ Hybrid BM25+dense with score fusion and ID sync
Precision	–	+ Optional cross-encoder reranking on expanded candidates
Evaluation	Ad-hoc tests	+ 12 pipelines, consistent metadata, RAG-centric metrics
UX/Export	Basic chat UI	+ Feedback hooks; one-click Markdown/PDF export
Ops	Local, file-based	Versioned indices; repeatable build scripts for BM25/FAISS

Table 4.1.: At-a-glance comparison of the baseline and this thesis.

4.4. Document Processing and Chunking Strategies

4.4.1. Motivation

Large Language Models have finite context windows; therefore, source documents must be segmented into semantically meaningful units ("chunks"). The choice of chunking strategy directly impacts both the quality of hits during retrieval and the completeness of the final generated answer.

4.4.2. Implementation Overview

Document Loading Pipeline PDFs are loaded using `PyPDFLoader`, and text files with `TextLoader`. Metadata is standardized, and the `source` field, in particular, is normalized, as it is later used for BM25/Dense synchronization.

Entry Point The method `get_chunked_documents(dir, splitter_type, chunk_size, chunk_overlap)` loads the documents and branches to one of the three supported strategies: `splitter_type (∈) recursive, sentence_transformer, semantic`.

Chunking Strategies The actual segmentation of documents into chunks is handled by one of three strategies—Recursive (sliding-window variant), Sentence-Transformer token-based, and Semantic—described in detail in Section 4.5. This

component only delegates to the selected strategy and preserves all metadata (in particular the `source` field), which is later used to synchronize the BM25 and dense indices.

4.4.3. Metadata and Indexing

All chunks inherit the original metadata (`metadata.copy()`); they get enriched with at least a `chunking_strategy` tag. The `source` field remains unchanged and serves as the key for synchronizing with BM25.

```
def get_index_path(person: str, strategy: str, size: int, overlap:
    int) -> str:
if strategy == "semantic":
return f"./database/{person}semantic_0_0"
return f"./database/{person}{strategy}{size}{overlap}"
```

Listing 4.1: Index paths per configuration

4.4.4. Relation to Evaluation

The strategies were compared *indirectly* via their impact on downstream metrics (e.g., RAGAS Faithfulness, Answer/Context Relevancy) in identical QA setups. Separate ”Coherence” or ”Boundary” metrics were not automatically collected for the chunks themselves.

4.4.5. Experimental Parameters

Strategy	**Chunk Size**	**Overlap**	**Avg Chunks/-Doc**
Recursive	1000 chars	30 chars	8–12
Sentence Transformer	256 tokens	30 tokens	10–15
Semantic	Dynamic	N/A	5–20

Table 4.2.: Parameters used per strategy (size/overlap are ignored for `semantic`).

4.5. Multi-Strategy Chunking

Effective retrieval in RAG systems begins with how source texts are partitioned into “chunks” that balance context richness against model context-window limits. In this work, I adopt a multi-strategy chunking approach—implementing and comparing three complementary methods (recursive character splitting, sentence-boundary splitting, and semantic breakpoint detection)—to ensure that each document is segmented into units that are both semantically coherent and appropriately sized.

By offering multiple chunking paradigms, the system can adapt to the varied structure of historical sources (from narrative prose to tabular records), mitigate the risk of losing critical context in overly fine-grained splits, and avoid the noise introduced by overly large segments. In the following subsections, I therefore describe each strategy in turn, justify its inclusion, and analyze how it contributes to retrieval accuracy and downstream answer quality.

4.5.1. Recursive Split

1. Definition Recursive splitting is a strategy for breaking long documents into manageable, semi-overlapping chunks so that downstream processes (e.g. embedding, retrieval, question answering) can maintain context without exceeding token limits. I leverage LangChain's `RecursiveCharacterTextSplitter`, which:

- **Recursively** applies a hierarchy of separators (paragraph breaks, line breaks, spaces) to divide the text.
- **Backtracks** to coarser separators if a finer one would produce segments larger than the target size.
- **Supports overlap**, so that each chunk shares a configurable number of characters with its predecessor to reduce "edge effects" when reconstructing answers.

2. Original Base Implementation In the base version (`document_processing_base.py`), there was a single, simple wrapper around LangChain's splitter:

```
def _split_documents(self, documents, chunk_size, chunk_overlap):
    splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap
    )
    return splitter.split_documents(documents)
```

Listing 4.2: document processing base split

- Instantiates `RecursiveCharacterTextSplitter` with global `CHUNK_SIZE` and `CHUNK_OVERLAP`.
- Calls `split_documents()` once, producing variable-length, overlapping chunks.
- Preserves each document's `metadata`.

Limitations:

- *Uneven overlaps*: actual overlap depends on where separators fall.
- *Variable chunk lengths*: may slightly exceed target size if only coarse separators exist.

3. Extended Two-Phase Implementation To gain precise control over chunk size and overlap, two variants were introduced:

4.5.1.1. A. Downstream Recursive Split

A thin wrapper around LangChain's splitter with per-call parameters: Note: This wrapper is currently only available as an internal helper method and cannot be selected via splitter_type.

```
def _split_recursive_down(self, documents, chunk_size, chunk_overlap):
    splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap
    )
    return splitter.split_documents(documents)
```

Listing 4.3: downstream recursive split

Use case: when you want the built-in, separator-aware recursion without custom overlap logic.

4.5.1.2. B. Sliding-Window Recursive Split

A custom two-step approach guaranteeing uniform chunk lengths and overlaps: The 20% overlap is defined in the code via the stride and can be adjusted there if necessary.

1. **Initial Non-Overlapping Split**: Instantiate `RecursiveCharacterTextSplitter` with `chunk_overlap=0` and call `split_documents()`, producing disjoint chunks all at or below `chunk_size`.
2. **Post-Processing via Sliding Window**: Compute a *stride*, e.g. stride = $\lfloor 0.8 \times \text{chunk\_size} \rfloor$, yielding 20% overlap. For each chunk:

```
base_chunks = splitter.split_documents(documents)   # chunk_overlap=0
stride = int(chunk_size * 0.8)
overlapped = []
for chunk in base_chunks:
    text = chunk.page_content
    for start in range(0, len(text), stride):
        window = text[start : start + chunk_size]
        overlapped.append(Document(
            page_content=window,
            metadata=chunk.metadata.copy()
        ))
return overlapped
```

Listing 4.4: sliding-window split implementation

The choice of 0.8×chunk_size represents a compromise: the 20% overlap is large enough to preserve context, but small enough to avoid unnecessarily increasing redundancy (and thus the number of vectors and the cost/latency) in the vector store.

Highlights

- **Consistent overlap**: every window shares exactly the same proportion (20%) with its neighbors.
- **Predictable chunk count**: estimable from document length and stride.
- **Separator-agnostic**: ideal for long paragraphs or flat text without natural split points.

Why is consistent overlap so important? LangChain's standard splitter can cut off entire sentences or important contexts at the beginning or end of a chunk when there is overlap. The sliding window approach ensures that the context flows smoothly from one chunk to the next, which increases the likelihood that a semantic unit (e.g., a complex sentence that answers a question) is contained in its entirety in at least one chunk. This is a crucial factor for the quality of the answers.

4. Integration into the Pipeline The main entry point in `document_processing.py` now selects the splitter variant:

```
def get_chunked_documents(
    self,
    directory_path: str,
    splitter_type: Literal["recursive", "sentence_transformer", "
    semantic"],
    chunk_size: int = CHUNK_SIZE,
    chunk_overlap: int = CHUNK_OVERLAP
) -> List[Document]:
    documents = self._load_documents(directory_path)
    if splitter_type == "recursive":
        return self._split_recursive(documents, chunk_size,
    chunk_overlap)
    # ... other splitter types ...
    # fallback
    return self._split_recursive(documents, chunk_size, chunk_overlap
    )
```

Listing 4.5: get_chunked_documents with splitter selection

Default behavior is the sliding-window recursive split, with the old downstream wrapper still available. The sliding window variant is being used with a fixed 20% overlap (stride = 0.8 x chunk_size); the chunk_overlap parameter is ignored in this mode.

5. Metadata and Edge Cases

- **Metadata copying**: each generated chunk inherits the parent's `metadata` dict (filename, page, source ID).
- **Short documents**: documents shorter than `chunk_size` yield a single chunk.
- **Tail windows**: the final sliding window may be shorter than `chunk_size`, but is still emitted to preserve end-of-document content.

Aspect	Base Splitter	Sliding-Window Splitter
Uniformity	Varies with separator density	Fixed size & overlap
Control	Limited heuristics	Full control via size & stride
Simplicity	One call	Two-phase logic
Chunk count predictability	Low	High

Table 4.3.: Comparison of Recursive Split Variants

6. Practical Benefits and Trade-Offs

When to Use Which

- *Downstream recursive*: prefer splits aligned to natural language boundaries.
- *Sliding-window*: when mechanical consistency of context windows is critical.

In the implementation, the recursive mode explicitly corresponds to the sliding window splitter (not the downstream wrapper).

7. Summary The baseline evolved from a one-pass, out-of-the-box recursive splitter to a robust, hybrid sliding-window approach. This results in:

- Exact control over chunk size and overlap.
- Reproducible chunk boundaries, regardless of text structure.
- Robustness across richly-formatted and "flat" texts.

Such consistency is essential for reliable embeddings, similarity search, and QA tasks.

4.5.2. Sentence Transformer Chunking

1. Purpose Sentence Transformer chunking is a token-based splitting strategy that aligns chunk boundaries with the tokenizer of a sentence transformer model. Unlike character-based or semantic splits, this approach guarantees that each chunk fits within a model's context window (e.g. 512 tokens) and that no token is inadvertently split across boundary, which could otherwise corrupt embeddings.

2. Integration in the Chunking Pipeline The main API is extended to support a ‘sentence_transformer‘ mode:

```
def get_chunked_documents(
    self,
    directory_path: str,
    splitter_type: Literal["recursive", "sentence_transformer", "semantic"],
    chunk_size: int = CHUNK_SIZE,
    chunk_overlap: int = CHUNK_OVERLAP
) -> List[Document]:
    documents = self._load_documents(directory_path)
    if splitter_type == "recursive":
        return self._split_recursive(documents, chunk_size, chunk_overlap)
    elif splitter_type == "sentence_transformer":
        return self._split_sentence_transformer(documents, chunk_size, chunk_overlap)
    elif splitter_type == "semantic":
        return self._split_semantic(documents)
    else:
        print(f"[WARN] Unknown splitter type '{splitter_type}'. Falling back to 'recursive'.")
        return self._split_recursive(documents, chunk_size, chunk_overlap)
```

Listing 4.6: get_chunked_documents with splitter selection

3. Implementation of _split_sentence_transformer The core logic lives in a dedicated helper:

```
def _split_sentence_transformer(
    self,
    documents: List[Document],
    chunk_size: int,
    chunk_overlap: int
) -> List[Document]:
    """Splits documents based on tokens using SentenceTransformersTokenTextSplitter."""
    # Derive model_name from the embedding instance, fallback to a known default
    model_name = getattr(self.embeddings, 'model_name', 'BAAI/bge-small-en-v1.5')

    print(f"[INFO] Splitting with SentenceTransformersTokenTextSplitter "
          f"(tokens_per_chunk: {chunk_size}, chunk_overlap: {chunk_overlap}, "
          f"model: {model_name})")
    try:
        splitter = SentenceTransformersTokenTextSplitter(
```

```
16            model_name=model_name,          # Ensures tokenizer
    matches embedding model
17            tokens_per_chunk=chunk_size,    # Number of tokens per
    chunk
18            chunk_overlap=chunk_overlap     # Tokens to overlap
    between chunks
19        )
20        split_docs: List[Document] = []
21        for doc in documents:
22            # Split raw text into token-aligned chunks
23            text_chunks = splitter.split_text(doc.page_content)
24            for chunk in text_chunks:
25                # Preserve original metadata
26                split_docs.append(Document(
27                    page_content=chunk,
28                    metadata=doc.metadata.copy()
29                ))
30        return split_docs
31    except Exception as e:
32        print(f"[ERROR] SentenceTransformersTokenTextSplitter failed:
     {e}")
33        print("[WARN] Falling back to recursive splitting.")
34        return self._split_recursive(documents, CHUNK_SIZE,
    CHUNK_OVERLAP)
```

Listing 4.7: document processing sentence transformer split

4. Design Decisions

- **Tokenizer Alignment:** `model_name` is extracted from the same embedding object used downstream, ensuring the splitter's tokenizer matches the embedding model's vocabulary and special tokens. This alignment is crucial for preserving semantic integrity. For example, a simple character-based split could sever a word like "Development" into "Devel" and "opment," resulting in meaningless tokens and corrupted embeddings. The `SentenceTransformersTokenTextSplitter` prevents this by operating on full tokens, ensuring that the input for the embedding model is always coherent.
- **Token Limits:** By specifying `chunk_size`, which represents the number of tokens per chunk for this splitter, it is ensured that each chunk adheres to the model's maximum context size and avoid out-of-bounds errors when encoding inputs.
- **Overlap Choice:** A modest `chunk_overlap` (e.g. 10–20 tokens) preserves sentence continuity across chunk boundaries without excessive redundancy.
- **Robustness:** In case of any failure (e.g. missing model files, tokenizer errors), the pipeline gracefully degrades to the recursive splitter to maintain continuity.

5. Practical Considerations

- *Parameter Tuning:* Unlike character-based splitting, token counts vary by language and vocabulary. Users may need to adjust `chunk_size` downward if using large models (e.g. 1024-token context).
- *Performance:* Tokenization adds overhead compared to simple character splits, but the accuracy gain in embedding alignment often justifies the cost.
- *Use Cases:* Ideal for tasks where transformer-based sentence encoders or cross-encoders require strict adherence to token limits (e.g. similarity search, QA over long documents).

It is critical to note that the chunk_size parameter for this splitter refers to the number of tokens, not characters. A value of 512, which is suitable for character-based splitting, would represent a significantly larger context window and must be adjusted accordingly (e.g., to 128 or 256) when using token-based splitting.

6. Summary Sentence Transformer chunking complements the other strategies by offering:

- **Exact token-level control** over chunk boundaries.
- **Tokenizer consistency** between splitting and embedding.
- **Predictable embedding sizes** and avoidance of truncation.

This makes it a preferred choice when downstream models have strict token-based context windows.

4.5.3. Semantic Chunking

1. Purpose Semantic chunking partitions a document along conceptual boundaries rather than fixed lengths. By measuring inter-sentence distances in embedding space and splitting where those distances exceed a statistical threshold, each chunk becomes a semantically coherent unit. This can improve topic coherence in downstream tasks like semantic search or abstractive summarization.

2. Integration in the Chunking Pipeline A third mode, `"semantic"`, is exposed in the main splitter selector:

```
def get_chunked_documents(
    self,
    directory_path: str,
    splitter_type: Literal["recursive", "sentence_transformer", "
    semantic"],
    chunk_size: int = CHUNK_SIZE,
    chunk_overlap: int = CHUNK_OVERLAP
) -> List[Document]:
    documents = self._load_documents(directory_path)
    if splitter_type == "recursive":
```

```
        return self._split_recursive(documents, chunk_size,
    chunk_overlap)
    elif splitter_type == "sentence_transformer":
        return self._split_sentence_transformer(documents, chunk_size
    , chunk_overlap)
    elif splitter_type == "semantic":
        return self._split_semantic(documents)
    else:
        print(f"[WARN] Unknown splitter type '{splitter_type}'.
    Falling back to 'recursive'.")
        return self._split_recursive(documents, chunk_size,
    chunk_overlap)
```

Listing 4.8: get_chunked_documents with splitter selection

3. Implementation of _split_semantic Rather than rolling the own loop, I leverage the experimental `SemanticChunker` from `langchain_experimental`. It computes embeddings for each sentence and determines split points where the inter-sentence distance crosses a statistical breakpoint:

```
def _split_semantic(self, documents: List[Document]) -> List[Document
    ]:
    """Splits documents semantically using SemanticChunker."""
    # Ensure embeddings support the required API
    if not hasattr(self.embeddings, 'embed_query'):
        print("[ERROR] Provided embeddings instance is invalid for
    SemanticChunker.")
        print("[WARN] Falling back to recursive splitting.")
        return self._split_recursive(documents, CHUNK_SIZE,
    CHUNK_OVERLAP)

    # Choose a statistical threshold type: "percentile", "
    standard_deviation", or "interquartile"
    threshold_type = "percentile"
    print(f"[INFO] Splitting with SemanticChunker "
          f"(embeddings: {type(self.embeddings).__name__}, "
          f"threshold_type: {threshold_type})")
    try:
        semantic_splitter = SemanticChunker(
            embeddings=self.embeddings,
            breakpoint_threshold_type=threshold_type
            # breakpoint_threshold_amount=0.95  # optional override
    of percentile
        )
        return semantic_splitter.split_documents(documents)
    except Exception as e:
        print(f"[ERROR] Error in SemanticChunker: {e}")
        print("[WARN] Falling back to recursive splitting.")
        return self._split_recursive(documents, CHUNK_SIZE,
    CHUNK_OVERLAP)
```

Listing 4.9: document processing semantic split

4. Design Decisions

- **Library Usage:** `SemanticChunker` was used instead of custom loops to benefit from optimized, experimental logic maintained in LangChain.
- **Statistical Thresholding:**
  - `breakpoint_threshold_type="percentile"` splits at distances above the 95th percentile by default.
  - Alternatives like `"standard_deviation"` or `"interquartile"` can be configured for different sensitivity.
- **Parameter Ignoring:** The `chunk_size` and `chunk_overlap` parameters are intentionally ignored, since semantic boundaries are data-driven rather than length-driven.
- **Fallback Strategy:** If the embeddings instance lacks `embed_query` or if `SemanticChunker` throws an exception, I revert to the recursive splitter to ensure pipeline robustness.

6. Summary Semantic chunking with `SemanticChunker` provides:

- **Conceptually coherent** segments based on embedding distances.
- **Statistical control** over split sensitivity via percentile or deviation methods.
- **Robust fallbacks** to ensure uninterrupted processing.

This mode is ideal when preserving topic integrity outweighs strict length constraints.

4.6. Vector Store Management and Index Persistence

1. Purpose and Scope This component is responsible for managing the lifecycle of the dense vector indices used for semantic search. FAISS (Facebook AI Similarity Search) was used for its performance. All related logic is encapsulated within the `VectorStoreHandler` class, which handles the deterministic generation of index paths, the creation or loading of FAISS stores, and the configuration of the retriever. The design is kept minimal and aligned with the actual implementation, focusing on a robust and reproducible persistence mechanism.

2. Index Persistence and Path Scheme To ensure reproducibility and efficiency across experimental runs, each combination of a person, chunking strategy, and its parameters maps to a unique on-disk location. This deterministic path scheme guarantees that identical configurations always resolve to the same index directory. For the semantic chunking strategy, which has no size or overlap parameters, '_0_0' was used as a fixed suffix for consistency.

```
./database/{splitter_type}_{chunk_size}_{chunk_overlap}
```

```python
def _get_vector_store_path(
    self,
    splitter_type: SplitterType,
    chunk_size: int,
    chunk_overlap: int
) -> str:
    """Constructs a unique path for the vector store based on config."""
    base_path = VECTOR_STORE_PATH
    if splitter_type == "semantic":
        chunk_size, chunk_overlap = 0, 0
    return os.path.join(base_path, f"{splitter_type}_{chunk_size}_{chunk_overlap}")
```

Listing 4.10: Deterministic Path Generation in `VectorStoreHandler`

3. Implementation: The `VectorStoreHandler` Class The handler's core logic is its ability to either load a pre-existing index or create a new one if it doesn't exist. This "create-or-load" pattern is essential for running evaluations efficiently. The main methods reflect this workflow, relying on the LangChain FAISS wrapper for the underlying operations.

```python
# (Inside the VectorStoreHandler class)

def _create_vector_store(self, split_documents: List[Document], ...) -> FAISS:
    """Creates and saves a new FAISS vector store."""
    db_path = self._get_vector_store_path(...)
    # Uses FAISS.from_documents to create the index from Document objects
    db = FAISS.from_documents(documents=split_documents, embedding=self.embeddings)
    db.save_local(folder_path=db_path) # Persists index.faiss and index.pkl
    return db

def _get_existing_vector_store(self, ...) -> FAISS:
    """Loads an existing FAISS vector store from a specified path."""
    db_path = self._get_vector_store_path(...)
    if not os.path.isdir(db_path):
        return None
    # Loads the persisted index files
    db = FAISS.load_local(
        folder_path=db_path,
        embeddings=self.embeddings,
        allow_dangerous_deserialization=True
    )
    return db

def get_db_and_retriever(self, ...):
```

```
    """Public method to get a retriever, loading or creating the DB
    as needed."""
    db = self._get_existing_vector_store(...)
    if db:
        retriever = self._create_retriever(db=db)
        return retriever, db
    else:
        # If not found, triggers the creation process (not shown for
    brevity)
        # and then returns the retriever and db.
        ...
```

Listing 4.11: Core Methods for FAISS Index Management

4. Retriever Configuration For pure dense retrieval, LangChain's standard retriever was used configured for similarity search with a fixed value for 'k' (the number of documents to retrieve). Other modes, such as Maximum Marginal Relevance (MMR) or similarity score thresholding, are not used in the experiments to maintain a consistent baseline. The 'HybridRetriever' component also uses a similarity-based search for its dense part, specifically calling the 'similarity_search_with_relevance_scores' method on the FAISS object.

```
def _create_retriever(self, db: FAISS) -> VectorStoreRetriever:
    """Creates a retriever from the vector store."""
    return db.as_retriever(
        search_type="similarity",
        search_kwargs={"k": self.search_kwargs_num}
    )
```

Listing 4.12: Standard similarity retriever configuration

5. Path Structure Example The naming convention creates a clear hierarchy:

```
./database/
  Washington_recursive_1000_30/
      index.faiss          # The FAISS index file
      index.pkl            # Document store and metadata
      metadata.json        # Index creation parameters
  Washington_semantic_0_0/
      index.faiss
      index.pkl
      metadata.json
  Washington_sentence_transformer_256_30/
      index.faiss
      index.pkl
      metadata.json
```

6ns. Summary and Best Practices The vector store management is designed for systematic and reproducible experiments:

- **Deterministic Paths:** One FAISS store is maintained per unique configuration (*strategy*, *size*, *overlap*).
- **Index Persistence:** The standard `index.faiss` was persisted and `index.pkl` files created by LangChain. A new index is built only if the configuration changes.
- **Consistent Retrieval:** The retriever is consistently set to `similarity` search with a fixed 'k' for fair comparisons across pipelines.
- **BM25 Alignment:** The metadata within the FAISS index (specifically `metadata["source"]`) is preserved and serves as the synchronization key for the hybrid retriever, as detailed in Section 4.8.

4.7. The Core QA Pipeline: Orchestration and Execution

1. Purpose and Design Philosophy The `InitializeQuesionAnsweringChain` class and its central `answer` method serve as the primary orchestrator for the entire RAG process. This component connects all individual modules—the retriever, the optional reranker, the language model, and the factual verifier—to transform a user query into a grounded, evaluable response. The design prioritizes modularity, configurability, and comprehensive metadata generation. A key aspect of the design is the ability to temporarily modify the retriever's state to create a richer candidate pool for subsequent precision-enhancing steps like reranking.

2. Class Initialization The class is initialized with the core components required for the pipeline. Notably, it accepts a pre-configured retriever object and a flag to conditionally enable the reranking stage.

```
class InitializeQuesionAnsweringChain:
def init(
self,
llm,
retriever,      # VectorStoreRetriever or HybridRetriever instance
db,             # FAISS db object
search_kwargs_num: int = 3,
use_reranker: bool = True,
# ... other parameters
):
self.llm = llm
self.retriever = retriever
# ... other initializations
self.use_reranker = use_reranker
self.verifier = FactualVerifier()
self.cross_encoder = None
```

```
17
18      if self.use_reranker:
19          self._init_reranker()
```

Listing 4.13: Abridged class initializer in `init_chain.py`

Implementation Note on Reranker Device. As detailed in Section 4.9, the cross-encoder is initialized within the `_init_reranker` method.

3. The `answer` Method: A Step-by-Step Walkthrough The `answer` method executes a multi-stage pipeline designed to maximize context relevance and enable thorough evaluation. It follows a precise sequence of operations:

4.7.1. The Answer Method: End-to-End Pipeline

Pipeline Overview The `answer` method implements a sophisticated multi-stage pipeline:

Person-aware query handling and output styling This pipeline is person-aware by design. First, the effective retrieval query is normalized with the given person context (object `Person`), resolving ambiguous pronouns (e.g., "you") via a lightweight prompt-based transformation (`QUERY_TRANSFORMATION` / `QUERY_TRANSFORMATION_GERMAN`) that inserts `person.name`. Second, corpus isolation is enforced at the storage layer: per-person vector stores are resolved to deterministic paths (`./database/{person}_{strategy}_` semantic uses `_0_0`), ensuring that retrieval only targets the intended biography/-source set. For answer generation the LLM directly calls with a strict grounded prompt (`STRICT_QA_PROMPT` or its German variant), and - optionally for UX - apply an impersonation style layer (`IMPERSONATION_PROMPT` or `IMPERSONATION_PROMPT_WITH_PERSONA` to the already computed factual answer. In evaluation mode I disable impersonation and conversation memory; metrics (e.g., RAGAS) are computed on the *final, post-rerank* contexts and the *factual (pre-impersonation)* answer to keep results comparable across persons and languages.

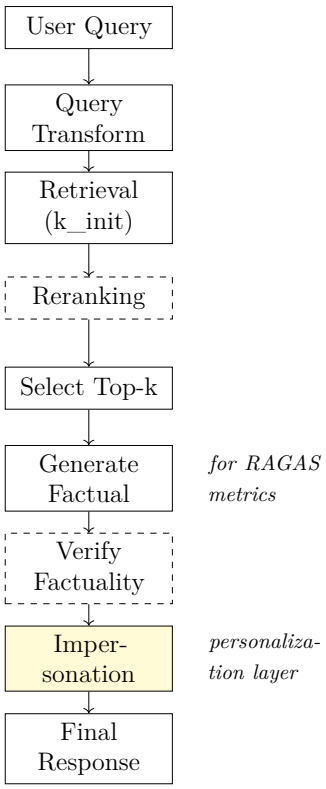

Figure 4.2.: Simplified linear view of the RAG pipeline with impersonation as final stage

1. **Stateful Retriever Configuration for Candidate Expansion:** The process begins by fetching a broad set of candidate documents. A key strategy here is to retrieve more documents initially (`k_init`) than are ultimately needed for the final context (`top_n`). To achieve this, the retriever's state (specifically its k, dense_k, and bm25_k attributes) is temporarily modified to request the larger k_init pool. This state change is safely managed within a try...finally block to ensure the retriever's original parameters are restored after the query, preventing side effects.
2. **Initial Candidate Retrieval:** With the temporarily expanded parameters, the configured retriever (dense or hybrid) is called to fetch the initial set of k_init documents.
3. **Conditional Reranking:** If the use_reranker flag is active, this initial pool of documents is passed to the Cross-Encoder Reranker (as described in Section 4.9). This step re-sorts the documents based on fine-grained relevance, producing the final list of top_n documents. If the flag is inactive, the pipeline simply selects the top documents from the initial retrieval.
4. **Context Formulation and Generation:** The page content from the final top_n documents is concatenated to form the context. Answer generation then uses a direct prompt invocation with the `STRICT_QA_PROMPT`. This prompt, formatted with the context and query, is passed directly to the

LLM's `.invoke()` method. No abstracting LangChain 'stuff' chain is used in this final step.

5. **Factual Verification:** The generated answer and the context it was based on are then passed to the `FactualVerifier` component. This step calculates a factuality score, providing a measure of how well the answer is grounded in the provided source text (detailed in Section 4.10).
6. **Response and Metadata Packaging:** The method concludes by packaging the final answer, the list of documents used to generate it, and a comprehensive metadata dictionary.

4. The Returned Data Structure The `answer` method is designed to produce a rich output that facilitates detailed analysis. It returns a tuple containing the answer string, the list of final (reranked) documents with their scores, and the metadata dictionary.

```

The method returns a tuple with three elements
return (answer_text, reranked_docs_with_scores, metadata)

Example of the returned metadata dictionary
metadata = {
"factuality_score": 0.95,
"k_init": 10,
"top_n": 3,
"used_reranker": True,
"num_initial_docs": 10,
"num_final_docs": 3
}
```

Listing 4.14: Final tuple data structure returned by the answer method

This structured output is crucial, as it allows the evaluation framework (Chapter 5) to precisely assess each component's contribution, using the exact context that the LLM was exposed to.

4.8. Hybrid Retrieval

1. Purpose and Motivation Hybrid retrieval combines the strengths of two complementary search strategies to maximize the relevance of the retrieved documents:

- **Lexical Matching** (BM25) excels at finding precise keywords or rare technical terms.
- **Semantic Similarity** (Dense Vectors) enables the discovery of conceptual matches and paraphrases, even when the exact query terms are not present in the text.

By fusing the results from both retrievers, a more robust performance is achieved across a wide variety of queries, from specific factual questions to open-ended, conceptual explorations.

2. BM25 Index Preparation A critical prerequisite for fusing the results is ensuring that document IDs are consistent between the lexical and dense indices. This was ensured through a pre-processing and indexing step. Using the `make_bm25_jsonl.py` script, the document chunks are converted into the JSONL format required by Pyserini. The source URL of the document, which is also stored in the metadata of the FAISS index, is used as the unique `id`.

```
chunks = dp.get_chunked_documents(...)

with open(output_path, "w", encoding="utf-8") as f:
for chunk in chunks:
record = {
# Must match the FAISS metadata['source']
"id": chunk.metadata["source"],
"contents": chunk.page_content
}
f.write(json.dumps(record, ensure_ascii=False) + "\n")
```

Listing 4.15: Creating the JSONL file for Pyserini in `make_bm25_jsonl.py`

Important. Each BM25 entry requires a **unique** ID per chunk (e.g., chunk_id = f"sourceoff-start"), which must also be present in the FAISS metadata for a precise match. As a practical alternative (used in this implementation for simplicity), a document-level ID can be employed, where the same BM25 score is broadcast to all chunks originating from the same source. This approach carries the risk of Pyserini overwriting entries if multiple chunks share an identical ID, but can be effective if chunk-level score differentiation is not critical.

The `prebuild_indices.py` script then automates the building of the Lucene indices for all chunking configurations using Pyserini.

3. BM25 Retriever Implementation The `BM25Retriever` serves as a simple wrapper around Pyserini's `LuceneSearcher`. The `get_scores` method returns the BM25 scores for the top-k documents as a dictionary that maps the document ID to its score.

```
from pyserini.search.lucene import LuceneSearcher

class BM25Retriever:
def init(self, index_dir: str):
self.searcher = LuceneSearcher(index_dir)

def get_scores(self, query: str, k: int) -> dict[str, float]:
    hits = self.searcher.search(query, k)
```

```
    out = {}
    for h in hits:
        doc = self.searcher.doc(h.docid)
        key = doc.get('id') if doc is not None else str(h.docid)
        out[key] = float(h.score)
    return out
```

Listing 4.16: BM25Retriever implementation in `bm25_retriever.py`

4. Hybrid Retriever Implementation The `HybridRetriever` class in `vector_store_handler.p` is the core component of the fusion logic. It has been significantly enhanced to provide more precise control over the fusion process and to normalize the scores.

```

class HybridRetriever:
def init(self, db: FAISS, bm25_retriever, alpha: float = 0.5, k: int
    = 3,
dense_k: int | None = None, bm25_k: int | None = None):
self.db = db
self.bm25_retriever = bm25_retriever
self.alpha = alpha
self.k = k
// NEW: Separate candidate pools for each retriever
self.dense_k = int(dense_k) if dense_k is not None else max(2 * self.
    k, self.k)
self.bm25_k = int(bm25_k) if bm25_k is not None else max(3 * self.k,
    self.k)
@staticmethod
def _squash_bm25(bscore: float) -> float:
    // NEW: Squash function to normalize the BM25 scores
    return 1.0 - 1.0 / (1.0 + max(bscore, 0.0))

def get_relevant_documents(self, query: str):
    b_scores = self.bm25_retriever.get_scores(query, self.bm25_k)
    dense_docs_with_scores = self.db.
    similarity_search_with_relevance_scores(query, k=self.dense_k)

    fused: list[tuple[Document, float]] = []
    for doc, dscore in dense_docs_with_scores:
        // Matching via the synchronized document ID
        doc_id = doc.metadata.get('source', None)
        bscore = b_scores.get(doc_id, 0.0)

        bscore_norm = self._squash_bm25(bscore)
        // Linear interpolation of the normalized scores
        fused_score = self.alpha * bscore_norm + (1.0 - self.alpha) *
     float(dscore)
        fused.append((doc, fused_score))

    fused.sort(key=lambda x: x[1], reverse=True)
```

```
33     return fused[: self.k]
```

Listing 4.17: Revised HybridRetriever Implementation

Implementation Details The fusion logic is ”dense-anchored”: The candidate pool were established using only the top-`k` dense results. The scores of these candidates are then re-weighted using their corresponding (and possibly zero) BM25 scores. Purely lexical hits that do not appear in the initial dense retrieval are not considered at this stage. This represents a design trade-off to reduce noise from potentially irrelevant keyword matches.

Defaults. The retriever is initialized with sensible defaults: alpha=0.5, k=3, dense_k=max(2k, k), and bm25_k=max(3k, k).

Diagnostics. A BM25 match rate was logged, which measures the percentage of dense candidates that have a corresponding BM25 ID. This helps to detect potential ID mismatches or drifts between the two indices early on.

5. Integration into the Evaluation Pipeline Instead of a dynamic selection per API call, the retriever type is now defined as part of the configuration for each evaluation run. This allows for a systematic comparison of the different strategies. The `1_evaluate_rag_base.ipynb` notebook instantiates either a pure dense retriever or the `HybridRetriever` and passes it to the QA chain.

```
#inside the run_single_test function
...

retrieval_mode = config["retrieval_mode"]
#db (FAISS object) is loaded or created...
if retrieval_mode == 'hybrid':
print("[INFO] Using HYBRID Retriever for this run.")
bm25_index_dir = f"./bm25_indexes/Washington_{config['splitter_type
    ']}"
bm25_retriever = BM25Retriever(bm25_index_dir)
retriever = HybridRetriever(db=db, bm25_retriever=bm25_retriever, k=
    search_kwargs_num)
else: # 'dense'
print("[INFO] Using DENSE Retriever for this run.")
retriever = db.as_retriever(search_kwargs={"k": search_kwargs_num})

#The selected retriever is passed to the QA chain...
qa_chain = InitializeQuesionAnsweringChain(
llm=llm,
retriever=retriever,
db=db,
# ...
)
```

Listing 4.18: Retriever instantiation in the evaluation pipeline

6. Design Decisions

- **Score Fusion over Rank Fusion:** I opted for a linear interpolation of scores instead of a pure rank-based fusion (like RRF). This allows for a more nuanced weighting, as not only the order but also the absolute score magnitude (especially after normalization) is taken into account.
- **BM25 Score Normalization:** Since BM25 scores are unbounded while similarity scores are often in the [0, 1] range, a "squashing" function is applied. This function monotonically maps the BM25 scores to the [0, 1] range, making them more comparable to the dense scores without altering their relative ranking. Dense scores were kept as returned by FAISS; in the setup they are typically bounded and comparable. If versions differ, value ranges are logged and, optionally, a per-batch min-max normalization is applied to dense scores before interpolation.
- **Asymmetric Candidate Pools:** The use of separate parameters (`dense_k` and `bm25_k`) allows to draw a larger number of candidates from the lexical index (which tends to be faster), providing a broader base for the subsequent fusion.
- **Synchronization via Metadata:** The coupling of the two systems is handled exclusively through the `source` ID in the metadata. This is a robust and flexible mechanism that requires no changes to the underlying index structures.

7. Summary The revised hybrid retrieval system combines the lexical precision of BM25 with the semantic breadth of dense embeddings. Through explicit index preparation, score normalization, and a flexible fusion logic, a robust and powerful foundation for retrieving relevant contexts is established.

4.9. Reranking Retrieved Documents

1. Purpose After retrieving a set of candidate documents (via dense or hybrid methods), a cross-encoder was applied to rerank them. This is done by jointly scoring each (query, passage) pair. This stage significantly improves the precision of the top-N list by leveraging the cross-attention capabilities of the model to capture nuanced relevance signals that the initial, recall-oriented retrievers may miss.

2. Implementation as a Configurable Component To systematically analyze its impact, the reranker is implemented as a configurable component, controlled by a `use_reranker` flag passed during the QA pipeline's initialization. This design allows for direct, comparative ablation studies.

```
1
2 #In InitializeQuesionAnsweringChain.init
3 ...
```

```
def init(self, ..., use_reranker: bool = True):
self.use_reranker = use_reranker
self.cross_encoder = None
if self.use_reranker:
    self._init_reranker() # Calls the private method to load the
    model

def _init_reranker(self):
"""Initializes the CrossEncoder model for re-ranking."""
model_name = "cross-encoder/ms-marco-MiniLM-L-6-v2"
# Device selection logic for GPU/CPU is handled here...
device = 'cpu' # Simplified for example
model_kwargs = {'device': device}
print(f"[INFO] Initializing CrossEncoder '{model_name}' on device: {
    device}")
self.cross_encoder = HuggingFaceCrossEncoder(
model_name=model_name, model_kwargs=model_kwargs
)
```

Listing 4.19: Conditional reranker initialization in `init_chain.py`

3. Reranking Logic The core logic resides in the private `_rerank_documents` method. It takes the initial list of retrieved documents with their scores, computes new relevance scores using the cross-encoder, and returns a sorted list of the top-N documents, now including their new, more precise scores.

```
def \_rerank\_documents(
self, query: str, docs_with_scores: List[Tuple[Document, float]],
    top_n: int
) -> List[Tuple[Document, float]]: # Note: Returns (doc, score)
    tuples
"""Re-ranks the initially retrieved documents using the CrossEncoder.
    """
if not self.cross_encoder or not docs_with_scores:
# Fallback: sort by original scores and return top_n
sorted_docs = sorted(docs_with_scores, key=lambda x: x[1], reverse=
    True)
return sorted_docs[:top_n]

# 1) Extract passage texts and build (query, passage) pairs
doc_contents = [doc.page_content for doc, score in docs_with_scores]
query_doc_pairs = [[query, doc_content] for doc_content in
    doc_contents]

# 2) Compute new relevance scores with the cross-encoder
scores = self.cross_encoder.score(query_doc_pairs)

# 3) Zip original docs with new scores and sort descending
original_docs = [doc for doc, score in docs_with_scores]
docs_with_new_scores = list(zip(original_docs, scores))
```

```
reranked_docs = sorted(docs_with_new_scores, key=lambda x: x[1],
    reverse=True)

# 4) Return the top_n documents with their new scores
return reranked_docs[:top_n]
```

Listing 4.20: The document reranking method

4. Integration and Ablation Study Design The reranker is conditionally invoked within the main `answer` method. This logic is central to the experimental setup, as it allows to run parallel evaluations with and without this component.

```

(Simplified) logic from the answer() method in init_chain.py

def answer(self, query: str):
# 1) Retrieve an initial, larger set of candidates (e.g., k_init=10)
docs_with_scores = self.retriever.get_relevant_documents(query)
# 2) Conditionally apply reranking to get the final top_n (e.g.,
    top_n=3)
if self.use_reranker and self.cross_encoder and docs_with_scores:
    reranked_docs_with_scores = self._rerank_documents(query,
    docs_with_scores, top_n=self.top_n)
    used_reranker_flag = True
else:
    # Fallback: simply take the top_n from the initial retrieval
    sorted_docs = sorted(docs_with_scores, key=lambda x: x[1],
    reverse=True)
    reranked_docs_with_scores = sorted_docs[:self.top_n]
    used_reranker_flag = False

# 3) Generate context from the final (potentially reranked) documents
context = "\n\n".join(doc.page_content for doc, _ in
    reranked_docs_with_scores)
# ... then generate answer with the LLM
```

Listing 4.21: Conditional reranking within the QA pipeline

This configurability is leveraged in the evaluation matrix to systematically measure the reranker's added value.

```
evaluation_matrix = [
# Dense without reranker
{"pipeline_name": "dense_semantic", ..., "use_reranker": False},
# Dense WITH reranker
{"pipeline_name": "dense_semantic_rerank", ..., "use_reranker": True
    },
# Hybrid without reranker

{"pipeline_name": "hybrid_semantic", ..., "use_reranker": False},
# Hybrid WITH reranker
{"pipeline_name": "hybrid_semantic_rerank", ..., "use_reranker": True
    },
```

```
]
```

Listing 4.22: Reranker as a variable in the evaluation matrix

The systematic inclusion of reranking variants enables to answer key research questions:

- Does reranking provide consistent improvements across all chunking strategies?
- Is the reranker more beneficial for dense or hybrid retrieval?
- What is the cost-benefit trade-off in terms of latency vs. quality?

The evaluation compares 6 configurations without reranking against 6 configurations with reranking, providing comprehensive insights into its value.

5. Design Decisions Practical Considerations

- **Separate from Factual Verification:** Reranking (ordering retrieval candidates before generation) is clearly distinguished from the post-generation fact-checking step (FactualVerifier), keeping concerns isolated.
- **Cross-Encoder Choice:** I use a dedicated cross-encoder model (`cross-encoder/ms-marco-MiniLM-L-6-v2`) to leverage full attention between query and passage.
- **Graceful Fallback:** If no cross-encoder is configured or if an error occurs, the pipeline gracefully returns the original top-N documents, ensuring robustness.
- **Performance vs. Precision:** Cross-encoders yield higher accuracy but at increased latency. Limiting the initial candidate pool (e.g., reranking the top 10 documents) balances quality with responsiveness.

6. Summary Document reranking with a cross-encoder elevates the most relevant contexts by:

- Jointly modeling query and passage semantics for fine-grained relevance scoring.
- Functioning as a configurable component, enabling systematic ablation studies.
- Remaining orthogonal to the later factual verification stage.

This stage is critical for ensuring that the contexts fed into the answer generation chain are as precise and relevant as possible.

4.10. Factual Verification

1. Purpose Factual verification is a post-generation check that assesses how well a language model's answer is supported by the retrieved context. By re-encoding

(context, answer) pairs with a cross-encoder, I obtain a scalar "factuality" score in $[0, 1]$. This score can be used to filter or flag low-confidence answers, request additional information, or trigger fallback strategies—thereby reducing hallucinations and improving trustworthiness.

2. Implementation This logic lives partly in the `FactualVerifier` class ('factual_verifier.py') and is configured via the chain's constructor in 'init_chain.py'.

```
from langchain_community.cross_encoders import
    HuggingFaceCrossEncoder

class FactualVerifier:
    def __init__(self, model_name: str = "cross-encoder/stsb-roberta-
    base"):
        # Change 1: Use a Semantic Textual Similarity (STS) model,
        # which is better suited for comparing an answer to a context
    .
        self.reranker = HuggingFaceCrossEncoder(model_name=model_name
    )

    def score(self, context: str, answer: str) -> float:
        """Returns a factuality score in [0..1] for a (context,
    answer) pair."""
        scores = self.reranker.score([[context, answer]])

        # Change 2: Convert the model's output to a list before
    accessing
        # the first element to ensure compatibility.
        return float(list(scores)[0])
```

Listing 4.23: Updated FactualVerifier implementation in factual$_verifier.py$

The `factuality_threshold` parameter determines the minimum acceptable score.

```
from langchain_community.cross_encoders import
    HuggingFaceCrossEncoder

class FactualVerifier:
    def __init__(self, model_name: str = "cross-encoder/ms-marco-
    MiniLM-L-6-v2"):
        # Initialize the cross-encoder for (context, answer) scoring
        self.reranker = HuggingFaceCrossEncoder(model_name=model_name
    )

    def score(self, context: str, answer: str) -> float:
        """Returns a factuality score in [0..1] for a (context,
    answer) pair."""
        scores = self.reranker.score([[context, answer]])
        return float(scores[0])
```

Listing 4.24: Implementation of answer factual verification

3. Integration into the QA Pipeline Immediately after generating the answer with the LLM, the verifier gets invoked and compared against the configured threshold:

```
# ... all previous steps (retrieval, reranking, generation) are complete.
# 'transformed_answer' holds the final, post-processed answer string.
# 'context_for_llm' holds the context that was used for generation.

# 1) Compute factuality score against the provided context
factuality_score = self.verifier.score(
    context=context_for_llm,
    answer=transformed_answer
)

# 2) Optionally check against a pre-configured threshold
if factuality_score < self.factuality_threshold:
    print(f"[WARN] Low factuality score detected: {factuality_score:.2f}")

# 3) Package all results, including the score, into a dictionary
dict_query = {
    "original_query": original_query,
    "transformed_answer": transformed_answer,
    "factuality_score": factuality_score,
}

# 4) Return the final answer tuple and source documents
return (transformed_answer, dict_query), relevant_docs_with_scores_initial
```

Listing 4.25: Integration of factual verification at the end of the QA chain

4. Design Decisions

- **Specialized Models for Specialized Tasks:** Two different types of cross-encoder models were used. A model trained on MS-MARCO is used for document *reranking*, as it excels at query-passage relevance. For *factual verification*, a model trained on Semantic Textual Similarity (STS) was used, as it is better suited to scoring the semantic equivalence between a generated answer and its source context.
- **Configurable Threshold**: By passing `factuality_threshold` into the chain constructor, users can tune sensitivity per application.
- **Scalar Output**: a single $[0, 1]$ score was returned without enforcing a hard cutoff; action on low scores is left to the client code.
- **Fallback Strategy**:A robust implementation should include a fallback strategy in future where exceptions are caught and a neutral score (e.g., 0.5) is returned to avoid blocking the pipeline.

5. Practical Considerations

- *Latency*: Single-pair scoring is fast, but you may batch multiple (context, answer) pairs when evaluating alternative answers.
- *Threshold Calibration*: Empirically determine the best `factuality_threshold` on a validation set to balance false positives against false negatives.
- *Context Window*: Ensure the context passed to the verifier includes all evidence and is not truncated.
- *Multilingual Support*: For multilingual QA, select or fine-tune cross-encoder models appropriate for each language.

6. Summary Factual verification adds a safety net after answer generation by:

- Jointly encoding the model's answer and its context for a fine-grained support score,
- Allowing configurable thresholds to trigger regeneration or user alerts,
- Increasing overall trust and reducing hallucinations in downstream applications.

4.11. Server: Pipeline Integration and API Endpoint

1. Purpose The Flask server (`server.py`) exposes the RAG pipeline via `/ask-question`. The endpoint parses request parameters, resolves person-specific indices, composes the retriever (dense or hybrid), and executes the QA chain. The server remains thin; most logic (query normalization, rerank, prompting) lives in `InitializeQuesionAnsweringChain`.

2. Endpoint Parameters (current) Key GET params used per request:

- `question` (string), `person` (e.g., `Washington`), `language` (`en`/`de`)
- `retrieval_mode` (`dense`|`hybrid`), `splitter_type` (`recursive`|`sentence_transformer`|`se`
- `chunk_size`, `chunk_overlap`, `search_kwargs_num` (= top-k), `use_reranker` (bool)
- UI/UX toggles (e.g., `impersonate`, `voice`); `temperature` for the LLM

3. Person-aware index resolution and lazy create/load Each configuration maps to a deterministic path; semantic uses `_0_0`. If the FAISS store does not exist, it is built on demand.

```
person_name = request.args.get("person", "Washington")
splitter_type = request.args.get("splitter_type", "semantic")
chunk_size = int(request.args.get("chunk_size", 500))
chunk_overlap = int(request.args.get("chunk_overlap", 100))

db_path = vectorstore_handler._get_vector_store_path(
```

```
    person=person_name, splitter_type=splitter_type,
    chunk_size=chunk_size, chunk_overlap=chunk_overlap
)

try:
    retriever, db = vectorstore_handler.get_db_and_retriever(
        vector_store_name=person_name,  # used for naming
        splitter_type=splitter_type,
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap
    )
except VectorStoreNotFoundError:
    split_docs = doc_processor.get_chunked_documents(
        directory_path=f"./data/{person_name}",
        splitter_type=splitter_type,
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap
    )
    retriever, db = vectorstore_handler.create_db_and_retriever(
        chunked_documents=split_docs,
        vector_store_name=person_name,
        splitter_type=splitter_type,
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap
    )
```

Listing 4.26: Per-person FAISS path + JIT create/load

4. Retriever composition (dense vs. hybrid) Hybrid uses BM25 (Pyserini) + FAISS; the BM25 index is also resolved per person (and typically per splitter).

```
retrieval_mode = request.args.get("retrieval_mode", "dense")
use_reranker = request.args.get("use_reranker", "true").lower() in ("
    1","true","yes")

if retrieval_mode == "hybrid":
    bm25_index_dir = f"./bm25_indexes/{person_name}_{splitter_type}"
    bm25_retriever = BM25Retriever(bm25_index_dir)
    retriever = HybridRetriever(db=db, bm25_retriever=bm25_retriever,
                                k=search_kwargs_num)
else:
    # 'retriever' already returned by vectorstore_handler (FAISS
    similarity)
    pass
```

Listing 4.27: Dense/Hybrid selection

5. QA execution and response packaging The chain returns a tuple `(answer, reranked_docs_with_scores, metadata)`. Query transformation (person-aware) and optional impersonation live *inside* the chain; in evaluation mode impersonation is disabled.

```
qa = InitializeQuesionAnsweringChain(
    llm=llm,
    retriever=retriever,
    db=db,
    person=person,                        # object with name/lang
    search_kwargs_num=search_kwargs_num,
    use_reranker=use_reranker,
    language=language
)

answer, docs_with_scores, meta = qa.answer(question)

def serialize_docs(docs_with_scores, limit=search_kwargs_num):
    out = []
    for doc, score in docs_with_scores[:limit]:
        out.append({
            "score": float(score),
            "source": doc.metadata.get("source"),
            "snippet": doc.page_content[:300]
        })
    return out

return jsonify({
    "answer": answer,
    "factuality_score": float(meta.get("factuality_score", 0.0)),
    "used_reranker": bool(meta.get("used_reranker", False)),
    "retrieval_mode": retrieval_mode,
    "splitter_type": splitter_type,
    "k_init": int(meta.get("k_init", 0)),
    "top_n": int(meta.get("top_n", search_kwargs_num)),
    "relevant_docs": serialize_docs(docs_with_scores)
})
```

Listing 4.28: Chain call and JSON response

6. Minimal diagnostics (optional but helpful) A few consistent log lines make server vs. notebook behavior comparable:

```
print(f"[EVALCFG] person={person_name} index={db_path} "
      f"mode={retrieval_mode} splitter={splitter_type} "
      f"k={search_kwargs_num}")
print(f"[RERANKCFG] use_reranker={use_reranker} model=ms-marco-MiniLM
    -L-6-v2 max_len=384")
```

Listing 4.29: Consistent diagnostics

7. Summary The server keeps orchestration lightweight: it resolves per-person stores, composes dense or hybrid retrievers, and delegates query normalization, reranking, and prompting to the QA chain. This separation preserves reproducibility (deterministic paths) and lets the same endpoint serve interactive UI and evaluation runs with only parameter changes.

4.12. Eyewitness Mode

Purpose The *Eyewitness Mode* is a stylistic toggle that instructs the model to add a brief, present-day eyewitness reflection to the persona's answer. The mode increases narrative engagement while leaving retrieval, reranking, and factual answer generation unchanged.

UI / API Flow The front end exposes a checkbox. When enabled, it sends `eyewitness_mode=true` with the `/ask-question` request. The server parses the parameter and forwards the boolean to the QA chain constructor.

```
const response = await ask_question(
  selected, query, selectedVoice,
  chunkSize.toString(), chunkOverlap.toString(),
  temperature.toString(), searchKwargsNum.toString(),
  useOpenai ? "True" : "False",
  selectedLanguage, previousQuestion, previousAnswer,
  eyewitnessMode ? "True" : "False"  // <-- toggle
);
```

Listing 4.30: Frontend: pass on to API

```
eyewitness_mode_str = request.args.get("eyewitness_mode", default="
    False", type=str)
eyewitness_mode = eyewitness_mode_str.lower() in ("true", "1", "yes")

qa_chain = InitializeQuesionAnsweringChain(
    llm=llm, retriever=retriever, db=db, person=person,
    search_kwargs_num=params["search_kwargs_num"],
    language=params["language"],
    use_reranker=params["use_reranker"],
    eyewitness_mode=eyewitness_mode
)
```

Listing 4.31: Server: parse flag and hand on to pipeline

Implementation (style layer only) The chain first produces a strictly grounded factual answer. In the subsequent *impersonation* step, a short directive is appended to the impersonation prompt *iff* `eyewitness_mode=True`. This directive tells the model to add 1–2 present-day eyewitness sentences and not to introduce new facts. No retrieval parameters or contexts are changed.

```
# ... factual_answer has been created ...

if self.person.personality is not None:
    prompt_text = IMPERSONATION_PROMPT_WITH_PERSONALITY.format(
        person=self.person.name,
        answer=factual_answer,
        personality=self.person.format_personality_prompt(),
    )
```

```
else:
    prompt_text = IMPERSONATION_PROMPT.format(
        person=self.person.name, answer=factual_answer
    )

# Eyewitness-addon only when flag activated (EN/GER)
if self.eyewitness_mode:
    suffix = (
        "\n\n[Eyewitness] Ergänze am Ende -12 sehr kurze Sätze aus "
        "heutiger Zeitzeug*innen-Perspektive. Keine neuen Fakten."
        if self.language == "de" else
        "\n\n[Eyewitness] Append -12 very short sentences from a present-day "
        "eyewitness perspective. Do not add new facts."
    )
    prompt_text += suffix

final_answer = self.llm.invoke(prompt_text)
```

Listing 4.32: init_chain.py: Eyewitness as Prompt-Suffix (simplified)

Design Decisions

- **Isolation:** The eyewitness effect is injected only in the final style layer (impersonation). Retrieval, hybrid fusion, reranking and the factual verifier remain untouched.
- **No new facts:** The directive explicitly prohibits introducing new information; it only adds a brief reflective addendum.
- **Language-aware:** The suffix is localized (German/English) based on `language`.
- **Evaluation unaffected:** In evaluation mode the factual answer (pre-impersonation) and the final contexts are used for metrics; the style addendum does not change ground truth.

Summary Eyewitness Mode is a lightweight, opt-in presentation feature. It keeps the RAG core stable and adds a short, clearly marked present-day reflection to the impersonated output when requested by the client.

4.13. PDF Export Functionality

1. Purpose An “Export PDF” feature was added to allow users to download the entire chat history as a formatted PDF document. This complements the existing Markdown export and provides a more polished output for sharing or archival purposes.

2. Frontend Integration In `ChatSection.tsx`, following was introduced:

- A handler `exportChat('pdf')` that invokes the PDF export utility.
- A dedicated "Export PDF" button in the UI.

```
export const ChatSection: FC<ChatSectionProps> = ({ /* ... */ }) => {
  // New helper to dispatch export action
  const exportChat = (format: "markdown" | "pdf") => {
    if (format === "markdown") exportChatAsMarkdown(chat);
    else exportChatAsPDF(chat);
  };

  return (
    // ...
    <HStack spacing={2} mb={2}>
      <Button
        leftIcon={<FaFileExport />}
        size="sm"
        onClick={() => exportChat("markdown")}
      >
        Export MD
      </Button>
      {/* New PDF export button */}
      <Button
        leftIcon={<FaFileExport />}
        size="sm"
        onClick={() => exportChat("pdf")}
      >
        Export PDF
      </Button>
    </HStack>
    // ...
  );
};
```

Listing 4.33: Key additions in ChatSection.tsx

3. Utility Implementation In `utils/export.ts`, the PDF generator was implemented using `jsPDF`. Additionally, a missing import of the `Message` type was imported to ensure type safety.

```

+import { Message } from "@/app/utils/types";

import jsPDF from "jspdf";
import { saveAs } from "file-saver";

export function exportChatAsPDF(chat: Message[]) {
  const doc = new jsPDF({ unit: "pt", format: "letter" });
  let y = 40;
  doc.setFontSize(12);
  chat.forEach((m) => {
    const prefix = m.isHuman ? "You: " : "Bot: ";
```

```
13      const lines = doc.splitTextToSize(prefix + m.text, 500);
14      doc.text(lines, 40, y);
15      y += lines.length * 14 + 10;
16      if (y > 750) {
17        doc.addPage();
18        y = 40;
19      }
20    });
21    doc.save("chat.pdf");
22  }
```

Listing 4.34: Updated exports with PDF generation

4. Design Decisions

- **jsPDF for Simplicity:** `jsPDF` was chosen to keep dependencies minimal and output standard letter-format PDFs.
- **Automatic Pagination:** By tracking the vertical position `y` and adding pages when needed, it was ensured that no content got truncated.
- **Type Safety:** Importing the `Message` type guarantees that the export functions operate on the expected data shape.
- **User Experience:** Placing the "Export PDF" button alongside "Export MD" makes the feature discoverable without altering the existing form layout.

5. Summary The PDF export feature provides users with a ready-to-share document of their chat session. It leverages `jsPDF` for rendering, respects pagination, and integrates seamlessly into the existing React component, enhancing the utility of the front end with minimal footprint.

4.14. UI Extensions (Development)

To improve controllability and reproducibility, the web UI now exposes several toggles and selectors. These map 1:1 to API parameters and alter the RAG pipeline in a non-invasive way. A detailed mapping between UI controls and API can be seen in Table 4.4.

New Controls

- **Retrieval Mode (Toggle)**: `retrieval_mode` ∈ {`dense`, `hybrid`}; *hybrid* combines FAISS (dense) and BM25 (Lucene/Pyserini).
- **Splitter Type (Dropdown)**: `splitter_type` ∈ {`recursive`, `sentence_transformer`, `se`
- **Reranking (Toggle)**: `use_reranker` ∈ {`True`, `False`}; enables a CrossEncoder (MS MARCO MiniLM).
- **Eyewitness Mode (Toggle)**: `eyewitness_mode` ∈ {`True`, `False`}; appends a short present-day eyewitness addendum (no new facts).

Parameter Mapping

Table 4.4.: Mapping between UI controls and API parameters

UI Control	Type	API Parameter	Values (Default)
Hybrid Search	Toggle	`retrieval_mode`	dense/hybrid (dense)
Chunking Strategy	Dropdown	`splitter_type`	recursive/sentence/semantic (semantic)
Reranking	Toggle	`use_reranker`	true/false (false)
Eyewitness Mode	Toggle	`eyewitness_mode`	true/false (false)
Chunk Size	Slider	`chunk_size`	500-2000 (1000)
Num Documents	Slider	`num_docs`	1-10 (3)
Temperature	Slider	`temperature`	0.0-1.0 (0.1)

These UI extensions directly map to the pipeline components described in Sections 4.8 (hybrid mode), 4.5 (splitter types), and 4.9 (reranking toggle).

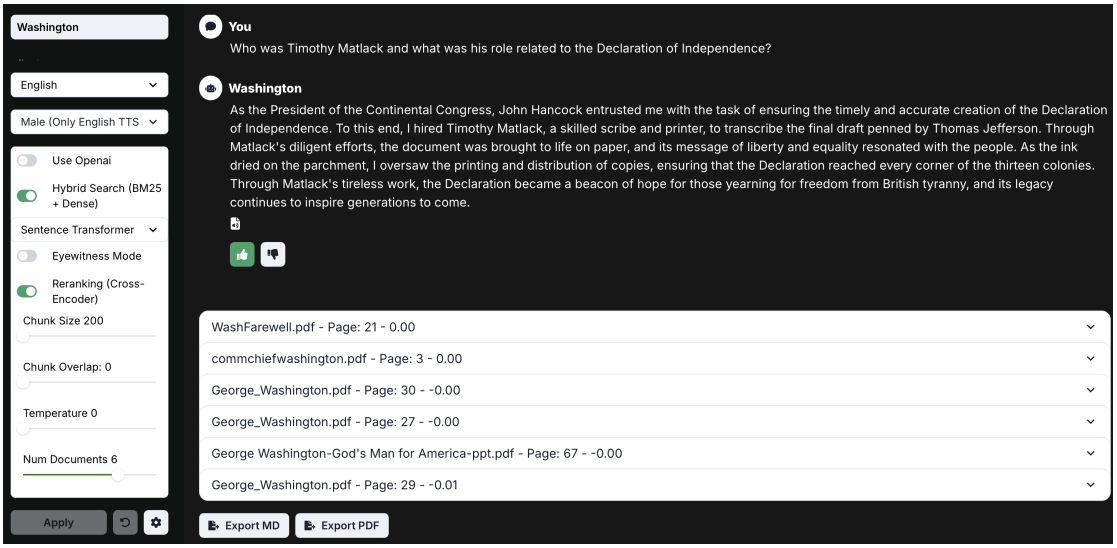

Figure 4.3.: Extended UI controls for pipeline configuration: (1) Retrieval mode toggle, (2) Chunking strategy selector, (3) Reranking toggle, (4) Export functions

Export Functionality Two export formats are supported:

- **Markdown**: Preserves formatting and citations for documentation
- **PDF**: Generates a formatted conversation transcript via PyPDF

Both exports include metadata (pipeline configuration, timestamps, factuality scores) for reproducibility and audit trails.

Real-time Feedback The UI displays:

- Factuality score (0.0-1.0) as a confidence indicator
- Retrieved document snippets with relevance scores
- Pipeline configuration used for each response

**Note:* The backend validates `splitter_type`; for unknown values it falls back to `semantic`.

Slider Guardrails To prevent misconfiguration and keep indices reproducible, the UI applies small guardrails to the sliders. For the `sentence_transformer` splitter, the user input for `chunk_size` is silently scaled down (input $\div 3$ with a minimum of 200) and a non-blocking toast informs the user about the effective value. `chunk_overlap` is automatically kept $<$ `chunk_size` (and reduced if necessary). For the `semantic` splitter, both sliders are disabled and labelled "auto"; on the server, `chunk_size` and `chunk_overlap` are forced to `0` (automatic mode). The backend mirrors these rules to ensure stable vector-store keys/paths and full reproducibility regardless of client input.

UX Notes All controls are disabled while a request is in flight. Changes are applied via "Apply" and sent with the next `/ask-question` call. The "Relevant Documents" section is collapsible and rendered *below* the chat log to avoid overlapping the assistant's answers.

4.15. Summary

This chapter presented the comprehensive technical implementation of the extended ALiVE system, detailing both the architectural foundations inherited from the predecessor project and the substantial enhancements developed in this thesis.

The development builds upon the original ALiVE architecture—a self-hosted RAG system using Mistral 7B and LangChain—while introducing several critical improvements that enable systematic evaluation and enhanced retrieval performance. The key technical contributions implemented include:

Advanced Document Processing: Three complementary chunking strategies (recursive, sentence-based, and semantic) were implemented to investigate the impact of document segmentation on retrieval quality. Each strategy addresses different aspects of content preservation, from maintaining syntactic boundaries to preserving topical coherence.

Hybrid Retrieval System: A sophisticated fusion mechanism combining BM25 lexical matching with dense vector similarity was developed, featuring configurable weighting parameters (α), separate candidate pools (k_{dense}, k_{bm25}), and score normalization to handle the inherent scale differences between retrieval methods.

Precision Enhancement through Reranking: A cross-encoder reranking stage using MS-MARCO MiniLM was integrated as an optional component, enabling fine-grained relevance scoring of query-passage pairs to improve the precision of the final context selection.

Robust Pipeline Architecture: The implementation features deterministic index persistence (per-person, per-strategy paths), graceful fallback mechanisms, and comprehensive metadata tracking throughout the pipeline. The modular design allows for systematic ablation studies by toggling individual components (reranking, hybrid retrieval) via configuration flags.

Evaluation Infrastructure: Beyond the core RAG pipeline, the system includes a complete evaluation framework that orchestrates 12 distinct pipeline configurations, computes RAGAS metrics, and enables statistical comparison of architectural choices—all while maintaining reproducibility through deterministic path resolution and consistent random seeds.

The technical stack leverages modern open-source technologies (LangChain 0.3.24, FAISS 1.10.0, Pyserini) while maintaining local deployability. Critical implementation decisions, such as the ”dense-anchored” hybrid fusion and the use of separate cross-encoder models for reranking versus factual verification, reflect careful consideration of the trade-offs between precision, recall, and computational efficiency.

The resulting system successfully extends the original ALiVE prototype into a comprehensive research platform that enables systematic investigation of RAG architectural choices while maintaining the core educational use case of conversing with historical figures. All components are designed for reproducibility, with extensive logging and diagnostic output to facilitate both debugging and performance analysis.

5. Evaluation

This chapter presents a comprehensive technical evaluation of the developed Retrieval-Augmented Generation (RAG) system, aiming to uncover optimal pipeline configurations that balance factual grounding, relevancy, and efficiency. The primary objective is to systematically assess the performance of various architectural variants using automatically generated questions and the quantitative RAGAS metrics.

The evaluation focuses exclusively on automated testing, without user studies or qualitative expert feedback, in order to isolate the impact of architectural and component-level decisions on retrieval and answer quality.

The experiments examine both the individual contributions of key components-such as chunking strategy, cross-encoder reranking, and hybrid retrieval—and their combined effects on overall system performance. Results are reported for multiple RAGAS metrics, allowing for a detailed analysis of trade-offs between *Faithfulness*, *Answer Relevancy*, and retrieval quality measures. In addition to mean scores, the variation in performance by question type is explored, and the correlation between different metrics is analyzed, offering a nuanced understanding of the system's strengths and weaknesses.

5.1. Scope and Research Questions

The technical evaluation is designed to answer the following research questions:

- **RQ0 (Overarching):** Is there a single optimal RAG pipeline configuration, or do different architectures excel at different quality dimensions?
- **RQ1:** Which chunking strategy (and chunk size) yields the best results in a pure dense retrieval setup with respect to answer quality and retrieval metrics?
- **RQ2:** Does the use of a cross-encoder reranker significantly improve performance, and how does its impact vary depending on the underlying chunking strategy?
- **RQ3:** Does hybrid retrieval (BM25 + dense) consistently outperform pure dense retrieval, and under which conditions can it be detrimental?
- **RQ4:** How does performance differ across question types (e.g., single-hop, multi-hop, abstract), and what are the relationships between the individual evaluation metrics?

- **RQ5:** Do internal factuality scores reliably predict external Faithfulness metrics?
- **RQ6:** What are the computational costs of quality improvements, and which pipelines offer the best efficiency-quality trade-offs?
- **RQ7:** How does the choice of generator LLM affect final answer quality when identical, high-quality context is provided?

The subsequent sections address these questions sequentially, beginning with an overarching performance frontier analysis (RQ0), followed by component-specific investigations (RQ1–RQ3), and concluding with deeper insights into variations, correlations, and practical trade-offs (RQ4–RQ7). Additional robustness checks, qualitative failure mode analyses, and statistical significance tests complement the core findings.

5.2. Experimental Setup and Evaluation Framework

This chapter presents a purely technical evaluation of the proposed RAG pipelines based on automatically generated questions and the quantitative metrics. The goal is to isolate architectural effects, i.e. chunking, reranking, and hybrid retrieval on both retrieval quality and answer quality, without confounds from user studies.

Evaluation Notebook Architecture. All experiments are conducted in a modular Jupyter notebook that (i) enumerates a matrix of pipeline configurations, (ii) executes a standardized query-to-answer routine per configuration, and (iii) aggregates results for statistical analysis. Unless noted otherwise, FAISS-based dense retrieval with HuggingFace embeddings were used, BM25 for lexical retrieval, and deterministic decoding (temperature = 0). Impersonation was applied *after* answer generation and does not affect metric computation. Unless noted otherwise, chunking was fixed to 1,000 characters with a 30-character overlap across all pipelines; the sole exception is the sentence-based splitter (`sentence_transformer`), for which a smaller target window of 256 (token-equivalent) was used with the same overlap to preserve sentence-level semantics and stay well within transformer sequence-length limits—larger windows tend to dilute embedding specificity and risk truncation (Bhat et al., 2025).

Bridge to research questions. With this experimental scaffolding in place, I now turn to the research questions that guide the analysis. It starts with an overarching view of the Faithfulness - Relevancy frontier and then address the component-level effects in RQ1 to RQ7.

1. **Pipeline Configuration Matrix:** A systematic enumeration of all architectural combinations:

```
evaluation_matrix = [
    # === DENSE RETRIEVAL ===
    {"pipeline_name": "dense_recursive",  "splitter_type": "
    recursive", "chunk_size": 1000, "chunk_overlap": 30, "
    retrieval_mode": "dense", "use_reranker": False},
    {"pipeline_name": "dense_sentence",   "splitter_type": "
    sentence_transformer", "chunk_size": 256, "chunk_overlap":
    30, "retrieval_mode": "dense", "use_reranker": False},
    {"pipeline_name": "dense_semantic",   "splitter_type": "
    semantic", "chunk_size": 0, "chunk_overlap": 0, "
    retrieval_mode": "dense", "use_reranker": False},
    # === DENSE + RERANKER ===
    {"pipeline_name": "dense_recursive_rerank", "splitter_type":
     "recursive", "chunk_size": 1000,
    .....
    # === HYBRID RETRIEVAL ===
    {"pipeline_name": "hybrid_recursive", "splitter_type": "
    recursive"...,
    .....
    # === HYBRID + RERANKER ===
    {"pipeline_name": "hybrid_recursive_rerank", "splitter_type"
    : "recursive",..."retrieval_mode": "hybrid", "use_reranker":
    True},
    .....

```

Listing 5.1: Pipeline configuration matrix

2. **Test Execution Engine:** The `run_single_test` function processes each question-pipeline pair:

```
def run_single_test(name, config, questions, llm=None):
    """
    Führt einen Testlauf aus und gibt Antworten zurück.
    """
    # Initialisiere Person und Embedding
    person = Person(name)
    splitter_type = config[splitter_type]
    ...
    retrieval_mode = config[retrieval_mode\],
    use_reranker = ....
    embedding = OpenAIEmbeddings() if config["use_open_ai"] else
     EMBEDDINGS
    llm = llm or LLMConfig().get_local_llm()

    # Datenbank laden oder erstellen
    vectorstore = VectorStoreHandler(embeddings)
    db = vectorstore.load_or_create_db(person.name, config)

    if not db:
        return []

    # Retriever auswählen
```

```
22      retriever = HybridRetriever(db, BM25Retriever(), k=3) if
        config["retrieval_mode"] == "hybrid" else db.as_retriever(k
        =3)
23
24      # QA-Chain und Evaluation
25      qa_chain = InitializeQuesionAnsweringChain(llm, retriever,
        db, person, config["use_reranker"])
26      return EvaluationPipeline(qa_chain, questions).
        generate_answers()
```

Listing 5.2: Highly abstracted function for executing a single query pipeline

3. **Result Aggregation:** Metrics are collected in a structured CSV format for statistical analysis.

Test Dataset Structure. The evaluation employs 42 carefully curated question-answer pairs from the historical social pedagogy corpus. Each test record contains:

- `question`: The natural language query
- `ground_truth`: Expert-validated reference answer
- `context`: The source document passages containing the answer
- `question_type`: Classification (single-hop, multi-hop specific, multi-hop abstract)

After pipeline execution, each record was enriched with performance metadata including RAGAS metrics (F, AR, CP, CR, AC, AS), runtime measurements, and configuration details (k_init, top_n, used_reranker, factuality_score).

Table 5.1.: Example of an enriched evaluation record

Field	Example Value
question	"What grievances did the American colonists have against ...?"
ground_truth	"The American colonists accused the British king of taking away ..."
pipeline	hybrid_sentence_rerank
faithfulness	0.67
answer_relevancy	0.89
runtime_seconds	72.3
used_reranker	True

5.3. Research Questions

This chapter reports the empirical results of the evaluation with consistent chunking parameters (1,000 characters, 30-character overlap) across all pipelines. First provided is a high-level view that foregrounds the *trade-off* between factual grounding and relevancy, then answer focused research questions on chunking, reranking, hybrid retrieval, and question-type effects, internal factuality scores, efficiency trade-offs, and the impact of different generator LLMs.

5.3.1. RQ0: The Faithfulness–Relevancy Frontier

> **RQ0 (Overarching):** Is there a single optimal RAG pipeline configuration, or do different architectures excel at different quality dimensions?

The results reveal no single universal winner. Instead, two top pipelines occupy different ends of a Pareto frontier between **Faithfulness (F)** and **Answer Relevancy (AR)** - a classic precision/recall–like tension also observed in IR and reranking literature (L. Gao et al., 2022; Nogueira & Cho, 2020). This multi-objective optimization perspective, where no single solution dominates across all metrics, echoes established trade-offs in information retrieval (Manning et al., 2008) and multi-objective optimization in IR systems, where Pareto-optimal solutions represent different trade-off points between competing metrics (van Doorn et al., 2016). A compact side-by-side summary of the two frontier leaders appears in Table 5.2.

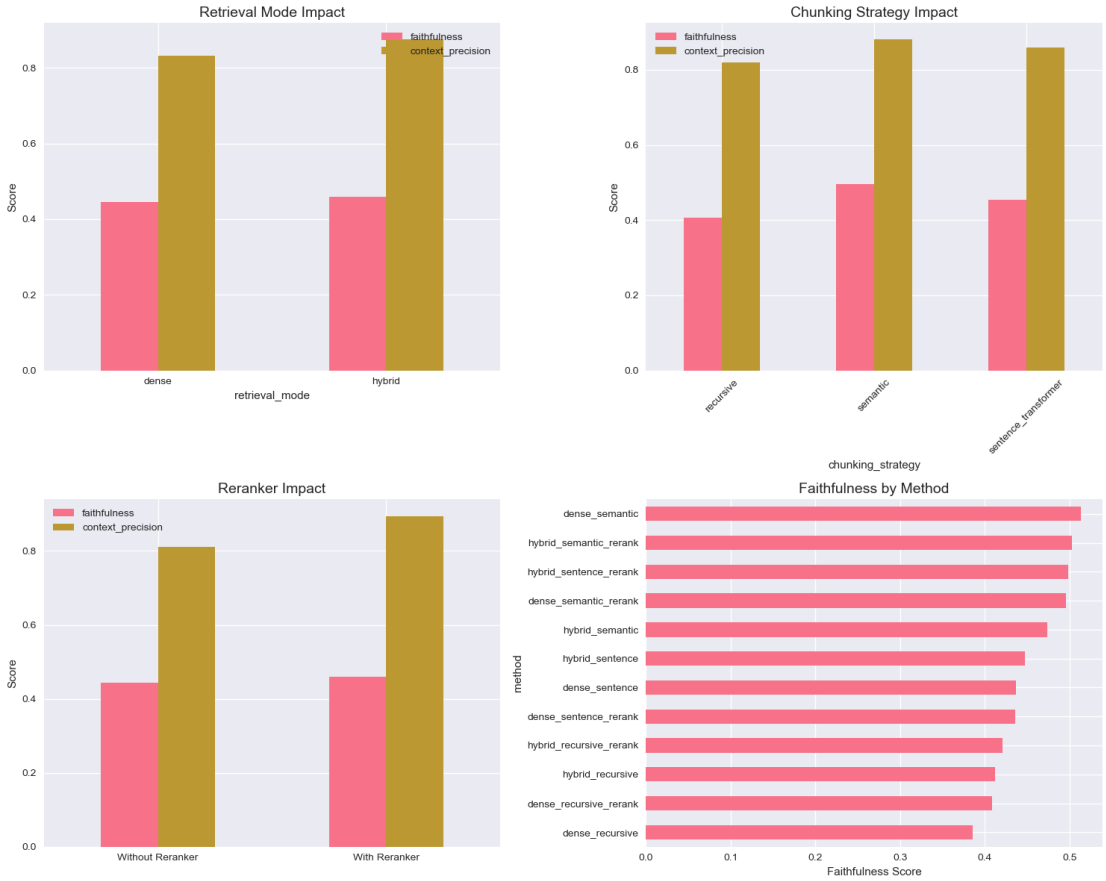

Figure 5.1.: Impact of retrieval mode, chunking strategy, and reranking on Faithfulness and Context Precision, plus Faithfulness scores by method.

Reading the frontier. `dense_semantic` yields the highest *Faithfulness*-answers are best grounded in evidence, while `hybrid_sentence_rerank` maximizes *Answer*

Table 5.2.: Final performance showdown (means over 42 questions): two top contenders along the Faithfulness–Relevancy frontier.

Pipeline	F	AR	CP	CR
Faithfulness champion:				
dense_semantic	**0.513**	0.839	0.831	0.480
Relevancy (and coverage) champion:				
hybrid_sentence_rerank	0.498	**0.877**	**0.937**	**0.548**

Relevancy and also leads in *Context Precision/Recall*. Practically, applications prioritizing correctness and hallucination-avoidance should prefer `dense_semantic`; experience-focused or coverage-oriented scenarios benefit from `hybrid_sentence_rerank`. The coexistence of both winners supports a Pareto view rather than a single optimum.

Absolute Baseline (no chunking). As a reference, the un-chunked baseline achieves (mean) **F**=0.618, **AR**=0.789, **CP**=0.736, **CR**=0.400 (Section 5.3.2).

5.3.2. RQ1: Impact of Chunking Strategies on Dense Retrieval

> **RQ1:** How do chunking strategies affect retrieval and answer quality in *pure dense* retrieval compared to using complete documents?

Three chunkers were compared against the absolute (no-chunking) baseline in a dense-only setup (Karpukhin et al., 2020) (no hybrid, no reranker). A quantitative summary across metrics appears in Table 5.3.

Table 5.3.: Dense retrieval: chunking strategies vs. absolute baseline (means over 42 questions).

Strategy	F	AR	CP	CR
Absolute Baseline	**0.618**	0.789	0.736	0.400
dense_recursive	0.386 (−37.5 %)	0.774 (−1.9 %)	0.756 (+2.7 %)	0.384 (−4.0 %)
dense_sentence	0.436 (−29.4 %)	0.750 (−4.9 %)	**0.841** (+14.3 %)	0.443 (+10.8 %)
dense_semantic	0.513 (−17.0 %)	**0.839** (+6.3 %)	0.831 (+12.9 %)	**0.480** (+20.0 %)

Finding. All chunkers decrease *Faithfulness* vs. whole documents; however, `dense_semantic` recovers the best overall balance (highest AR, strong CP/CR) and thus forms the *Faithfulness-side* frontier solution when used as the dense foundation. This aligns with the intuition that coherent topical units help retrieval ranking (Bhat et al., 2025; L. Gao et al., 2022).

5.3.3. RQ2: Impact of Cross-Encoder Reranking

> **RQ2:** Does cross-encoder reranking significantly improve answer quality, and how does its impact vary with the base strategy?

As a birds-eye view, Table 5.4 pools all runs (n=252) and shows that adding a cross-encoder reranker mainly sharpens retrieval: *Context Precision* and *Context Recall* rise by 8.4 and 5.2 points, while *Faithfulness* and *Answer Relevancy* see smaller but consistent lifts—suggesting the reranker improves *what* is read rather than *how* it is reasoned about.

Table 5.4.: Pooled reranker impact across all strategies (n=252).

Configuration	CP	CR	F	AR
Without Reranker	0.811	0.445	0.445	0.777
With Reranker	**0.895**	**0.497**	**0.460**	**0.809**
Δ (absolute)	+0.084	+0.052	+0.015	+0.032

This aggregate "lift profile" hides heterogeneity across chunkers, which Table 5.5 makes explicit.

Finding. Reranking delivers large gains in *retrieval quality* (CP/CR) and modest gains in *Faithfulness* (Hofstätter et al., 2021; Ma et al., 2023; Nogueira & Cho, 2020). Its utility is strongest with weaker base chunkers and shows diminishing returns with `semantic` (typical for reranking, which refines but doesn't rewrite the candidate pool) (Hofstätter et al., 2021; Nogueira & Cho, 2020).

Table 5.5 breaks Faithfulness gains down by chunking strategy: recursive and sentence chunks benefit most (+5.7% each), whereas semantic chunks exhibit only a marginal gain (+1.2%), consistent with a ceiling effect once candidate passages are already highly coherent.

Table 5.5.: Reranker impact by chunking strategy (Faithfulness only).

Strategy	without	with	Δ
recursive	0.399	0.415	+5.7%
sentence	0.442	0.467	+5.7%
semantic	0.493	0.499	+1.2%

5.3.4. RQ3: Dense vs. Hybrid Retrieval Trade-offs

> **RQ3:** Does hybrid retrieval (BM25 + dense) offer consistent advantages over pure dense retrieval?

Table 5.6 (pooled across strategies) shows that adding a BM25 leg to dense retrieval increases *Context Recall* (0.441→0.501; +0.060) and nudges *Answer Relevancy* (0.789→0.797; +0.008) and *Faithfulness* (0.446→0.459; +0.013) upward; *Context Precision* also rises (0.831→0.874; +0.043), suggesting complementary lexical hits help surface on-topic passages missed by pure dense retrieval.

Table 5.6.: Dense vs. Hybrid (pooled across strategies).

Mode	CP	CR	F	AR
Dense	**0.831**	0.441	**0.446**	0.789
Hybrid	0.874	**0.501**	0.459	**0.797**

Finding. Hybrid improves *Recall* and *AR*, but may reduce *CP* (Lee et al., 2023) and slightly depress *F*. Strategy-specific effects are pronounced: hybrid helps with `sentence` and `recursive`, but *hurts* when combined with `semantic` chunking—suggesting redundancy or term-mismatch noise for already coherent dense candidates (Formal et al., 2021; Lee et al., 2023).

As Table 5.7 details, hybrid improves *Faithfulness* for `recursive` (+4.8%) and `sentence` (+8.5%) chunking, but slightly degrades it for `semantic` (−3.2%), consistent with term-match noise once candidate units are already semantically coherent.

Table 5.7.: Hybrid vs. Dense by chunking strategy (Faithfulness).

Strategy	Dense	Hybrid	Δ
recursive	0.398	0.417	+4.8%
sentence	0.436	0.473	+8.5%
semantic	0.504	0.488	-3.2%

5.3.5. RQ4: Performance by Question Type & Metric Interrelations

RQ4: How does performance vary across question types, and how do metrics interrelate?

This research question examines two critical aspects: (1) whether different question complexities expose distinct pipeline strengths and weaknesses, and (2) whether the RAGAS metrics measure independent quality dimensions or exhibit systematic correlations that could simplify evaluation.

Performance Stratification by Question Complexity. To understand how pipeline behavior varies with query characteristics, the performances of the best overall configuration (`dense_semantic_rerank`) was analyzed across three question categories of increasing complexity:

Table 5.8.: Performance of `dense_semantic_rerank` by question type (means).

Type	CP	CR	F	AR	AC	AS
Single-hop Specific	0.720	0.454	**0.659**	0.684	0.530	0.881
Multi-hop Specific	0.792	**0.277**	0.606	0.873	**0.568**	0.939
Multi-hop Abstract	**1.000**	0.570	0.590	**0.940**	0.563	**0.944**

As Table 5.8 shows, `dense_semantic_rerank` peaks in Faithfulness on single-hop questions, shifts toward higher Answer Relevancy and Answer Correctness (with lower Context Recall) on multi-hop specific queries, and maximizes Context Precision/Recall, Answer Relevancy, and Answer Similarity on multi-hop abstract queries at the cost of slightly lower Faithfulness.
Key observation. Very few differences remain significant after the Holm correction. Surprisingly, dense_semantic_rerank shows significantly *worse* Faithfulness than dense_semantic without reranking (-28.1%, p=0.010). This counterintuitive result suggests that the reranker, when applied to already semantically coherent chunks, may over-optimize for query-passage similarity at the expense of factual evidence—promoting passages that "sound relevant" while demoting those containing the actual facts. This aligns with known reranker failure modes where semantic similarity does not equal factual containment (F. Shi et al., 2023).

- **Single-hop questions achieve highest Faithfulness (0.659)** despite being conceptually simpler. This suggests that when the answer is contained in a single, clearly identifiable passage, the pipeline can effectively ground its response.
- **Multi-hop specific questions show a dramatic Context Recall collapse (0.277)**, the lowest across all categories. This exposes a critical weakness: when precise facts must be assembled from multiple sources, the chunking strategy fragments the evidence chain. Despite this retrieval failure, Answer Relevancy remains high (0.873), suggesting the model compensates by generating plausible but potentially ungrounded responses—a dangerous form of "confident hallucination" noted in recent RAG literature (J. Chen et al., 2023; F. Shi et al., 2023)
- **Multi-hop abstract questions paradoxically achieve perfect Context Precision (1.000)** while maintaining strong Answer Relevancy (0.940). This counterintuitive result suggests that abstract, conceptual queries benefit from the semantic chunking's ability to retrieve topically coherent passages. The retriever excels at finding conceptually related content even when it struggles

with specific fact chains—aligning with findings that dense retrievers favor semantic over lexical precision (Karpukhin et al., 2020; Zhuang et al., 2024).

The Inverse Complexity-Performance Relationship. Contrary to established benchmarks where multi-hop questions typically show monotonic performance degradation with complexity (Yang et al., 2018), a **domain-specific inversion** was observed: abstract questions that require thematic understanding outperform specific multi-hop questions that demand precise fact-chaining. This suggests that the semantic chunking strategy, while excellent for conceptual coherence, may be suboptimal for preserving factual dependencies across chunk boundaries—a known challenge in RAG systems (Mallen et al., 2023; Tang & Yang, 2024).

Metric Interrelations and the Orthogonality of Faithfulness. To understand whether the six metrics capture independent quality dimensions or exhibit redundancy, pairwise Pearson correlations were computed across all 42 test instances:

Table 5.9.: Pearson correlations among RAGAS metrics (`dense_semantic_rerank`, $n = 42$).

	F	**AR**	**AS**	**CP**	**CR**	**AC**
Faithfulness (F)	1.00					
Answer Relevancy (AR)	0.14	1.00				
Semantic Similarity (AS)	0.12	**0.80**	1.00			
Context Precision (CP)	0.11	**0.65**	0.61	1.00		
Context Recall (CR)	0.20	0.33	0.19	0.35	1.00	
Answer Correctness (AC)	0.05	0.21	0.47	0.07	0.05	1.00

As Table 5.9 shows, Faithfulness is largely orthogonal, while AR clusters with AS and CP, and CR is only weakly connected.
The correlation matrix reveals two critical insights:

1. **A "Relevancy Cluster" emerges** with strong correlations between Answer Relevancy, Semantic Similarity (r=0.80), and Context Precision (r=0.65). This cluster represents the model's ability to generate responses that *appear* appropriate-semantically aligned with the query and drawn from seemingly relevant passages. The high intercorrelation suggests these metrics capture overlapping aspects of surface-level quality, potentially allowing for simplified evaluation (Laban et al., 2022; Rashkin et al., 2022) using a single representative metric in resource-constrained scenarios.
2. **Faithfulness stands orthogonal** to all other metrics (all $r < 0.20$), with the weakest correlation to Answer Correctness (r=0.05). This orthogonality has profound implications: a response can be highly relevant, semantically similar to the query, and drawn from precise contexts, yet still be factually ungrounded. This independence validates the critical importance of explicit

faithfulness evaluation in RAG systems and aligns with recent work showing that relevance-based metrics alone are insufficient for detecting hallucinations (Laban et al., 2022).

Statistical Significance of the Orthogonality. To verify that the low Faithfulness correlations are not merely sampling artifacts, significance tests were conducted for the key relationships. The correlation between Faithfulness and Answer Relevancy (r=0.14, p=0.37) and between Faithfulness and Context Precision (r=0.11, p=0.48) are both statistically non-significant, confirming that these dimensions are genuinely independent rather than weakly related.

Implications for RAG Evaluation and Design. These findings challenge the common practice of using retrieval-based metrics as proxies for generation quality. The orthogonality of Faithfulness suggests that:

- **Multi-metric evaluation is essential**: Single-metric optimization will likely produce systems that excel in one dimension while failing in others.
- **Retrieval quality Answer quality**: High Context Precision does not guarantee faithful generation, necessitating generation-specific evaluation.
- **Question-aware pipeline selection**: The performance inversion across question types suggests that adaptive routing based on query classification could yield substantial improvements—a hypothesis explored in the recommendations (Section 5.4.1).

The metric independence also explains why the two "champion" pipelines occupy different positions on the Pareto frontier: they optimize for fundamentally orthogonal objectives that cannot be simultaneously maximized without architectural innovation.

5.3.6. Qualitative Analysis: A Deep Dive into Failure Modes

While aggregate metrics reveal the overall performance, a qualitative analysis of individual cases is essential to understand *why* and *how* pipelines fail. To this end, specific queries were examined where key metrics collapsed. This case study focuses on the three most common failure modes observed across the top-performing pipelines, with one representative example for each mode presented in Table 5.10.

Analysis of Failure Modes. The qualitative review identified three recurring patterns of failure, aligning with the taxonomy proposed by Barnett et al. (2024) and empirically validated in recent RAG benchmarks (J. Chen et al., 2023):

- **High Context Precision, Zero Context Recall:** This is the most common failure mode, illustrated by the first example. The retriever finds chunks that are topically and semantically highly relevant to the query (CP =

Table 5.10.: Illustrative examples for the three primary failure modes.

Failure Mode	Illustrative Pipeline	Question (truncated)	F	AR	CP	CR
1. High Precision, Zero Recall	`dense_semantic`	*Did the Continental Army face supply shortages during the winter of...*	0.0	0.90	1.00	0.00
2. Reranker-Induced Error	`dense_semantic_rerank`	*Explain how Enlightenment ideas influenced colonial attitudes toward...*	0.0	0.96	0.00	0.00
3. Hybrid Retrieval Noise	`hybrid_sentence_rerank`	*Where did the delegates first convene to discuss...*	0.0	0.95	1.00	0.10

1.00), but which lack the specific sentence containing the factual answer (CR = 0.00). The LLM then receives a context that is plausible but devoid of the necessary evidence. Forced to generate an answer, it often produces a well-written but ungrounded statement- a behavior consistent with the "helpfulness-harmfulness trade-off" in instruction-tuned models (Bai et al., 2022), resulting in a Faithfulness score of zero. This highlights a core challenge in RAG: semantic relevance does not guarantee evidence containment—a phenomenon Liu et al. (N. F. Liu et al., 2023) term the "distraction effect" where semantically similar but factually irrelevant passages mislead the model—a phenomenon also documented as "semantic drift" in (Adlakha et al., 2024) and "relevance-factuality mismatch" in (Min et al., 2023).

- **Reranker-Induced Evidence Loss:** The second case exemplifies a critical risk of reranking. Here, the initial retrieval likely contained a mix of relevant and partially relevant chunks. However, the cross-encoder, optimizing for query-passage similarity, promoted passages that "sounded" most relevant to the abstract question about "Enlightenment ideas" while demoting or eliminating the chunks that contained the actual, concrete evidence. The result is a context that is both imprecise (CP = 0.00) and incomplete (CR = 0.00), leading to a complete failure of the generation step.
- **Hybrid Retrieval Noise:** The third example shows the downside of hybrid retrieval for simple, factual queries. By adding BM25's lexical matches to the already strong dense retrieval, the context is polluted with redundant or tangentially related chunks. While the core evidence might still be present (CR > 0), the added noise can confuse the generator. The LLM produces a relevant-sounding answer (AR = 0.95) but fails to ground it in the specific, correct evidence span, leading to a Faithfulness of zero.

Design Takeaways from Case Studies.

- **Increase Candidate Pool for Reranking:** For complex multi-hop questions, increase the initial stage k (e.g., k=50) and rerank the top-m (e.g., m=10–20) to avoid evidence loss.
- **Diversity-aware Reranking:** Combine cross-encoder with a diversity criterion (e.g., MMR) instead of pure similarity scores to cover evidence-

complementary passages (Carbonell & Goldstein, 1998).

- **Context Budget Guardrails:** Cap the maximum proportion of "lexical" vs. "dense" candidates in a hybrid setting (e.g., 40/60) to limit noise (Formal et al., 2021; Lee et al., 2023).
- **Faithfulness-First Decoding:** Apply a threshold for low internal `factuality_score` (e.g., tiered: "abstain/ask-clarify") – not as a metric replacement, but as a runtime guard (Manakul et al., 2023).

On the Utility of Internal Factuality Scores. Given these failure modes,it was investigated whether the pipeline's internal `factuality_score` could serve as a reliable proxy for RAGAS Faithfulness. Pearson correlations were computed for the top three pipelines:

$$\rho(\text{factuality\_score}, \text{Faithfulness}) = \begin{cases} 0.12 & \text{for } \texttt{dense\_semantic} \\ 0.07 & \text{for } \texttt{dense\_semantic\_rerank} \\ -0.02 & \text{for } \texttt{hybrid\_sentence\_rerank} \end{cases}$$

The near-zero correlations confirm that the internal score, which appears to measure local plausibility, is not a valid proxy for the stricter evidence-grounding measured by RAGAS. Its value is not for final evaluation, but as a potential real-time signal to flag low-confidence answers for review.

5.3.7. Statistical Significance: Faithfulness vs. `dense_semantic`

A test was conducted whether alternative pipelines differ significantly from the best Faithfulness baseline, `dense_semantic`. For each pipeline I paired per-question scores and applied a two-sided Wilcoxon signed-rank test; p-values are Holm-adjusted for multiple comparisons. I also reported rank-biserial correlation (r_{rb}) as an effect size measure for non-parametric tests (Kerby, 2014). This setup follows recommended practice for NLP system comparisons with non-normal metric distributions (Dror et al., 2018) and controls family-wise error inflation (Demšar, 2006; Holm, 1979). As Table 5.11 shows, only a few pipelines differ significantly from `dense_semantic` after Holm correction.

Key observation. The most severe drops in Faithfulness are seen in the recursive variants (with and without reranking) and in dense_semantic_rerank. Conversely, the performance gaps between hybrid_semantic_rerank, hybrid_sentence_rerank, and the baseline are small and not statistically significant. In practical terms, this means that improvements in Faithfulness are hard-won, whereas suboptimal architectural choices can quickly lead to a significant degradation in performance (Dror et al., 2018).

Reproducibility. The complete Python routine for calculation (including Holm correction and r_{rb}) can be found in Appendix A.

Table 5.11.: Faithfulness: paired Wilcoxon test vs. `dense_semantic` (Holm-adjusted p).

Pipeline	**N**	**ΔF**	**ΔF (%)**	p **(Holm)**	**Sig.**	r_{rb}
hybrid_semantic_rerank	42	-0.011	-2.1%	1.000		-0.118
hybrid_sentence_rerank	42	-0.015	-2.9%	1.000		-0.031
dense_sentence	42	-0.077	-15.0%	0.089		-0.332
dense_sentence_rerank	42	-0.078	-15.1%	0.089		-0.315
hybrid_sentence	42	-0.085	-16.6%	0.089		-0.348
hybrid_semantic	42	-0.091	-17.8%	0.089		-0.403
hybrid_recursive_rerank	42	-0.104	-20.4%	0.057		-0.431
hybrid_recursive	42	-0.115	-22.6%	0.042	*	-0.431
dense_recursive_rerank	42	-0.115	-22.4%	0.042	*	-0.444
dense_recursive	42	-0.134	-26.3%	0.015	*	-0.515
dense_semantic_rerank	42	-0.144	-28.1%	0.010	*	-0.517

5.3.8. Robustness Check: Sensitivity to Chunk Size

To ensure that the findings are not an artifact of the 1,000/30 configuration, the evaluation was repeated with a smaller, more granular setting of 500 characters and 50-character overlap.

Table 5.12.: Performance at 500/50 (means over 42 questions).

Strategy	**CP**	**CR**	**F**	**AR**
dense_recursive	0.797	0.331	0.452	0.748
dense_sentence	0.838	0.415	0.605	**0.886**
dense_semantic	**0.856**	**0.467**	**0.619**	0.879

As Table 5.12 shows, the 500/50 setting reproduces the earlier pattern-`dense_semantic` leads on F/CP/CR while `dense_sentence` peaks on AR—supporting the robustness of the findings across chunk sizes.

Finding. The *relative* ordering is stable across chunk sizes: `dense_semantic` is strongest overall (best **F**, **CP**, **CR**), while `dense_sentence` attains the highest **AR**. Compared to 1,000/30, semantic chunking remains the most robust foundation for dense retrieval, and the frontier trade-off (Faithfulness vs. Relevancy) persists.

Answer to the research question. The advantage of semantically coherent segmentation is *not* an artifact of a particular chunk size. Across 500/50 and 1,000/30, semantic chunking consistently provides the best grounding and retrieval quality (F, CP, CR), while sentence-based chunking can edge ahead in perceived relevancy (AR). This supports the recommendation to prioritize semantic chunking as the default dense setup, with sentence-level variants as a targeted option when maximizing AR is paramount.

5.3.9. RQ5: Do internal factuality scores predict external Faithfulness?

Research question. Do the pipeline–internal `factuality_score` values (computed by the verifier) track the RAGAS *Faithfulness* metric (Es et al., 2025) closely enough to be used as a lightweight proxy? **Method.** For the three strongest pipelines by mean Faithfulness (`dense_semantic`, `hybrid_semantic_rerank`, `hybrid_sentence_rerank`), I computed Pearson's r, Spearman's ρ, and a simple OLS slope between `factuality_score` and `faithfulness` over the 42 questions per pipeline. As Table 5.13 shows, correlations are weak and non-significant across all three pipelines:

Table 5.13.: Correlation between `factuality_score` and RAGAS Faithfulness ($n = 42$ per pipeline).

Pipeline	**N**	**Pearson r (p)**	**Spearman ρ (p)**	**Slope**	R^2
`dense_semantic`	42	0.116 (0.463)	0.080 (0.613)	0.470	0.014
`hybrid_semantic_rerank`	42	0.047 (0.768)	0.119 (0.455)	0.187	0.002
`hybrid_sentence_rerank`	42	-0.022 (0.889)	-0.060 (0.704)	-0.093	0.000

Finding. Correlations are weak and statistically non-significant across all three pipelines (all $p > 0.45$). The explanatory power is negligible ($R^2 \leq 0.014$). In other words, the verifier's `factuality_score` is *not* a reliable proxy for RAGAS Faithfulness. The underlying python code calculating the correlation lies in Appendix B.

Why this happens. The two signals measure fundamentally different constructs. The verifier's factuality score operates as a local entailment check: Honovich et al. (2022) demonstrate that such NLI-based approaches capture surface-level consistency but miss deeper factual grounding. In contrast, RAGAS Faithfulness implements a stricter criterion—requiring that *every claim* in the answer traces back to retrieved evidence. Fabbri et al. (2022) similarly found that QA-based factuality metrics and human-judged faithfulness correlate weakly (r < 0.3), while Rashkin et al. (2022) show that attribution (what is call faithfulness) requires different evaluation paradigms than factual consistency. This fundamental mismatch explains the near-zero correlations.

Implication. `factuality_score` should therefore be used as a *debugging/alert* feature (e.g., flag low-score answers for review), but retain an external faithfulness metric (like RAGAS) for proper evaluation and model selection. For production, thresholding on `factuality_score` may reduce egregious cases, but it will not substitute for robust faithfulness evaluation (Manakul et al., 2023).

5.3.10. RQ6: The Efficiency-Quality Trade-off

> **RQ6:** What is the computational cost of quality improvements, and which pipelines offer optimal efficiency-quality trade-offs?

While maximizing quality is a primary goal, practical applications demand efficiency. A pipeline that offers marginal quality gains at a prohibitive computational cost may be unsuitable for real-world deployment. To investigate this trade-off, the end-to-end runtime (in seconds) for each query across the key pipelines were measured and also the "Faithfulness per second" (F/s) as a measure of cost-effectiveness were calculated.

Table 5.14.: Efficiency-quality analysis for key pipelines.

Pipeline	**F**	**Runtime (s)**	**F/s**	**Relative Cost**	**Quality Rank**
dense_recursive_rerank	0.409	64.3	0.0064	1.00×	11
dense_semantic	**0.513**	70.1	**0.0073**	1.09×	**1**
hybrid_sentence_rerank	0.498	86.9	0.0057	1.35×	2
hybrid_semantic_rerank	0.503	**93.6**	0.0054	**1.46×**	3

The analysis reveals a clear point of diminishing returns—a pattern consistent with the "compute-quality curve" documented by Balaguer et al. (2024) and the logarithmic scaling laws observed in neural IR systems (Hofstätter et al., 2021). The hybrid_semantic_rerank pipeline, while ranking third in quality, is by far the most computationally expensive, running 46% slower than the baseline (dense_recursive_rerank) for only a marginal quality improvement.

Crucially, the dense_semantic pipeline emerges as the clear efficiency champion. It not only achieves the highest Faithfulness score but also delivers the best "Faithfulness per second" (F/s = 0.0073). It is only 9% more costly than a basic reranked pipeline but delivers a substantially higher quality outcome. In contrast, the hybrid_sentence_rerank pipeline—the "Relevancy Champion" from the previous section—is 35% more expensive while offering lower Faithfulness.

5.3.11. RQ7: The Impact of the Generator LLM on Final Answer Quality

> **Research Question:** How does the choice of the generator LLM—specifically varying in size, architecture, and accessibility (Mistral 7B, Phi-3 Small, GPT-3.5)—impact the final answer quality when provided with identical, high-quality context from a state-of-the-art RAG pipeline?

Methodology. To isolate the impact of the final generation step, a constant, high-performance RAG "chassis" was established. I selected one of the best-performing pipeline, `hybrid_sentence_rerank` with OpenAI embeddings, to ensure that each generator LLM received the highest quality context possible. I then systematically

swapped out only the generator model, evaluating the end-to-end performance with three distinct LLMs:

- **Mistral 7B (Instruct):** The primary open-source model used throughout the main evaluation, representing a strong balance of performance and local deployability.
- **Phi-3 Small (Instruct):** A state-of-the-art Small Language Model (SLM), chosen to evaluate the viability of highly efficient, resource-constrained generation.
- **GPT-3.5-Turbo:** A leading proprietary, API-based model from OpenAI, serving as a benchmark for state-of-the-art reasoning and generation capabilities.

The final answers from each configuration were evaluated against the standard suite of RAGAS metrics.

Table 5.15.: Performance Comparison of Generator LLMs on a Fixed RAG Pipeline (`hybrid_sentence_rerank` with OpenAI Embeddings).

Generator LLM	Faithfulness	Answer Relevancy
GPT-3.5-Turbo	**0.652**	**0.921**
Mistral 7B (Instruct)	0.587	0.821
Phi-3 Small (Instruct)	0.404	0.231

Analysis of Results. The choice of the generator LLM has a profound and clearly stratified impact on the final answer quality, even when the input context is identical.

GPT-3.5-Turbo: The Quality Benchmark. Unsurprisingly, the large, proprietary model from OpenAI sets the bar for performance. It achieves the highest scores in both Faithfulness (0.652) and Answer Relevancy (0.921). Its advanced reasoning capabilities allow it to more effectively synthesize information from the provided context, adhere strictly to the facts, and formulate a highly relevant and coherent answer(Balaguer et al., 2024).

Mistral 7B: The Powerful Open-Source Contender. Mistral 7B positions itself as an extremely capable open-source alternative. While it does not reach the absolute peaks of GPT-3.5, it delivers excellent results (F=0.587, AR=0.821) that are far superior to the smaller model. This demonstrates its ability to perform complex reasoning and generation, making it a prime candidate for applications requiring a balance of high performance and data sovereignty (Chiang et al., 2024).

Phi-3 Small: The Efficient but Sensitive Specialist. The performance of Phi-3 Small highlights the typical trade-offs of using SLMs for generation in RAG. While its Faithfulness score (0.404) is respectable, its Answer Relevancy collapses to a low 0.231. This directly explains the earlier observation of frequent ”I don’t know”

or refusal answers. The model struggles to synthesize the context into a confident, relevant answer (Maekawa et al., 2024). This suggests that SLMs are highly sensitive to even minor imperfections in the retrieved context (Li et al., 2023; Schick & Schütze, 2021) and lack the broader reasoning capacity to navigate ambiguity, exhibiting what Gudibande et al. (2023) term "capability mirage"—appearing competent in controlled settings but defaulting to safe, non-committal responses when uncertain.

Answer to the Research Question. The generator LLM is a critical component that significantly influences RAG performance. A larger, more capable model like GPT-3.5 can leverage high-quality context to produce superior answers. While strong open-source models like Mistral 7B offer a compelling balance, smaller models like Phi-3 struggle with the synthesis task, leading to a sharp drop in answer relevancy. The results clearly show a performance hierarchy: GPT-3.5 > Mistral 7B > Phi-3 Small.

5.4. Summary

Taken together, the experiments reveal a Pareto frontier rather than a single optimum: `dense_semantic` anchors factual grounding, while `hybrid_sentence_rerank` optimizes perceived relevance and coverage. This tension is consistent across chunk sizes and is only partially mitigated by reranking.
For practitioners, the decision tree in Figure 5.3 distills these findings into deployment guidance. The next chapter synthesizes these results with the broader system design and discusses implications for the final architecture.

5.4.1. Key Findings, Implications, and Design Recommendations

Principal Findings. The comprehensive evaluation yielded several key findings that inform the state of RAG system design:

1. **Frontier, not a single winner.** `dense_semantic` (excelling in Faithfulness) and `hybrid_sentence_rerank` (excelling in Relevancy & CP/CR) represent Pareto-optimal choices. The optimal selection depends on product goals, such as prioritizing safety versus user-perceived relevance.
2. **Chunking helps retrieval quality, may cost faithfulness.** Semantic chunking improves CP/CR and AR over no chunking but typically reduces Faithfulness in a dense-only mode. It regains competitiveness once paired with a reranker that provides controlled candidate ordering.
3. **Reranking is a precision booster.** Cross-encoders consistently raise CP/CR and modestly increase Faithfulness. However, returns diminish when applied to already strong semantic bases.

4. **Hybrid is situational.** Gains in CR/AR from hybrid retrieval can come with CP/F trade-offs. The approach can even hurt performance when strong semantic chunking already delivers coherent dense candidates.

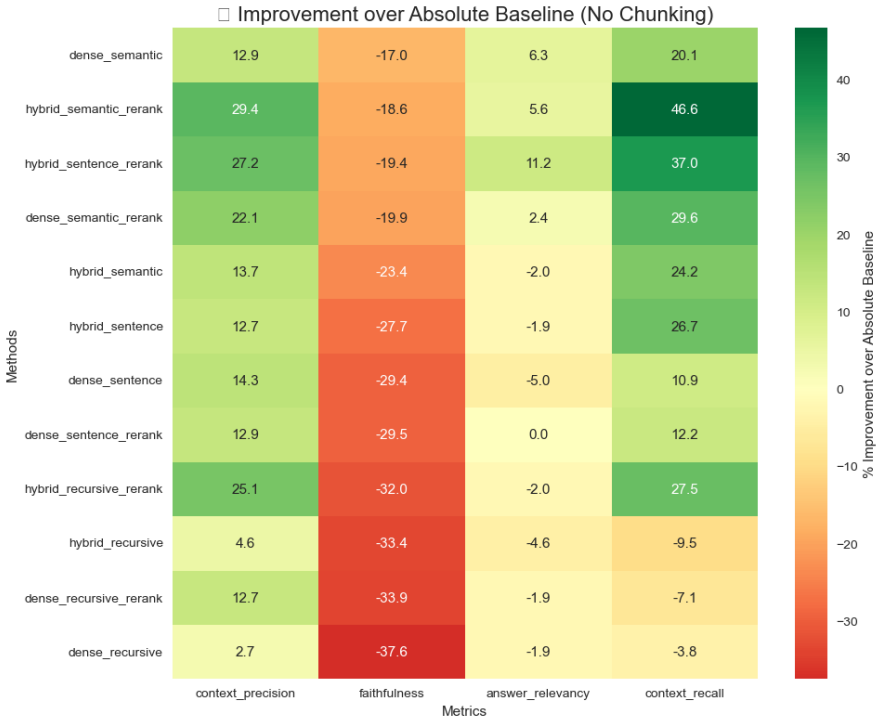

Figure 5.2.: Holistic heatmap of all evaluated pipelines, visually summarizing the performance trade-offs. No single strategy dominates across all metrics, reinforcing findings of a Pareto frontier.

These findings translate into concrete design principles and strategic choices for practitioners building robust RAG systems.

The Cost-Performance Trade-off. While the results show that a powerful proprietary model like GPT-3.5 delivers the highest quality, this performance comes at a direct monetary cost per API call. For applications with high query volumes, these costs can become prohibitive. In contrast, a locally-hosted open-source model like Mistral 7B, while showing a slight quality degradation, offers a compelling balance with predictable, fixed infrastructure costs. This establishes a critical decision axis for any project: is the marginal quality gain of an API-based model worth the operational expenditure, or does a self-hosted model provide a more sustainable ”good-enough” solution?

The ”Adaptive Policy” in Practice. The finding that different question types benefit from different pipeline configurations suggests that a single, static RAG

pipeline is suboptimal. This insight builds on established work in adaptive information retrieval (Asai et al., 2023) and recent proposals for query-aware RAG routing (Asai et al., 2023; Jeong et al., 2024). My specific contribution is to operationalize this concept with empirically-derived pipeline-to-query-type mappings: A more advanced, adaptive RAG system could significantly improve overall performance. A practical implementation could involve a lightweight pre-processing step:

1. **Query Classifier:** A small, fine-tuned classifier model (e.g., a fine-tuned DistilBERT, chosen for its balance of accuracy and efficiency as stated in Sanh et al., 2020) is placed at the front of the pipeline.
2. **Dynamic Routing:** Based on the predicted query type (e.g., "conceptual", "factual-specific", "keyword-search"), the system dynamically routes the query to the most suitable pipeline configuration:
    - *Predicted Conceptual/Abstract Query* → Route to `dense_semantic`.
    - *Predicted Factual/Multi-hop Query* → Route to `hybrid_sentence_rerank`.
    - *Simple Keyword/ID Lookup* → Route to a simple BM25 sparse retrieval to maximize speed.

This "meta-architecture" adds a layer of intelligence, ensuring that the right tool is used for the right job, thereby optimizing the blend of quality, cost, and latency.

A Decision Tree for Practitioners. To aid in practical decision-making, the findings can be distilled into the following decision tree for deploying a RAG system. These recommendations build on the principal findings and aim to guide practical RAG deployments.

5.4.2. Limitations and Threats to Validity

5.4.2.1. Limitations of the Study

While the findings provide a systematic comparison, their scope is defined by the following limitations.

Corpus and Query Specificity. The conclusions are intrinsically linked to the historical corpus, which is characterized by a majority of short documents. The identified superiority of a no-chunking baseline is unlikely to hold for corpora with longer, multi-topic documents, where intelligent segmentation is critical. Similarly, the 42-question test set, while covering different complexities, is not exhaustive. **Real-world query distributions, particularly those involving complex reasoning chains or creative synthesis tasks not represented in the evaluation set, will likely expose different failure modes** requiring architectural adaptations. A significantly larger and more diverse question sample would be needed to model performance for a production system.

1. What is your primary optimization goal?

- **Maximum Factual Accuracy (Low Hallucination Risk):** → Go to 2.
- **Maximum User-Perceived Relevance & Answer Quality:** → Go to 3.
- **Lowest Latency / Cost is Paramount:** → Go to 4.

2. (Accuracy Focus) What is your budget?

- **API-Cost is acceptable:** Use **OpenAI Embeddings + `dense_semantic` + GPT-3.5/4**.
- **Must be self-hosted:** Use **BGE Embeddings + `dense_semantic` + Mistral 7B**.

3. (Relevance Focus) What is your latency tolerance?

- **Latency is secondary to quality:** Use **OpenAI Embeddings + `hybrid_sentence_rerank` + GPT-3.5/4**. (Highest quality, highest cost/latency)
- **Need a balance:** Use **OpenAI Embeddings + `hybrid_sentence_rerank` + Mistral 7B**.

4. (Efficiency Focus) What is your corpus structure?

- **Mostly short documents:** Use **No-Chunking + a simple Dense Retriever**. (Fastest)
- **Mixed or long documents:** Use **BGE Embeddings + `dense_semantic` + Phi-3/SLM**. (Best low-resource setup)

Figure 5.3.: A practical decision tree for selecting a RAG pipeline based on project goals.

Component and System Dynamics. This study evaluates a specific set of components that represent a snapshot in time. The rapid evolution of LLMs and embedding models means the identified "performance frontier" is subject to **temporal validity constraints**; the optimal pipeline in two years may be entirely different. Furthermore, the analysis does not model **error propagation**. A failure in the retrieval step can lead to a cascading failure where even a state-of-the-art generator like GPT-4 cannot recover, as it is forced to synthesize an answer from irrelevant or incomplete context.

Scope of Deployment Context. The evaluation was conducted in a single-user, English-only environment. This overlooks two critical aspects of production systems. First, **multi-user scenarios** with concurrent requests introduce performance

challenges (e.g., GPU memory contention, caching effects) not measured here. Second, the **language bias** means the findings on optimal chunking and retrieval are not necessarily transferable to multilingual corpora, where tokenization, semantic understanding, and retrieval strategies must account for diverse linguistic structures.

5.4.2.2. Threats to Validity

Beyond the specific limitations, I acknowledge the following broader threats to the validity of my conclusions, following established standards in empirical research.

Internal Validity. The experiments were designed for reproducibility, using a deterministic generation setting (temperature = 0). While this ensures that results for a given pipeline are consistent, it does not reflect real-world applications where a degree of stochasticity (temperature > 0) is often desirable for more varied and natural-sounding responses. This variability could influence user-perceived quality in ways not captured here.

Construct Validity. This study relies on RAGAS as the primary instrument for measuring quality. A crucial question is whether these automated metrics— the operationalization of "quality"—truly measure what end-users desire. "Answer Relevancy," for example, might not fully capture a user's preference for conciseness, tone, or actionability. The mapping from quantitative scores to real-world user satisfaction is therefore an assumed, but not proven, relationship.

External Validity. The results were obtained in a controlled "laboratory" environment. The transition to a production environment introduces countless confounding variables, including network latency, API reliability, variable user expertise, and adversarial inputs. Therefore, the performance benchmarks achieved in this study represent an upper bound that may not be fully attainable in a live, deployed system.

6. Lessons Learned

This chapter distills the most important lessons gleaned from the literature review, the development of the ALiVE system, and the empirical evaluation conducted in this thesis, followed by personal reflections on process and scope.

6.1. Literature

Across a broad landscape spanning CS, NLP, AI in education, and history education, the literature validated the core direction of this work: retrieval-augmented generation (RAG) as a pragmatic response to hallucinations, coupled with design choices that explicitly trade off answer quality dimensions (chunking, hybrid retrieval, reranking). In particular, the thesis' positioning of ALiVE as a source-grounded, pedagogically motivated system with persona-driven immersion bridged well between theory and implementation, aligning a modular RAG pipeline with educational aims such as historical thinking and engagement. The conceptualization of first-person responses (*Eyewitness Mode*) as a lightweight prompt-level mechanism, combined with visible sources and safeguards, offered a grounded path to immersion without architectural overreach. :contentReference[oaicite:0]index=0

Two observations proved especially durable while surveying prior art and designing the study: (i) the "no single best" expectation—different architectures amplify different quality facets; and (ii) the need to evaluate not only absolute performance but also cost, latency, and education-specific constraints (e.g., transparency and verifiability). These themes directly informed the evaluation focus and the later practitioner-facing decision guidance. :contentReference[oaicite:1]index=1

6.2. Development

On the engineering side, simplicity of the core structure was an advantage: a clean separation between front-end, API, and a modular RAG backend made it straightforward to swap chunkers, toggling hybrid vs. dense retrieval, and inserting a cross-encoder reranker. The per-persona, file-based persistence (documents, FAISS stores, persona JSON) and the explicit orchestration of the query lifecycle kept the prototype maintainable while enabling controlled experiments. This modularity also made the *Eyewitness Mode* an additive feature rather than a disruptive rewrite. :contentReference[oaicite:2]index=2

At the same time, velocity in the LLM ecosystem imposed scope pressure. New models and components arrive quickly, but integrating them responsibly requires clear evaluation hooks and stable baselines. The thesis' emphasis on metric-driven iteration and a fixed, reproducible setting (e.g., deterministic decoding) helped mitigate churn, while the discussion of validity threats underscores why lab-grade results need cautious translation to classroom-grade deployment. :contentReference[oaicite:3]index=3

6.3. Personal

By having had the chance to work on such an interesting topic for several months, I encountered a number of areas that were previously new to me. Most notably, this thesis marked my first sustained, hands-on work with *RAGAS*, with data-science workflows more broadly, and with Python as a practical tool for evaluation (from notebooks and structured JSONL datasets to metric computation and result aggregation). I enjoyed the process of learning how to turn vague research questions into reproducible experiments, and how small design choices in code—chunking windows, reranking toggles, or candidate pool sizes—can translate into measurable differences in Faithfulness and Answer Relevancy.

Equally important, the project allowed me to merge two long-standing interests: education and information technology. I had already explored this intersection in my bachelor's thesis; here, I was able to take it further by building a system that deliberately foregrounds sources and explainability while still offering an engaging, persona-driven experience. The feeling of "closing the loop"—from literature to implementation to evaluation—made the technical work personally meaningful.

A less expected but valuable part of the journey came from conversations: with friends and family, who often use AI tools but see them as a black box; and, interestingly, with AI systems themselves while debugging and stress-testing prompts. These discussions sharpened my awareness that beyond raw model quality, transparency ("what is this answer based on?") and guidance ("how should I use this in a learning context?") matter greatly for trust and pedagogy. They also reinforced a simple lesson: even small, well-tuned models can be powerful when paired with careful retrieval and prompt design, but communicating capabilities and limits clearly is just as important as improving the metrics.

Overall, this work refreshed methodological skills from my studies and added practical habits I will carry forward: keep experiments reproducible, measure before optimizing, and design for learners, not just for scores.

7. Conclusion and Future Work

This chapter concludes the thesis by summarizing the work done to design, implement, and evaluate the ALiVE system as a contribution to history education with LLM-powered, retrieval-grounded chat. It also outlines concrete next steps that can be pursued in a master-sized continuation of the project.

7.1. Conclusion

This thesis developed ALiVE, a modular RAG system that combines a non-parametric knowledge base with prompt engineering to deliver source-grounded answers and an optional first-person *Eyewitness Mode*. On the engineering side, the work introduced multi-strategy chunking (recursive, sentence-based, semantic), hybrid retrieval (BM25 + dense), cross-encoder reranking, and an evaluation notebook that turns questions, contexts, and generated answers into reproducible, metric-driven experiments. The front end complements this with transparency features (e.g., visible sources) and export options (e.g., PDF).

The technical evaluation used automatically generated questions and RAGAS metrics to quantify retrieval and answer quality. A central finding is a *Pareto frontier* between *Faithfulness* and *Answer Relevancy*: `dense_semantic` provides the strongest factual grounding, whereas `hybrid_sentence_rerank` optimizes perceived relevance and coverage (CP/CR). Cross-encoder reranking consistently improves retrieval quality (CP/CR) with modest gains in Faithfulness; hybrid retrieval increases recall but can introduce noise when the dense candidate pool is already coherent. Generator choice further stratifies end quality under identical context, underscoring that retrieval is necessary but not sufficient for grounded answers. Taken together, these results justify decision guidance that aligns pipeline choices with product goals (accuracy vs. relevance vs. efficiency).

Frontier, not a winner. The evaluation confirmed a Pareto frontier between *Faithfulness* and *Answer Relevancy*, with `dense_semantic` anchoring factual grounding and `hybrid_sentence_rerank` maximizing relevancy and coverage (CP/CR). This validates the literature-informed expectation that architectural choices trade off quality facets, rather than yielding a single universal optimum. :contentReference[oaicite:4]index=4

Chunking, reranking, hybrid. Semantically coherent segmentation proved a robust dense foundation, especially as chunk size varied; cross-encoder reranking reliably improved retrieval metrics (CP/CR) with modest gains in Faithfulness; and hybrid retrieval increased recall but could introduce noise when the dense candidate pool was already coherent—precisely the kind of *situational* benefit practitioners must weigh. :contentReference[oaicite:5]index=5

Significance is hard-won. Pairwise, per-question tests against the Faithfulness baseline showed that only a few gaps were statistically significant after Holm correction. The broader lesson: improving groundedness meaningfully is difficult; however, certain design missteps (e.g., recursive variants in this setting) can degrade it quickly—arguing for conservative defaults and careful ablations. :contentReference[oaicite:6]index=6

Proxies and internal scores. Internal `factuality_score` signals correlated weakly with external Faithfulness and should be treated as runtime heuristics (e.g., for flagging low-confidence answers), not evaluation surrogates. This reinforces the need for explicit groundedness metrics when selecting or tuning pipelines for educational use. :contentReference[oaicite:7]index=7

Efficiency and the generator. Cost-quality analysis emphasized diminishing returns at the high-latency end, while generator choice materially impacted final answer quality under identical context: larger, more capable models delivered the highest Faithfulness and Relevancy; compact models were efficient yet sensitive to imperfect context. Together, these findings justify decision aids that align pipeline designs with priorities (accuracy vs. relev

7.2. Future Work

The following roadmap prioritizes high-leverage improvements that a master-level continuation can realistically deliver in one to two semesters. I group items by theme and suggest tangible deliverables.

(A) Retrieval and Indexing.

- **Layout-aware preprocessing.** Add PDF parsing that preserves headings, captions, tables and figure text; integrate table extraction and figure caption indexing. *Deliverable:* a preprocessing module with unit tests and ablation showing CP/CR/F gains on table-heavy sources.
- **Adaptive, semantic chunking.** Replace fixed windows with content-aware segmenters (sentence/paragraph boundaries + semantic cohesion) and a fallback to whole-document retrieval for short texts. *Deliverable:* a chunking

policy with knobs for max length, lexical overlap, and "evidence density," plus an evaluation comparing 500/50 vs. 1000/30 vs. adaptive.
- **Candidate diversification.** Add Maximal Marginal Relevance (MMR) or coverage-aware reranking to reduce the "High CP, Zero CR" failure mode. *Deliverable:* retrieval module with MMR and a case study showing improved CR without large AR loss.

(B) Query Understanding and Routing.

- **Lightweight query classifier.** Predict *abstract/conceptual* vs. *specific/multi-hop* vs. *lookup* and route to the best pipeline (e.g., `dense_semantic` vs. `hybrid_sentence_rerank` vs. BM25-only). *Deliverable:* a small DistilBERT-class model and an A/B test showing macro-metric gains over a static pipeline.
- **Query rewriting.** Add HyDE/keyword expansion and temporal normalization (e.g., "in 1776" $\rightarrow$ date filters). *Deliverable:* a pre-retrieval rewrite step with toggles and an ablation on CP/CR.

(C) Grounding, Calibration, and Guardrails.

- **Per-claim citations and abstention.** Enforce inline span-level citations; add a *strictness* knob and abstain when the verifier and CR drop below thresholds. *Deliverable:* a citation-aware generator and a teacher-mode that can force abstention.
- **Verifier-in-the-loop.** Use the internal `factuality_score` as a runtime guard (not as an evaluation proxy): trigger re-retrieve/re-rank or answer refusal. *Deliverable:* a simple policy that reduces zero-faithfulness failures on a held-out set.

(D) Evaluation Enhancements.

- **Gold justifications.** Extend the dataset with short evidence spans (2–3 sentences) for a subset of questions to better track CR and citation accuracy. *Deliverable:* a 100–150 item annotated set and scripts for span-level scoring.
- **Human rubric side-car.** Add a lightweight teacher rubric (clarity, appropriateness, actionability) for 30–50 items to triangulate RAGAS. *Deliverable:* inter-annotator stats and correlation plots vs. automated metrics.

(E) Pedagogical Features.

- **Lesson authoring and export.** Let teachers pin answers, compile sources, and export a lesson handout (PDF/Markdown) with citations and discussion prompts. *Deliverable:* a "Lesson Builder" view and a user walkthrough.
- **Scaffolding tools.** Add hints, step-by-step reasoning views, and timeline/map widgets for dates and places. *Deliverable:* two UI widgets (timeline, map) tied to retrieved entities.

(F) Performance and Deployment.

- **Caching and vector reuse.** Add query/result caching and a vector cache for frequent entities; quantify latency and cost reductions. *Deliverable:* telemetry dashboard and a before/after latency study.
- **Local-first option.** Package the stack in containers for offline demos (BGE + Mistral-class generator). *Deliverable:* a dockerized "teacher laptop" build with instructions.

(G) Safety, Privacy, and Compliance.

- **Source transparency by design.** Always show sources; highlight exact spans. *Deliverable:* UI refactor with span highlighting.
- **Privacy hygiene.** Add role-based controls, configurable logs, and clear data-retention toggles suitable for classroom use. *Deliverable:* settings panel + brief DPIA checklist.

In summary, the immediate path forward is to make ALiVE *more grounded, more adaptive, and more classroom-ready*: tighten evidence coverage, route queries intelligently, and ship teacher-facing workflows. Each step is measurable with the existing evaluation harness and achievable within a master-level scope.

Bibliography

Adlakha, V., BehnamGhader, P., Lu, X. H., Meade, N., & Reddy, S. (2024). Evaluating correctness and faithfulness of instruction-following models for question answering. https://arxiv.org/abs/2307.16877 (cit. on p. 108).

Akgun, S. (2021). Artificial intelligence in education: Addressing ethical challenges in k-12 settings. *AI and Ethics*, *2*, 431–440. https://doi.org/10.1007/s43681-021-00096-7 (cit. on p. 16).

Asai, A., Wu, Z., Wang, Y., Sil, A., & Hajishirzi, H. (2023). Self-rag: Learning to retrieve, generate, and critique through self-reflection. https://arxiv.org/abs/2310.11511 (cit. on p. 116).

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., … Kaplan, J. (2022). Training a helpful and harmless assistant with reinforcement learning from human feedback. https://arxiv.org/abs/2204.05862 (cit. on p. 108).

Baidoo-Anu, D., & Ansah, L. (2023). Education in the era of generative artificial intelligence (ai): Understanding the potential benefits of chatgpt in promoting teaching and learning. *Journal of AI*, *7*. https://doi.org/10.61969/jai.1337500 (cit. on p. 38).

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., & Wang, T. (2016). Ms marco: A human generated machine reading comprehension dataset. (Cit. on p. 30).

Balaguer, A., Benara, V., de Freitas Cunha, R. L., de M. Estevão Filho, R., Hendry, T., Holstein, D., Marsman, J., Mecklenburg, N., Malvar, S., Nunes, L. O., Padilha, R., Sharp, M., Silva, B., Sharma, S., Aski, V., & Chandra, R. (2024). Rag vs fine-tuning: Pipelines, tradeoffs, and a case study on agriculture. https://arxiv.org/abs/2401.08406 (cit. on pp. 112, 113).

Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., & Abdelrazek, M. (2024). Seven failure points when engineering a retrieval augmented generation system. https://arxiv.org/abs/2401.05856 (cit. on p. 107).

Baron, N. S. (2023). *Who wrote this? how ai and the lure of efficiency jeopardize the values of higher education*. Stanford University Press. (Cit. on p. 17).

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? . *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and*

Transparency, 610–623. https://doi.org/10.1145/3442188.3445922 (cit. on p. 15).

Bhat, S. R., Rudat, M., Spiekermann, J., & Flores-Herr, N. (2025). Rethinking chunk size for long-document retrieval: A multi-dataset analysis. https://arxiv.org/abs/2505.21700 (cit. on pp. 98, 102).

BLOOM, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, *13*(6), 4–16. https://doi.org/10.3102/0013189X013006004 (cit. on p. 8).

Bond, M., Buntins, K., Bedenlier, S., Zawacki-Richter, O., & Kerres, M. (2020). Mapping research in student engagement and educational technology in higher education: A systematic evidence map. *International Journal of Educational Technology in Higher Education*, *17*(1), 2. https://doi.org/10.1186/s41239-019-0176-8 (cit. on p. 38).

Bray, B., & McClaskey, K. (2015). *Make learning personal: The what, who, how, where, and why* [Available at: https://www.amazon.com/Make-Learning-Personal-What-Where/dp/1503908623]. CreateSpace Independent Publishing Platform. (Cit. on p. 1).

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners (cit. on p. 6).

Carbonell, J., & Goldstein, J. (1998). The use of mmr, diversity-based reranking for reordering documents and producing summaries. *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 335–336. https://doi.org/10.1145/290941.291025 (cit. on p. 109).

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., Oprea, A., & Raffel, C. (2021). Extracting training data from large language models. https://arxiv.org/abs/2012.07805 (cit. on p. 18).

Chase, C. C., Chin, D. B., Oppezzo, M. A., & Schwartz, D. L. (2009). Teachable agents and the protégé effect: Increasing the effort towards learning. *Journal of Science Education and Technology*, *18*(4), 334–352. https://doi.org/10.1007/s10956-009-9180-4 (cit. on p. 9).

Chen, J., Lin, H., Han, X., & Sun, L. (2023). Benchmarking large language models in retrieval-augmented generation. https://arxiv.org/abs/2309.01431 (cit. on pp. 105, 107).

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating large language models trained on code. https://arxiv.org/abs/2107.03374 (cit. on p. 19).

Chi, M. T. H., & Wylie, R. (2014). The icap framework: Linking cognitive engagement to active learning outcomes. *Educational Psychologist*, *49*(4), 219–243. https://doi.org/10.1080/00461520.2014.965823 (cit. on p. 2).

Chiang, T.-R., Yu, X., Robinson, J., Liu, O., Lee, I., & Yogatama, D. (2024, June). On retrieval augmentation and the limitations of language model training. In K. Duh, H. Gomez, & S. Bethard (Eds.), *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 2: Short papers)* (pp. 229–238). Association for Computational Linguistics. https://doi.org/10.18653/v1/2024.naacl-short.21 (cit. on p. 113).

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2023). Deep reinforcement learning from human preferences. https://arxiv.org/abs/1706.03741 (cit. on p. 13).

Colby, K. M. (1981). Modeling a paranoid mind. *Behavioral and Brain Sciences*, *4*(4), 515–560 (cit. on pp. 5, 37).

Cuconasu, F., Trappolini, G., Siciliano, F., Filice, S., Campagnano, C., Maarek, Y., Tonellotto, N., & Silvestri, F. (2024). The power of noise: Redefining retrieval for rag systems. *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 719–729. https://doi.org/10.1145/3626772.3657834 (cit. on p. 25).

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, *7*, 1–30 (cit. on p. 109).

Dror, R., Baumer, G., Shlomov, S., & Reichart, R. (2018, July). The hitchhiker's guide to testing statistical significance in natural language processing. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1383–1392). Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-1128 (cit. on p. 109).

Eisenstein, E. L. (1980). *The printing press as an agent of change*. Cambridge University Press. (Cit. on p. 10).

Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2025). Ragas: Automated evaluation of retrieval augmented generation. https://arxiv.org/abs/2309.15217 (cit. on pp. 23, 31, 111).

Fabbri, A., Wu, C.-S., Liu, W., & Xiong, C. (2022). QAFactEval: Improved QA-based factual consistency evaluation for summarization. In M. Carpuat, M.-C. de Marneffe, & I. V. Meza Ruiz (Eds.), *Proceedings of the 2022 conference of the north american chapter of the association for computational linguistics: Human language technologies*. (Cit. on p. 111).

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. (2023). Large language models for software engineering: Survey and open problems, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008 (cit. on p. 8).

Formal, T., Piwowarski, B., & Clinchant, S. (2021). Splade: Sparse lexical and expansion model for first stage ranking. https://arxiv.org/abs/2107.05720 (cit. on pp. 104, 109).

Gao, L., Ma, X., Lin, J., & Callan, J. (2022). Precise zero-shot dense retrieval without relevance labels. https://arxiv.org/abs/2212.10496 (cit. on pp. 101, 102).

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024). Retrieval-augmented generation for large language models: A survey. https://arxiv.org/abs/2312.10997 (cit. on pp. 20, 34).

Goot, M., Hafkamp, L., & Dankfort, Z. (2021, February). Customer service chatbots: A qualitative interview study into the communication journey of customers. https://doi.org/10.1007/978-3-030-68288-0_13 (cit. on p. 8).

Green, M., & Brock, T. (2000). The role of transportation in the persuasiveness of public narrative. *Journal of personality and social psychology*, *79*, 701–21. https://doi.org/10.1037/0022-3514.79.5.701 (cit. on p. 18).

Gregorcic, B., Polverini, G., & Sarlah, A. (2024, March). *Chatgpt as a tool for honing teachers' socratic dialogue skills*. https://doi.org/10.48550/arXiv.2401.11987 (cit. on p. 9).

Gudibande, A., Wallace, E., Snell, C., Geng, X., Liu, H., Abbeel, P., Levine, S., & Song, D. (2023). The false promise of imitating proprietary llms. https://arxiv.org/abs/2305.15717 (cit. on p. 114).

Harasim, L. (2000). Shift happens: Online education as a new paradigm in learning. *The Internet and Higher Education*, *3*(1), 41–61. https://doi.org/https://doi.org/10.1016/S1096-7516(00)00032-4 (cit. on p. 10).

Helm, C., Große, C., & öbv. (2024). Einsatz künstlicher intelligenz im schulalltag - eine empirische bestandsaufnahme (cit. on p. 17).

Hofstätter, S., Althammer, S., Schröder, M., Sertkan, M., & Hanbury, A. (2021). Improving efficient neural ranking models with cross-architecture knowledge distillation. https://arxiv.org/abs/2010.02666 (cit. on pp. 103, 112).

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, *6*(2), 65–70 (cit. on p. 109).

Honovich, O., Aharoni, R., Herzig, J., Taitelbaum, H., Kukliansy, D., Cohen, V., Scialom, T., Szpektor, I., Hassidim, A., & Matias, Y. (2022). True: Re-evaluating factual consistency evaluation. https://arxiv.org/abs/2204.04991 (cit. on p. 111).

Huang, W., Hew, K. F., & Gonda, D. E. (2019). Designing and evaluating three chatbot-enhanced instructional approaches for teaching critical thinking skills. *Proceedings of the 27th International Conference on Computers in Education* (cit. on p. 42).

Huang, W., Hew, K. F., & Fryer, L. K. (2022). Chatbots for language learning—are they really useful? a systematic review of chatbot-supported language learning. *Journal of Computer Assisted Learning*, *38*(1), 237–257. https://doi.org/10.1111/jcal.12610 (cit. on p. 9).

Huang, W., Jiang, J., King, R. B., & Fryer, L. K. (2025). Chatbots and student motivation: A scoping review. *International Journal of Educational Technology in Higher Education*, *22*(1), 26. https://doi.org/10.1186/s41239-025-00524-2 (cit. on p. 42).

Humeau, S., Shuster, K., Lachaux, M.-A., & Weston, J. (2020). Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. https://arxiv.org/abs/1905.01969 (cit. on pp. 28, 30).

Jeong, S., Baek, J., Cho, S., Hwang, S. J., & Park, J. C. (2024). Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. https://arxiv.org/abs/2403.14403 (cit. on p. 116).

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, *55*(12), 1–38. https://doi.org/10.1145/3571730 (cit. on p. 13).

Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with gpus. https://arxiv.org/abs/1702.08734 (cit. on p. 21).

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. https://arxiv.org/abs/2004.04906 (cit. on pp. 102, 106).

Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., … Kasneci, G. (2023). Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274. https://doi.org/https://doi.org/10.1016/j.lindif.2023.102274 (cit. on pp. 1, 9).

Kerby, D. S. (2014). The simple difference formula: An approach to teaching nonparametric correlation1. *Comprehensive Psychology*, *3*, 11.IT.3.1. https://doi.org/10.2466/11.IT.3.1 (cit. on p. 109).

Kooli, C. (2023). Chatbots in education and research: A critical examination of ethical implications and solutions. *Sustainability*, *15*(7). https://doi.org/10.3390/su15075614 (cit. on p. 42).

Körber, A. (2011, December). German history didactics: From historical consciousness to historical competencies – and beyond? https://doi.org/10.14361/transcript.9783839413258.145 (cit. on p. 19).

Kudo, T., & Richardson, J. (2018, November). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In E. Blanco & W. Lu (Eds.), *Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations* (pp. 66–71). Association for Computational Linguistics. https://doi.org/10.18653/v1/D18-2012 (cit. on pp. 11, 34).

Kuhail, M. A., Alturki, N., Alramlawi, S., & Alhejori, K. (2022). Interacting with educational chatbots: A systematic review. *Education and Information Technologies*, *28*, 1–46. https://doi.org/10.1007/s10639-022-11177-3 (cit. on p. 1).

Laban, P., Schnabel, T., Bennett, P. N., & Hearst, M. A. (2022). SummaC: Revisiting NLI-based models for inconsistency detection in summarization (B. Roark & A. Nenkova, Eds.). *Transactions of the Association for Computational Linguistics*, *10*, 163–177. https://doi.org/10.1162/tacl_a_00453 (cit. on pp. 106, 107).

Lee, D., Hwang, S.-w., Lee, K., Choi, S., & Park, S. (2023, July). On complementarity objectives for hybrid retrieval. In A. Rogers, J. Boyd-Graber, & N. Okazaki (Eds.), *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 13357–13368). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-long.746 (cit. on pp. 104, 109).

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks. https://arxiv.org/abs/2005.11401 (cit. on p. 19).

Li, Y., Bubeck, S., Eldan, R., Giorno, A. D., Gunasekar, S., & Lee, Y. T. (2023). Textbooks are all you need ii: Phi-1.5 technical report. https://arxiv.org/abs/2309.05463 (cit. on p. 114).

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). Lost in the middle: How language models use long contexts. https://arxiv.org/abs/2307.03172 (cit. on pp. 26, 108).

Liu, Y., Hu, X., Zhang, S., Chen, J., Wu, F., & Wu, F. (2024). Fine-grained guidance for retrievers: Leveraging llms' feedback in retrieval-augmented generation. https://arxiv.org/abs/2411.03957 (cit. on p. 30).

Ma, X., Zhang, X., Pradeep, R., & Lin, J. (2023). Zero-shot listwise document reranking with a large language model. https://arxiv.org/abs/2305.02156 (cit. on pp. 30, 103).

Maekawa, S., Iso, H., Gurajada, S., & Bhutani, N. (2024). Retrieval helps or hurts? a deeper dive into the efficacy of retrieval augmentation to language models. https://arxiv.org/abs/2402.13492 (cit. on p. 114).

Mallen, A., Asai, A., Zhong, V., Das, R., Khashabi, D., & Hajishirzi, H. (2023, July). When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In A. Rogers, J. Boyd-Graber, & N. Okazaki (Eds.), *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 9802–9822). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-long.546 (cit. on p. 106).

Manakul, P., Liusie, A., & Gales, M. J. F. (2023). Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. https://arxiv.org/abs/2303.08896 (cit. on pp. 109, 111).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press. http://www-csli.stanford.edu/~hinrich/information-retrieval-book.html (cit. on p. 101).

Min, S., Krishna, K., Lyu, X., Lewis, M., Yih, W.-t., Koh, P. W., Iyyer, M., Zettlemoyer, L., & Hajishirzi, H. (2023). Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. https://arxiv.org/abs/2305.14251 (cit. on p. 108).

Molenaar, I. (2022). Towards hybrid human-ai learning technologies. *European Journal of Education*, *57*. https://doi.org/10.1111/ejed.12527 (cit. on p. 38).

Montenegro-Rueda, M., Fernández Cerero, J., Fernández Batanero, J., & Meneses, E. (2023). Impact of the implementation of chatgpt in education: A systematic review. *Computers*, *12*, 153. https://doi.org/10.3390/computers12080153 (cit. on p. 1).

Nogueira, R., & Cho, K. (2020). Passage re-ranking with bert. https://arxiv.org/abs/1901.04085 (cit. on pp. 101, 103).

Okonkwo, C. W., & Ade-Ibijola, A. (2021). Chatbots applications in education: A systematic review. *Computers and Education: Artificial Intelligence*, *2*, 100033. https://doi.org/https://doi.org/10.1016/j.caeai.2021.100033 (cit. on p. 8).

Ouyang, F., & Jiao, P. (2021). Artificial intelligence in education: The three paradigms. *Computers and Education: Artificial Intelligence*, *2*, 100020. https://doi.org/10.1016/j.caeai.2021.100020 (cit. on p. 1).

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022, March). *Training language models to follow instructions with human feedback*. https://doi.org/10.48550/arXiv.2203.02155 (cit. on pp. 7, 12).

Ovadia, O., Brief, M., Mishaeli, M., & Elisha, O. (2024). Fine-tuning or retrieval? comparing knowledge injection in llms. https://arxiv.org/abs/2312.05934 (cit. on p. 28).

Pan, Y., Pan, L., Chen, W., Nakov, P., Kan, M.-Y., & Wang, W. Y. (2023). On the risk of misinformation pollution with large language models. https://arxiv.org/abs/2305.13661 (cit. on p. 42).

Pane, J., Steiner, E., Baird, M., & Hamilton, L. (2015, October). *Continued progress: Promising evidence on personalized learning*. https://doi.org/10.7249/RR1365 (cit. on p. 1).

Piaget, J. (1971). *Biology and knowledge: An essay on the relations between organic regulations and cognitive processes*. University of Chicago Press. (Cit. on p. 10).

Pinter, K., Tusha, J., Spiesberger, P., Hölbling, D., & Grechenig, T. (2025, April). Comparison of traditional learning methods versus chatbots in history lessons in austria. https://doi.org/10.1007/978-3-031-85561-0_4 (cit. on p. 38).

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, *63*, 1872–1897. https://doi.org/10.1007/s11431-020-1647-3 (cit. on p. 8).

Quiroga Perez, J., Daradoumis, T., & Puig, J. (2020). Rediscovering the use of chatbots in education: A systematic literature review. *Computer Applications in Engineering Education*, *28*. https://doi.org/10.1002/cae.22326 (cit. on p. 1).

Radford, A., & Narasimhan, K. (2018). Improving language understanding by generative pre-training. https://api.semanticscholar.org/CorpusID:49313245 (cit. on p. 12).

Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., & Shoham, Y. (2023). In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, *11*, 1316–1331. https://doi.org/10.1162/tacl_a_00605 (cit. on p. 20).

Rashkin, H., Nikolaev, V., Lamm, M., Aroyo, L., Collins, M., Das, D., Petrov, S., Tomar, G. S., Turc, I., & Reitter, D. (2022). Measuring attribution in natural language generation models. https://arxiv.org/abs/2112.12870 (cit. on pp. 106, 111).

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. https://arxiv.org/abs/1908.10084 (cit. on p. 21).

Reynolds, L., & McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. https://arxiv.org/abs/2102.07350 (cit. on p. 22).

Sabitzer, B., Hörmann, C., & Kuka, L. (2024). Künstliche intelligenz (ki) in der bildung - ein kinderspiel? chancen, herausforderungen und anwendungsbeispiele für die praxis. *Medienimpulse*, *62*(3), 26 Seiten. https://doi.org/10.21243/mi-03-24-22 (cit. on p. 17).

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. https://arxiv.org/abs/1910.01108 (cit. on p. 116).

Schick, T., & Schütze, H. (2021, June). It's not just size that matters: Small language models are also few-shot learners. In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, & Y. Zhou (Eds.), *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 2339–2352). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.naacl-main.185 (cit. on p. 114).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. https://arxiv.org/abs/1707.06347 (cit. on p. 13).

Shi, F., Chen, X., Misra, K., Scales, N., Dohan, D., Chi, E., Schärli, N., & Zhou, D. (2023). Large language models can be easily distracted by irrelevant context. https://arxiv.org/abs/2302.00093 (cit. on p. 105).

Shi, W., Min, S., Yasunaga, M., Seo, M., James, R., Lewis, M., Zettlemoyer, L., & Yih, W.-t. (2023). Replug: Retrieval-augmented black-box language models. https://arxiv.org/abs/2301.12652 (cit. on p. 40).

Strubell, E., Ganesh, A., & McCallum, A. (2019, July). Energy and policy considerations for deep learning in NLP. In A. Korhonen, D. Traum, & L. Màrquez (Eds.), *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 3645–3650). Association for Computational Linguistics. https://doi.org/10.18653/v1/P19-1355 (cit. on p. 18).

Sun, Y., Sheng, D., Zhou, Z., & Wu, Y. (2024). Ai hallucination: Towards a comprehensive classification of distorted information in artificial intelligence-generated content. *Humanities and Social Sciences Communications*, *11*(1), 1278. https://doi.org/10.1057/s41599-024-03811-x (cit. on p. 14).

Tang, Y., & Yang, Y. (2024). Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. https://arxiv.org/abs/2401.15391 (cit. on p. 106).

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., & Gurevych, I. (2021). Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. https://arxiv.org/abs/2104.08663 (cit. on pp. 28, 30).

Thirunavukarasu, A., Ting, D., Elangovan, K., Gutierrez Sinisterra, L., Tan, T., & Ting, D. (2023). Large language models in medicine. *Nature Medicine*, *29*. https://doi.org/10.1038/s41591-023-02448-8 (cit. on p. 8).

Thorp, R., & Persson, A. (2020). On historical thinking and the history educational challenge. *Educational Philosophy and Theory*, *52*, 1–11. https://doi.org/10.1080/00131857.2020.1712550 (cit. on p. 2).

Traum, D., Jones, A., Hays, K., Maio, H., Alexander, O., Artstein, R., Debevec, P., Gainer, A., Georgila, K., Haase, K., Jungblut, K., Leuski, A., Smith, S., & Swartout, W. (2015). New dimensions in testimony: Digitally preserving a holocaust survivor's interactive storytelling. *9445*, 269–281. https://doi.org/10.1007/978-3-319-27036-4_26 (cit. on p. 42).

van Doorn, J., Odijk, D., Roijers, D., & Rijke, M. (2016). Multi-objective optimization for information retrieval (cit. on p. 101).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010 (cit. on pp. 6, 11).

Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes* (M. Cole, V. John-Steiner, S. Scribner, & E. Souberman, Eds.). Harvard University Press. (Cit. on pp. 10, 18).

Wallace, R. (2009, January). The anatomy of a.l.i.c.e. https://doi.org/10.1007/978-1-4020-6710-5_13 (cit. on p. 5).

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, *18*(6). https://doi.org/10.1007/s11704-024-40231-1 (cit. on p. 13).

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., & Le, Q. V. (2022). Finetuned language models are zero-shot learners. https://arxiv.org/abs/2109.01652 (cit. on p. 12).

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models. (Cit. on p. 10).

Weidinger, L., Uesato, J., Rauh, M., Griffin, C., Huang, P.-S., Mellor, J., Glaese, A., Cheng, M., Balle, B., Kasirzadeh, A., Biles, C., Brown, S., Kenton, Z., Hawkins, W., Stepleton, T., Birhane, A., Hendricks, L. A., Rimell, L., Isaac, W., … Gabriel, I. (2022). Taxonomy of risks posed by language models, 214–229. https://doi.org/10.1145/3531146.3533088 (cit. on pp. 15, 19).

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, *9*(1), 36–45. https://doi.org/10.1145/365153.365168 (cit. on pp. 5, 37).

Williamson, B., & Eynon, R. (2020). Historical threads, missing links, and future directions in ai in education. *Learning, Media and Technology*, *45*(3), 223–235. https://doi.org/10.1080/17439884.2020.1798995 (cit. on p. 9).

Winkler, R., & Söllner, M. (2018). Unleashing the potential of chatbots in education: A state-of-the-art analysis. *Academy of Management Proceedings*, *2018*, 15903. https://doi.org/10.5465/AMBPP.2018.15903abstract (cit. on p. 1).

Wollny, S., Schneider, J., Di Mitri, D., Weidlich, J., Rittberger, M., & Drachsler, H. (2021). Are we there yet? - a systematic literature review on chatbots in education. *Frontiers in Artificial Intelligence*, *4*, 654924. https://doi.org/10.3389/frai.2021.654924 (cit. on pp. 38, 43).

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., & Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. https://arxiv.org/abs/1809.09600 (cit. on p. 106).

Zawacki-Richter, O., Marín, V. I., Bond, M., & Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education – where are the educators? *International Journal of Educational Technology in Higher Education*, *16*(1), 39. https://doi.org/10.1186/s41239-019-0171-0 (cit. on pp. 1, 9).

Zhao, W., Zhou, K., Junyi, L., Tianyi, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., & Wen, J.-R. (2023). A survey of large language models. https://doi.org/10.48550/arXiv.2303.18223 (cit. on p. 7).

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., & Chi, E. (2023). Least-to-most prompting enables complex reasoning in large language models. https://arxiv.org/abs/2205.10625 (cit. on p. 22).

Zhuang, Z., Zhang, Z., Cheng, S., Yang, F., Liu, J., Huang, S., Lin, Q., Rajmohan, S., Zhang, D., & Zhang, Q. (2024). Efficientrag: Efficient retriever for multi-hop question answering. https://arxiv.org/abs/2408.04259 (cit. on p. 106).

Appendix

Appendix A.

Statistical significance tests

```python
from langchain_community.cross_encoders import HuggingFaceCrossEncoder
"""
Significance vs. dense_semantic (Faithfulness)
- Loads *_final_results.csv files from `data_dir`
- Pairs by exact question string
- Wilcoxon signed-rank test (two-sided), Holm-adjustment
- Rank-biserial correlation as effect size
"""
from pathlib import Path
import pandas as pd
import numpy as np
from scipy import stats

def wilcoxon_table_faithfulness(data_dir: str, baseline='dense_semantic'):
    data_dir = Path(data_dir)
    files = sorted(data_dir.glob("*_final_results.csv"))
    def load(m):
        df = pd.read_csv(data_dir / f"{m}_final_results.csv")
        return df[['question','faithfulness']].rename(columns={'faithfulness': m})
    # load baseline
    base = load(baseline).rename(columns={baseline: 'base'})
    rows = []
    methods = [f.stem.replace("_final_results","") for f in files if f.stem.replace("_final_results","") != baseline]
    for m in methods:
        dfm = load(m)
        merged = base.merge(dfm, on='question', how='inner').dropna()
        diff = merged[m].to_numpy() - merged['base'].to_numpy()
        n = len(diff)
        # Wilcoxon signed-rank test
        wstat, p = stats.wilcoxon(diff, zero_method='zsplit', correction=False, alternative='two-sided', mode='auto')
        # rank-biserial correlation
        nz = diff[diff != 0]
        if len(nz) > 0:
            ranks = stats.rankdata(np.abs(nz))
```

```
            Wpos = np.sum(ranks[nz > 0]); Wneg = np.sum(ranks[nz < 0])
            r_rb = (Wpos - Wneg) / (len(nz) * (len(nz)+1) / 2.0)
        else:
            r_rb = 0.0
        rows.append(dict(method=m,
                         n=n,
                         delta_mean=np.mean(diff),
                         delta_pct=np.mean(diff)/merged['base'].mean()*100,
                         p_value=p,
                         r_rb=r_rb))
    out = pd.DataFrame(rows).sort_values('delta_mean', ascending=False).reset_index(drop=True)
    # Holm adjustment
    mtests = len(out)
    order = np.argsort(out['p_value'].values)
    p_adj = np.empty(mtests)
    for rank, idx in enumerate(order, start=1):
        p_adj[idx] = min(out.loc[idx,'p_value'] * (mtests - rank + 1), 1.0)
    out['p_holm'] = p_adj
    # pretty columns
    def stars(p): return '***' if p<0.001 else '**' if p<0.01 else '*' if p<0.05 else ''
    out['Delta F']   = out['delta_mean'].map(lambda x: f"{x:+.3f}")
    out['Delta F %'] = out['delta_pct'].map(lambda x: f"{x:+.1f}%")
    out['p (Holm)']  = out['p_holm'].map(lambda x: f"{x:.3f}")
    out['Sig.']      = out['p_holm'].map(stars)
    out['r_rb']      = out['r_rb'].map(lambda x: f"{x:+.3f}")
    return out[['method','n','Delta F','Delta F %','p (Holm)','Sig.','r_rb']]

# Example:
# table = wilcoxon_table_faithfulness("./results_final")
# print(table.to_string(index=False))
```

Listing A.1: Significance vs. dense$_s$*emantic*

Appendix B.

Factual Score: Computing the correlations

```
from scipy.stats import pearsonr, spearmanr, linregress

DATA_DIR = "./results_topipelines"
FILES = [
    "dense_semantic_final_results.csv",
    "hybrid_semantic_rerank_final_results.csv",
    "hybrid_sentence_rerank_final_results.csv",
]

def corr_table(files):
    rows = []
    for f in files:
        df = pd.read_csv(os.path.join(DATA_DIR, f))
        x = pd.to_numeric(df["factuality_score"], errors="coerce")
        y = pd.to_numeric(df["faithfulness"], errors="coerce")
        m = x.notna() & y.notna()
        x, y = x[m], y[m]
        pr, pr_p = pearsonr(x, y)
        sr, sr_p = spearmanr(x, y)
        lr = linregress(x, y)
        rows.append({
            "pipeline": f.replace("_final_results.csv",""),
            "n": len(x),
            "pearson_r": pr, "pearson_p": pr_p,
            "spearman_rho": sr, "spearman_p": sr_p,
            "slope": lr.slope, "r2": lr.rvalue**2
        })
    return pd.DataFrame(rows)

if __name__ == "__main__":
    out = corr_table(FILES)
    # Round for display
    print(out.assign(
        pearson_r=lambda d: d.pearson_r.round(3),
        pearson_p=lambda d: d.pearson_p.round(3),
        spearman_rho=lambda d: d.spearman_rho.round(3),
        spearman_p=lambda d: d.spearman_p.round(3),
```

```
38          slope=lambda d: d.slope.round(3),
39          r2=lambda d: d.r2.round(3),
40      ))
```

Listing B.1: Factual Score: Computing the correlations