UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc “INFORMATICS”

ΠΜΣ «ΠΛΗΡΟΦΟΡΙΚΗΣ»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Educational Software with Large Language Models** Εκπαιδευτικό Λογισμικό με Μεγάλα Γλωσσικά Μοντέλα
Student’s name-surname: Ονοματεπώνυμο φοιτητή:	**Tsirini Varvara** Τσιρίνη Βαρβάρα
Father’s name: Πατρώνυμο:	**George** Γεώργιος
Student’s ID No: Αριθμός Μητρώου:	ΜΠΠΛ20083
Supervisor: Επιβλέπουσα:	**Maria Virvou, Professor** Μαρία Βίρβου, Καθηγήτρια

December 2025 / Δεκέμβριος 2025

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Alepis Efthimios

Αλέπης Ευθύμιος

Professor

Virvou Maria

Βίρβου Μαρία

Professor

Sakkopoulos Evangelos

Σακκόπουλος Ευάγγελος

Associate Professor

ABSTRACT

Rapid technological and social changes, coupled with increasing time pressures, have defined a new reality for modern societies and extend to heavily affect the educational process. For example, educators often struggle with time limitations that restrict their capacity to redesign the teaching material in a more engaging, creative, and responsive way that better addresses the individuals' learning requirements. At the same time, rapid technological advancements such as the advent of Generative AI, have become deeply embedded in everyday life and have fundamentally transformed learners' cognitive patterns and educational preferences (e.g. younger learners, who have typically grown up in digital environments, naturally prefer learning experiences that incorporate technology that allows them to interact with the learning content instead of passively receiving information).

These evolving patterns of both educators and learners impose the need for the development of educational tools that benefit from advanced technologies and enable adaptability to the diverse learning preferences while efficiently supporting teachers in delivering personalized content. Towards this direction, in the present thesis we explore how Generative and (agentic) Artificial Intelligence can empower the next generation of education software design, bridging this gap and ensuring that the teaching and learning processes remain effective and relevant in the face of ongoing societal and technological shifts.

In particular, we developed a series of AI-based tools that dynamically adjust the learning content into different formats, meeting each user's unique learning preferences. These tools employ the use of Large Language Models (LLM) and Retrieval Augmented Generation (RAG) in the context of AI agents and AI flows, that enable the retrieval of information from the given education curriculum and the provision of grounded responses in multimodal forms. For example, we introduce agentic designs that:

- enable the creation of interactive quizzes and quests, gamifying the educational content while empowering knowledge learning and assessment,
- allow for the effective development of learning plans and summaries (with visual and audio support) for addressing the time pressure bottlenecks as well as for covering for the diverse learners' cognitive styles.
- empower the interactive Q&A sessions, fueled by agentic tools that can handle multimedia input and leverage planning and reasoning in order to assist the learner address inquiries on the learning subjects of their preferences, in a grounded and reliable way

Combined with a prototype implementation of such AI agents, we also provide details on the architectural backbone and functional interplay of these tools and conclude this study by sharing ideas for prospect future further research.

Keywords: Next-Generation Educational Software, AI in Education, Cognitive Learning, Large Language Models, Generative Artificial Intelligence, Retrieval Augmented Generation, AI agents, AI flows

ΠΕΡΙΛΗΨΗ

Οι ραγδαίες τεχνολογικές και κοινωνικές εξελίξεις, σε συνδυασμό με την ολοένα αυξανόμενη έλλειψη χρόνου, έχουν διαμορφώσει μια νέα πραγματικότητα στις σύγχρονες κοινωνίες και κατ' επέκταση επηρεάζουν σημαντικά και τις εκπαιδευτικές διαδικασίες. Ενδεικτικά, αν και όλο και περισσότεροι εκπαιδευτικοί αναγνωρίζουν την ανάγκη για αναδιαμόρφωση του διδακτικού υλικού με τρόπο πιο ελκυστικό και δημιουργικό, που θα ανταποκρίνεται καλύτερα στις ατομικές μαθησιακές ανάγκες, η διαρκώς αυξανόμενη πίεση χρόνου το καθιστά δύσκολο έργο προς επίτευξη. Παράλληλα, η έλευση νεοφυών τεχνολογιών, όπως επί παραδείγματι η εμφάνιση της παραγωγικής τεχνητής νοημοσύνης (Generative AI) σε ποικίλες εκφάνσεις της στην καθημερινότητα των σπουδαστών, έχει ταυτόχρονες επιρροές στις γνωσιακές τους συνήθειες και μαθησιακές τους προτιμήσεις. Για παράδειγμα, σπουδαστές νεότερης ηλικίας, που έχουν μεγαλώσει σε περιβάλλοντα χαρακτηριζόμενα από ευρεία έκθεση σε σύγχρονες τεχνολογικές δομές και ψηφιακά ερεθίσματα, ρέπουν προς γνωσιακές συμπεριφορές που χαρακτηρίζονται από έντονη παρουσία της τεχνολογίας και που τους επιτρέπει να αλληλοεπιδρούν με το εκπαιδευτικό υλικό, αντί να λειτουργούν ως παθητικοί καταναλωτές της πληροφορίας.

Οι εξελισσόμενες αυτές δυναμικές στην εκπαίδευση, τόσο από πλευράς εκπαιδευτικών όσο και σπουδαστών, καθιστούν αναγκαία την εκπόνηση ευμετάβλητων εκπαιδευτικών εργαλείων που επιτρέπουν την ανάπτυξη εξατομικευμένου διδακτικού περιεχομένου και ταυτόχρονα αξιοποιούν τις καινοτόμες τεχνολογίες που χρήζουν εφαρμογής στο χώρο της εκπαίδευσης. Αναγνωρίζοντας την επιτακτική αυτή ανάγκη, στα πλαίσια της παρούσας διατριβής εξετάζονται οι ποικίλοι τρόποι με τους οποίους η παραγωγική τεχνητή νοημοσύνη, συνδυαστικά με τη χρήση ευφυών πρακτόρων (AI agents), μπορούν να αποτελέσουν λύση στο εν λόγω ζήτημα, διασφαλίζοντας ότι οι εκπαιδευτικές διαδικασίες μπορούν να παραμείνουν αποτελεσματικές και επικαιροποιημένες, παρά τις συνεχιζόμενες κοινωνικές και τεχνολογικές μεταβολές.

Ειδικότερα, στην παρούσα διατριβή, εισηγούνται και παρουσιάζονται μια σειρά από εργαλεία βασισμένα στην τεχνητή νοημοσύνη, που επιτρέπουν τη δυναμική αναπροσαρμογή του μαθησιακού υλικού σε διαφορετικές μορφές (π.χ. εικόνα, ήχο) προκειμένου να καλύψουν καλύτερα τις ποικίλες μαθησιακές προτιμήσεις των σύγχρονων σπουδαστών. Τα εργαλεία υλοποιούν ευφυής πράκτορες τεχνητής νοημοσύνης που βασίζονται στο δραστικό συνδυασμό μεγάλων γλωσσικών μοντέλων (Large Language Models) με ταυτόχρονη χρήση παραγωγικής σύνθεσης επαυξημένης με μηχανισμούς ανάκτησης πληροφορίας (Retrieval Augmented Generation – RAG) και επιτρέπουν την αξιοποίηση του δοθέντος εκπαιδευτικού υλικού ενώ εξασφαλίζουν την παροχή τεκμηριωμένων και αξιόπιστων απαντήσεων σε πολυτροπικές μορφές. Πιο συγκεκριμένα, παρουσιάζονται αρχιτεκτονικές που:

- αξιοποιούν το εκπαιδευτικό υλικό δια μέσω της δημιουργίας διαδραστικών κουίζ και μηχανισμών παιχνιδοποίησης (gamification) του εκπαιδευτικού περιεχομένου, ενισχύοντας καθ' αυτό τον τρόπο την απόκτηση γνώσης με ευχάριστο και εποικοδομητικό τρόπο
- διευκολύνουν την αποτελεσματική ανάπτυξη μαθησιακών πλάνων και τη σύνταξη συνεκτικών περιλήψεων (με ταυτόχρονη οπτικοακουστική υποστήριξη) στοχεύοντας στην αντιμετώπιση των χρονικών περιορισμών και την κάλυψη των διαφορετικών γνωστικών προτιμήσεων των διδασκομένων
- επιτρέπουν την άμεση αλληλεπίδραση των σπουδαστών με το διδακτικό περιεχόμενο υπό τη μορφή ψηφιακού βοηθού (Q&A agent) που μπορεί να δώσει απαντήσεις σε μαθησιακά ερωτήματα του σπουδαστή, με τεκμηριωμένο και αξιόπιστο τρόπο

Τέλος, ταυτόχρονα με την λεπτομερή ανάλυση των απαιτήσεων και τεχνικών υλοποίησης των προαναφερθέντων πρακτόρων τεχνητής νοημοσύνης, παρατίθενται και αναφορές σχετικά με τις απαιτούμενες αρχιτεκτονικές υποδομές και τις αλληλεπιδράσεις των πρακτόρων αυτών, ενώ εν κατακλείδι παρουσιάζονται ιδέες για μελλοντική περαιτέρω έρευνα και ανάπτυξη.

Λέξεις-κλειδιά: Τεχνολογίες ανάπτυξης εκπαιδευτικού λογισμικού, Τεχνητή νοημοσύνη στην εκπαίδευση, Γλωσσικά μοντέλα, Παραγωγική τεχνητή νοημοσύνη, Επαυξημένη ανάκτηση πληροφοριών, Πράκτορες τεχνητής νοημοσύνης

Table of Contents

Table of Figures

Table of Abbreviations

Abbreviation	Full Description
3D	Three-Dimensional
AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
CAT	Computer Adaptive Testing
CLT	Cognitive Load Theory
EdTech	Educational Technology
ERD	Entity Relationship Diagram
GDPR	General Data Protection Regulation
Gen Z	Generation Z
GenAI	Generative Artificial Intelligence
GPU	Graphics Processing Unit
HNSW	Hierarchical Navigable Small World
IBM	International Business Machines
ITS	Intelligent Tutoring Systems
JSON	JavaScript Object Notation
LLM	Large Language Model
MOOC	Massive Open Online Course
MS MARCO	Microsoft Machine Reading Comprehension
OECD	Organization for Economic Co-operation and Development
PDF	Portable Document Format
RAG	Retrieval-Augmented Generation
RPG	Role-Playing Game
TTS	Text-to-Speech
UDL	Universal Design for Learning
UML	Unified Modeling Language
VARK	Visual, Aural, Read/Write, Kinesthetic
VR	Virtual Reality
KG	Knowledge Graphs
NER	Named Entity Recognition

1. Introduction

1.1. Background of the Study

Educational research over the past several decades has consistently highlighted the importance of optimizing learning processes, with Sweller's Cognitive Load Theory (CLT) emerging as a foundational framework for understanding how students process information (Sweller, 1994). CLT demonstrates that human working memory has inherent limitations when handling complex information, necessitating instructional approaches that prevent cognitive overload. This understanding presents contemporary educators with a significant challenge: how can we manage information delivery effectively in an era where technological advancement accelerates rapidly and the spread of digital information occurs nearly in real time?

Mayer and Moreno's research provides valuable insights into this challenge by advocating for simultaneous presentation of content through both auditory and visual formats (Mayer & Moreno, 2003). This approach allows learners to distribute cognitive processing across multiple channels, thereby reducing the burden on any single processing system. Mayer and Moreno's subsequent research has further validated this approach, demonstrating that learning environments incorporating text, images, and audio produce superior comprehension and retention outcomes compared to traditional single-mode instruction (Moreno & Mayer, 1999).

Such inclusive education practices share fundamental alignment with CLT principles, as they advocate for multi-format content presentation to ensure accessibility, while at the same time optimizing learning outcomes. In parallel, recent studies have demonstrated these methods are also found to be beneficial for students with various disabilities (e.g. visual, auditory, or reading impairments), who require alternative content formats to access educational materials effectively. The Universal Design for Learning (UDL) framework addresses this need by emphasizing multiple means of representation, engagement, and expression to accommodate individual learner differences (Ralabate, 2011), whereas recent studies such as (Capp, 2017) indicate that the implementation of these principles has consistently resulted in improved academic performance and engagement among students with disabilities.

Contemporary educational challenges are further complicated by generational shifts in learning preferences and information processing patterns. Generation Z (Gen Z) students demonstrate preferences for visual and interactive media over traditional text-based materials (Seemiller & Grace, 2017). Having grown up in environments with strong presence of smartphones and social media platforms, these students expect educational content to match the engagement levels and format diversity of their daily digital interactions.

This requirement is even more pronounced in Generation Alpha students (i.e. born after 2010), who typically exhibit higher inclination to interactive and multimedia learning experiences. Studies indicate that these students demonstrate significantly higher engagement rates when content is presented through gamified, brief multimedia formats that mirror their consumption patterns on platforms like TikTok and YouTube, since their cognitive preferences have been adapted to rapid content switching and multi-modal information integration, rendering traditional, lecture-based learning methods increasingly ineffective (2022).

When educational approaches align with learners' preferences, studies indicate that students achieve better exam results compared to traditional teaching methods (Freeman et al., 2014). Institutions failing to adapt to these generational expectations report declining satisfaction and increased disengagement, particularly in higher education where students exercise greater choice in their educational experiences.

Despite the imperative need to infuse such multimedia educational approaches in the learning process, it requires educators to create multiple content versions, a process that significantly increases preparation time (Suprayogi et al., 2017). However, educators are often challenged to balance between pedagogical ideals and resource constraints, rendering the adaptation of comprehensive content extremely challenging to accomplish. Kopcha's research documented that teachers perceive multiple barriers to technology integration, with time constraints being among the most significant obstacles (Kopcha, 2012). The 2019 OECD Teaching and Learning International Survey documented that teachers spend an average of 6.5 hours weekly on lesson planning and material preparation, with many reporting insufficient time for content adaptation and differentiation (OECD, 2019). Such time constraints become particularly problematic when addressing diverse student needs in contemporary classrooms, especially when implementing evidence-based approaches like UDL.

Another blocker in the adoption of emerging technologies in the educational process, also often lies in how schools and universities allocate their budgets, with professional development programs often being the first “victims” when financial restrictions arise. Such budget decisions, end up dictating what kind of education students receive. For example, in cases of restricted budgeting, teachers who may want to innovate, often find themselves without the proper resources or training to achieve their goals and subsequently revert to rely almost exclusively on lecture-based classes (Bingimlas, 2009).

A third constraint lies in the absence of specialized support staff which in turn requires educators to assume roles as content creators, instructional designers, and technology specialists—responsibilities that typically demand distinct skill sets and dedicated time allocation. This expectation extends educators beyond their primary expertise areas, potentially compromising both teaching quality and content development effectiveness. Additionally, such misallocation of skilled personnel prevents institutions from pursuing meaningful digital transformation initiatives. Expertise that could drive innovative learning solutions becomes redirected toward basic administrative functions, creating institutional bottlenecks that impede modernization efforts and educational delivery improvements (Ertmer et al., 2012).

These individual and institutional challenges create systemic barriers to educational modernization that affect entire educational systems and hinder the adoption of improvements that could enhance educational outcomes and operational effectiveness.

Research in artificial intelligence and adaptive learning systems suggests promising directions for addressing these challenges. Studies have demonstrated that fuzzy logic and adaptive learning systems can create personalized educational environments (Chrysafiadi & Virvou, 2012), while educational games and mobile learning platforms effectively engage diverse student populations (Virvou et al., 2005; Virvou & Alepis, 2005). Student modeling approaches provide frameworks for understanding individual learning needs and preferences (Chrysafiadi & Virvou, 2013), enabling more targeted content delivery. Recent advances in artificial intelligence, particularly in automated content generation and personalized tutoring systems (Chen et al., 2020; Kasneci et al., 2023), offer new possibilities for supporting educators in creating multimodal learning experiences. Understanding how existing platforms and research projects employ these AI capabilities provides important context for exploring where academic research might contribute additional insights.

1.2 Related Work

Recent years have seen substantial growth in AI-powered educational tools responding to the challenges outlined above. Both commercial platforms and research projects now address cognitive load management, accessibility, and student engagement through various technical approaches. This section reviews representative examples to understand current strategies for content transformation and personalization.

1.2.1 Proprietary Educational AI Platforms

Khanmigo is an AI tutor from Khan Academy that provides personalized guidance through conversational interaction, helping students work through problems by asking questions rather than giving direct answers. The system focuses primarily on conversational tutoring within Khan Academy's existing content ecosystem (n.d.).

Duolingo employs gamification and adaptive AI to teach languages through bite-sized lessons that incorporate audio, visual, and text feedback. The platform has become one of the most widely used language-learning applications globally (2023).

Quizlet enables users to create digital flashcards and leverages AI to generate personalized study plans through features like Magic Notes and Q-Chat. The platform has established itself as a popular study tool among students across various educational levels (Quizlet Inc., n.d.).

Nearpod allows teachers to transform static lessons such as PowerPoint slides into interactive experiences by embedding quizzes, polls, videos, and virtual reality content. This approach helps educators create more engaging classroom presentations (n.d.).

Genially provides a no-code authoring tool that allows users to build interactive e-Learning content, presentations, and gamified experiences with animations and clickable elements. The platform emphasizes visual design and interactivity (n.d.).

Mindgrasp allows users to upload documents, PDFs, audio, and video lectures, then employs AI to generate summaries, notes, flashcards, and quizzes from the content. The application supports multiple input formats for content processing (n.d.).

Wayground uses artificial intelligence to create gamified quizzes and interactive lessons from existing text, documents, or website links. The tool aims to reduce content creation time for educators while maintaining student engagement (n.d.).

Storywizard.ai creates personalized storybooks that engage children in reading and writing through generated stories with comprehension questions and collaborative storytelling prompts. The platform targets younger learners and emphasizes creative literacy development (n.d.).

1.2.2 Academic Research Projects

EduChat is a large-scale language model-based chatbot system developed for intelligent education in China. The system provides question-answering capabilities over educational materials with multi-turn dialogue support and features such as Socratic teaching and emotional support (Dan et al., 2023).

Learn Your Way by Google is a research prototype that transforms traditional textbooks into immersive, multimodal learning experiences by generating audio lessons, mind maps, summaries, and quizzes. The tool personalizes content according to grade level and student interests, though it currently lacks conversational AI interaction that would allow students to ask follow-up questions or engage in real-time dialogue with the system (Learning & Education, n.d.).

These platforms and research projects demonstrate diverse approaches to AI-enhanced education. Commercial solutions show market-validated implementations addressing specific educational needs, while academic projects explore emerging capabilities and novel pedagogical applications. Each represents thoughtful responses to the challenges of cognitive load management, accessibility, engagement, and educator support.

1.3 Summary of Findings

AI clearly accelerates the learning process, as demonstrated by the numerous projects and educational solutions available. These solutions vary considerably across different aspects of the learning experience: Some may excel in conversational tutoring or domain-specific instruction handling while others may focus more on providing strong capabilities for generating quizzes or interactive presentations. Yet, both from a learner's and from an educator's perspective, it is often necessary to be able to easily customize, extend or even integrate such educational tools within an existing ecosystem of learning solutions. For example, educators may require flexibility to use their own materials and study subjects, or be able to pre-process them in tailored ways, and while available solutions may be content and feature- rich or demonstrate strength in content transformation, they may offer limited capabilities for customization or impose the need for the instructor to revert to using more than one of them to accomplish their goals. Such fragmentation may in turn lead to inefficiencies and disruptions of the learning experience.

Recent research on educational AI architectures emphasizes that effective systems benefit from combining the natural language capabilities of large language models with custom educational components tailored to specific pedagogical goals, creating hybrid solutions that leverage both conversational fluency and domain-specific accuracy (Virvou & Tsihrintzis, 2023b). This integration approach enables educators to maintain control over content while benefiting from AI's interactive capabilities.

Educators need access to tools that leverage generative and agentic AI, empowering them to design, extend, and integrate their own educational vision with minimal friction and at institutional scale. By providing such accessible yet powerful infrastructure, educators can switch from consumers of educational technology to creators and innovators, able to scale personalized, high-quality learning experiences across diverse student populations while maintaining their instructional autonomy and pedagogical integrity. Our research explores how academic investigation of generative AI architectures can contribute to this evolving educational technology landscape.

1.4 Research Questions

Building on the observations above, this study investigates two fundamental questions:

1. RQ1: How effectively can generative AI support the learning process for both students and educators?
2. RQ2: In what ways, mechanisms and agentic interplays can generative AI support the conversion of static, lecture-based learning processes into a more engaging and interactive learning experience?

1.5 Research Objectives

To address these research questions, this study develops and evaluates a set of interconnected AI tools and designs, specifically tailored to enhance the learning experience, that:

- Implement robust Retrieval-Augmented Generation (RAG) mechanisms that ground AI answers in actual educational materials, ensuring accurate and contextually relevant responses in lieu of hallucinated or fabricated information (often generated when "generalist" AI models are employed).
- Explore a variety of AI-fueled ways via which students may interact with the content and deliver an enhanced learning experience. These tools rely on the RAG pipeline and implement novel flows that can
  - assist in understanding the user's learning styles
  - automate the planning of learning objectives and achieve effective summarization of learning material
  - generate quizzes and quests with interactive storytelling for game-based learning experiences and knowledge assessment,
  - enable the development of grounded AI agents for supporting question-answering sessions through which the learners' inquiries are addressed based on the educational content while facilitating multimedia input and output.

The above research objectives align with established principles for trustworthy AI systems, which emphasize reliability, predictability, transparency, and user control as critical dimensions for building user confidence in AI-powered educational tools. Our RAG-based architecture directly addresses these trust requirements by grounding AI responses in verified educational content, providing transparent reasoning through source attribution, and enabling educators to maintain control over the knowledge base (Virvou & Tsihrintzis, 2023a).

1.6 Limitations

The current implementation is limited to English-language content for both technical and practical reasons. In particular: (i) Large Language Models (LLMs) presently are more efficient in English compared to other languages, (ii) most educational materials(e.g. textbooks, academic articles and publications) that served as the foundation for construction our knowledge base were primarily published in English and (iii) the use of English optimizes our computational costs and resource requirements, since English text processing typically requires significantly fewer tokens compared to other inflectionally richer languages.

Another limitation lies in the lack of openly available and accessible annotated datasets that could be used for the out-of-the-box evaluation of the performance of the proposed tools. Albeit libraries for supporting the evaluation of Large Language Model (LLM) applications do exist (e.g. LangTest, Ragas, DeepEval), these libraries often rely on the pre-existence of annotated datasets that can be used for the accuracy assessment. Similarly, a thorough investigation of the retrieval accuracy (in terms of precision and recall) on top of the shortlisted learning material would require the development of such an annotated dataset (i.e. containing a series of questions alongside with their respective expected correct answers as supported by the educational content) which would be a time intense activity (as it requires for extensive human expert supervision) and would go beyond the scope of the present study which mainly aims to leverage the existing proven best practices and mechanisms to address the research questions outlined above, instead of attempting to push the boundaries of the current state-of-the-art in the RAG domain.

Additionally, while our system leverages state-of-the-art RAG mechanisms to ground AI responses in educational materials, research emphasizes that generative AI systems require ongoing human expert oversight for validation, bias detection, and accuracy assessment beyond automated evaluation metrics (Virvou & Tsihrintzis, 2023). The current prototype provides mechanisms for educator oversight, but comprehensive validation frameworks remain an area for future development.

1.7. Structure of This Work

The remainder of this study is organized as follows. Section 2 reviews the relevant literature across three domains: the evolution of educational technology from early computing to modern AI systems, the principles and challenges of personalized learning, and the specific capabilities of generative AI in educational contexts. This review establishes the theoretical foundation that informs our system design decisions.

Building on this theoretical foundation, Section 3 presents the system analysis and design behind our proposition. We define the functional requirements (the specific capabilities the system must provide) and non-functional requirements (quality attributes such as accuracy, performance, and usability). We also present UML-based models, including sequence diagrams that illustrate how system components interact to fulfill user requests.

Section 4 translates these design specifications into concrete implementation. We describe the RAG pipeline architecture that grounds AI responses in educational materials, explain the design of each AI agent (VARK Assessment, Enhanced QA, Smart Digest, Quiz Generation, and Quest Generation), and discuss the technical decisions made to balance performance, accuracy, and cost-effectiveness.

Having detailed the technical architecture, Section 5 demonstrates the system through a functional prototype. We show how each agent operates in practice, illustrating the user interactions and AI-generated outputs that validate our design choices.

To complement the technical demonstration, Section 6 provides a user guide that explains how learners interact with each modality, detailing the interface elements, configuration options, and feedback mechanisms from the user's perspective.

Finally, Section 7 synthesizes our findings, addresses the research questions posed in Section 1.4, and outlines future research directions including production readiness, enhanced retrieval mechanisms, and evaluation frameworks.

2. Literature Review

2.1 Advances in Education Technology

Educational Technology (EdTech) brings together digital tools and teaching methods to help students learn better. It connects teachers, who understand how students learn, with tech experts who build useful digital tools aiming at making learning more effective, engaging, and accessible for everyone via the use of videos, simulations, and hands-on activities (Roblyer & Hughes, 2019).

While early attempts date back to the 1920s, with radio technology entering education and the 1950s and 60s with the introduction of television and films as teaching tools, it was not until the 1980s before computers were first used for educational purposes. Seymour Papert was one of the first educators to recognize the value of this technological advancement and its potential. He designed and developed a programming language for children named Logo, because he believed that students should build things with technology, not just consume it (Papert, 1993). In the 1990s, these academic benefits were further enhanced by means of:

- multimedia computers that could combine text, images, sound, and video all in one place, enabling teachers address different learning styles (visual, auditory, reading and kinesthetic) (Mayer, 2002) all in a single lesson.
- the advent of the internet, breaking down traditional classroom walls and making distance education possible (Anderson & Dron, 2011). This connectivity revolution created the foundation for today's e-learning platforms and massive open online courses (MOOCs), bringing high-quality education to millions of people regardless of where they live or their economic situation. Over time, these early online courses that mainly served as simple file-sharing systems, have evolved to sophisticated learning environments that can support blended learning, track performance, and create personalized experiences for students (Mittal et al., 2024).

While computer-based platforms transformed classroom learning, the next revolution came in the form of mobile learning (via smartphones and tablets). Studies show that well-designed learning apps (in place of heavy textbooks) actually keep students more engaged and help them remember information better (Virvou & Alepis, 2005), while enabling easy access to interactive lessons, videos, and quizzes anytime and anywhere.

Building on these foundational capabilities, the evolution of mobile educational technology has progressed from simple content delivery to sophisticated intelligent systems. In another study, the authors (Alepis & Virvou, 2014) demonstrate this progression through their development of mobile authoring tools that enable educators to create intelligent tutoring systems accessible from any device. Their work illustrates how mobile platforms can support both learner engagement and educator productivity, addressing the dual challenge of student needs and instructor time constraints. This bi-directional support, serving both learners and educators, represents a significant advancement over earlier mobile learning approaches that focused exclusively on content consumption.

Beyond content delivery and authoring capabilities, mobile learning systems have started to recognize and respond to student emotions. Research shows that educational applications can detect emotional states by analyzing both how students speak (tone, pitch, speech patterns) and their facial expressions. These systems can identify emotions like frustration, confusion, happiness, or surprise with accuracy rates between 55-100%, depending on the emotion type (Alepis et al., 2010). This matters because when a system detects that a student is frustrated or confused, it can adapt in real time by simplifying explanations, adjusting the pace, or offering encouragement. Different emotions are better recognized through different channels: positive emotions like happiness show up clearly in facial expressions (90-95% accuracy), while negative emotions like frustration can be detected equally well through both voice and facial cues (55-68% accuracy). This emotional awareness adds another layer to personalization, allowing systems to respond not just to what students know, but also to how they feel during learning.

Parallel to these developments in mobile intelligent tutoring, newer technologies such as virtual and augmented reality (VR and AR respectively) opened the pathway to novel learning experiences such as enabling students walk through historical events, conduct lab simulations, and interact with three-dimensional (3D) models to understand complex concepts (Akçayır & Akçayır, 2017).

The newest frontier in EdTech is artificial intelligence (AI), which can fuel the development of intelligent tutoring platforms that are always available, learn how each student studies best, predict where they might struggle, and suggest personalized learning strategies. These AI systems can act like personal tutors that never get tired and always remember what works best for each student (VanLehn, 2011). AI can recognize such learning patterns and create customized experiences for each person. This opens the road towards an education that is truly data-driven, and where teaching decisions are based on real evidence about what actually helps each student learn. These technological advances have enabled increasingly sophisticated approaches to personalized education, which we examine in the following section.

2.2 Personalization in Education

Personalized education is about moving away from teaching everyone the same way to creating learning experiences that fit each individual. Recent research studies by Chrysafiadi and Virvou (2022) demonstrate that personalized learning environments significantly improve student engagement and learning outcomes by adapting to individual cognitive styles and preferences (Chrysafiadi & Virvou, 2012). Each student is different, comes with different background, learns in different ways, stays motivated by different things, and has different strengths and weaknesses. This in respect imposes the need for personalized education.

Technology nowadays makes this possible in ways never imagined before. Adaptive learning systems use smart algorithms to monitor how students learn, figure out where they're struggling, and adjust lessons accordingly. These systems track indicators such as how long students take to answer questions, what mistakes they make, when they ask for help, which videos they watch, and where they are struggling. Machine learning algorithms can work behind the scenes to analyze this data and figure out the best learning strategies for each student, helping them overcome their struggles before they fail and providing them help when it's most needed (Chen et al., 2020).

Intelligent Tutoring Systems (ITS) take this even further by acting like personal tutors. Studies show that good tutoring systems can help students learn almost as well as working with a human tutor, with the additional advantage of being available 24/7. These systems use cognitive models to understand how learners think (Chrysafiadi & Virvou, 2013), and when combined with mobile technology, they can provide personalized instruction wherever and whenever learning occurs.

The evolution of adaptive learning has seen various approaches to knowledge modeling and personalization. Earlier adaptive learning systems used techniques like Fuzzy Cognitive Maps to model knowledge dependencies between concepts, enabling personalized navigation paths through explicit knowledge modeling (Chrysafiadi & Virvou, 2013a). Research has established that effective personalization requires identifying learner heterogeneity across knowledge level, cognitive abilities, preferences, and learning styles, then adapting content delivery to these individual characteristics (Chrysafiadi & Virvou, 2015). When used in combination with Computer Adaptive Testing (CAT), such systems can provide users with instant feedback and identify specific areas where learners may need additional guidance (Reckase, 2009).

Empirical evaluations of ChatGPT in educational contexts have revealed significant factual accuracy limitations (1.97/10), with the model frequently presenting incorrect information without indicating uncertainty (Virvou et al., 2023), necessitating grounding mechanisms such as RAG for reliable educational applications.

Building on these adaptive learning principles, modern generative AI approaches expand adaptive capabilities by automating content generation and processing.

The use of innovative technologies to facilitate personalized learning comes with multiple benefits:

- Personalized learning pathways can adjust to evolving student needs and changing academic interests over time (Reckase, 2009), allowing them to follow individualized educational sequences tailored to their specific goals, interests, and existing knowledge base.
- Enable microlearning experiences, meaning the delivery of content through concise, targeted segments designed to accommodate individual schedules and learning preferences. This methodology can be particularly valuable for adult learners, where educational activities must be integrated with existing work and personal commitments, and can be further enhanced with AI that analyzes optimal delivery timing, selects appropriate content sequences, and

determines the most suitable presentation methods for individual learners (Drachsler et al., 2015).

- Fosters social learning personalization, where students can collaborate with each other and benefit from peer interactions and group-based activities, while also benefitting from AI that can optimize the learning outcomes by facilitating strategic group formation, matching students with complementary academic strengths and compatible learning styles (Järvelä et al., 2016).

In spite of the numerous benefits, the design and implementation of such personalized education systems present several significant challenges that require careful consideration. Privacy concerns may arise from the extensive data collection necessary for effective personalization, and compliance with national and internation laws and regulations (e.g. GDPR and EU AI Act) must always be considered. These privacy considerations become more complex in AI-enabled educational environments where personalization mechanisms continuously generate inferred data about students beyond the factual information they explicitly provide, such as automated classifications of learning styles, assessments of knowledge levels, and predictions of performance patterns, requiring transparent frameworks for consent management and ethical oversight of automated student profiling (Mougiakou et al., 2025). Ensuring educational equity and preventing algorithmic bias in personalized systems requires attention to system design, testing, and ongoing validation processes (Chrysafiadi & Virvou, 2013). Recent developments in generative artificial intelligence have introduced new capabilities that address many of these personalization challenges while opening additional opportunities for educational innovation.

2.3 The Role of Generative AI in Education

From an educational perspective, Generative Artificial Intelligence (GenAI for short) offers unprecedented opportunities for creating dynamic, personalized, and engaging learning experiences that adapt to individual student needs and preferences (Kasneci et al., 2023).

The multimodal capabilities of modern GenAI systems enable the creation of comprehensive learning materials that combine textual explanations, visual illustrations, audio narrations, and interactive elements. Text-to-image generation models such as DALL-E, Midjourney, and Stable Diffusion can automatically create visual content that supports textual explanations, making abstract concepts more concrete and accessible to visual learners (Brown et al., 2020), which can be particularly valuable in subjects such as science, history, and literature, where visual representation can significantly enhance understanding.

Similarly, audio generation technologies, including text-to-speech systems and music generation models, enable the creation of rich auditory learning experiences via natural-sounding narrations, supporting auditory learners and improving accessibility for students with reading difficulties (Chen et al., 2020).

Lastly, video generation models enable the creation of dynamic visual content that can illustrate complex processes, historical events, and scientific phenomena (Mittal et al., 2024), which similar to the image generation models can make abstract concepts tangible by providing visual representations of processes that are difficult or impossible to observe directly (e.g. simulations of scientific experiments or chemical reactions).

The drastic fusion of the aforementioned powerful modalities can reshape the way GenAI-based educational systems operate. It can enable the development of systems that can automatically create quizzes, practice exercises, and assessment tests based on learning objectives and curriculum requirements. These systems can generate diverse question types (e.g. multiple-choice, short-answer, and essay questions), while ensuring appropriate difficulty levels and alignment with educational standards (Mittal et al., 2024). Unlike static educational materials, AI-generated content can be continuously modified and optimized based on student interactions and feedback. This adaptability ensures that learning materials remain relevant, challenging, and engaging throughout the learning process (Mittal et al., 2024).

Building on these theoretical foundations and technological capabilities, the following section translates the research objectives outlined in Section 1.5 into concrete system requirements and architectural design.

3. Analysis and Design

To address the research questions and objectives defined previously, this chapter presents a detailed analysis of the system's requirements. The chapter is divided into two primary subsections. The first subsection specifies the Functional Requirements, describing the specific behaviors and functions the system must perform. The second subsection details the Non-Functional Requirements, which focus on the system's operational qualities and constraints.

3.1 Core Users

The system is designed to serve two primary user groups:

- **Learners**: These are the primary users. They are contemporary students who benefit from personalized, interactive, and multimedia-rich educational content. Their goal is to engage with learning materials more effectively through tools like interactive Q&A, automated summaries, quizzes, and gamified quests.
- **Educators**: These are the secondary users. They provide the educational source material. Their goal is to transform static course content into dynamic learning experiences without needing extensive technical skills. This type of users can benefit from the system's ability to adapt to different curricula by simply changing the knowledge base.

3.2 Functional Requirements

Functional requirements specify what the system is designed to accomplish. They describe the interactions between the system and its users, as well as the specific processes it must execute to meet the research objectives. The detailed technical architecture that supports these functions, which involves advanced information retrieval and automated reasoning flows, is analytically presented in the following chapter (**Section 4: Implementation Details**).

1. **Core Information Processing and Retrieval**

For the core information processing and retrieval, the system must be able to:

a. Process large collections of lengthy and content-rich educational materials in various file formats, including PDF and text documents.
b. Parse and preserve key structural elements from the source documents, such as headings, lists, and tables.
c. Deconstruct large documents into smaller, semantically coherent segments to create a searchable knowledge base.
d. Retrieve relevant segments from the knowledge base in response to a user's query.
e. Use the retrieved information as context to generate factually grounded and relevant responses.
f. Allow the underlying knowledge base to be updated or replaced, enabling the system to adapt to new subjects and curricula without requiring the core language model to be retrained.

2. **Learner Personalization and Assessment**

Similarly, for learner personalization and assessment, the system must be able to:

a. Implement and run assessment questionnaires to identify the users’ primary learning preferences (Visual, Aural, Read/Write, Kinesthetic).
b. Analyze the received responses to determine the users’ dominant learning style(s).

c. Generate personalized reports with tailored study strategies based on the user's learning profile.

3. Student Interaction and Engagement Tools

Additionally, for an enhanced learner interaction and engagement experience, the system must be able to:

a. Provide an interactive interface for users to ask open-ended questions.
b. Allow users to submit questions that include both text and image inputs for analysis.
c. Generate concise summaries of specified educational topics, with user-configurable length and complexity.
d. Automatically generate multiple-choice quizzes based on selected learning materials.
e. Create interactive, story-based quests to assess knowledge in a gamified format.
f. Provide immediate, explanatory feedback for answers submitted in quizzes and quests.
g. Convert generated text responses into audible speech upon user request.

4. System Interface and Data Management

Finally, for system interface and data management, the system must be able to support:

a. Web-based user interfaces to access all functionalities.
b. Interfaces that clearly display text, images from source documents, and synthetically generated images.
c. Storage and retrieval of user performance data (e.g. from the quizzes and quests).
d. Summarization and visualization of learners’ performance history and progress through interactive dashboards.

3.3 Non-Functional Requirements

Non-functional requirements define the quality attributes and operational standards of a system. They do not affect the system's core functions but are crucial for ensuring the prototype is effective, reliable, and user-friendly for its intended purpose. Towards this direction, the system must be tailored and tuned across the below pillars:

a. **Accuracy**: Generate responses that are factually grounded in the provided educational materials to minimize fabricated or irrelevant information.
b. **Performance**: Process user queries and return responses within a reasonable timeframe to support a coherent and interactive user experience.
c. **Usability**: Provide an intuitive and easy-to-navigate user interface that effectively demonstrates the system's core capabilities to a user.
d. **Reliability**: Perform its functions consistently and predictably under typical usage scenarios for a small group of concurrent users.
e. **Maintainability**: Be designed with a modular structure that allows for clear understanding and modification of its components.
f. **Compatibility**: Be functional and render correctly on a standard, modern web browsers with support for both desktop and mobile devices.

3.4 System Design

To translate the above-outlined requirements into a formal design, this study employs the Unified Modeling Language (UML). UML offers a standard visual language to create a blueprint of the system, ensuring that the design is clear, coherent, and directly linked to the functional and non-functional requirements.

Architecturally, the system is centered on an agentic AI architecture that performs the advanced processing and content generation tasks. These services are managed by a backend layer, which handles the core logic, while a user interface serves as the access point for users to interact with these underlying AI capabilities.

To model the high-level interactions between these logical components we provide the below sequence diagram visualizations, as a representative overview of the system's communication flow.

Sequence Diagram

The Sequence diagram illustrates the interactions between objects or components in a time-ordered sequence. It is utilized to model the flow of messages between different parts of a system to accomplish a specific task. The diagram below models the sequence of messages passed between the user interface, the backend layer, and the AI services to fulfill a key user function, providing a clear view of how different parts of the system collaborate.

The first sequence, shown in Figure 1, illustrates the user registration workflow, showing the collaboration between the Web App, Node Service, and File System. The Node Service receives the user's registration details and validates the data to check if the user's email already exists. It then models the two alternative outcomes: a successful registration resulting in a new user session, or a failure that returns an error message. This sequence defines the system's core logic for new user onboarding and ensures data integrity by preventing the creation of duplicate user accounts.

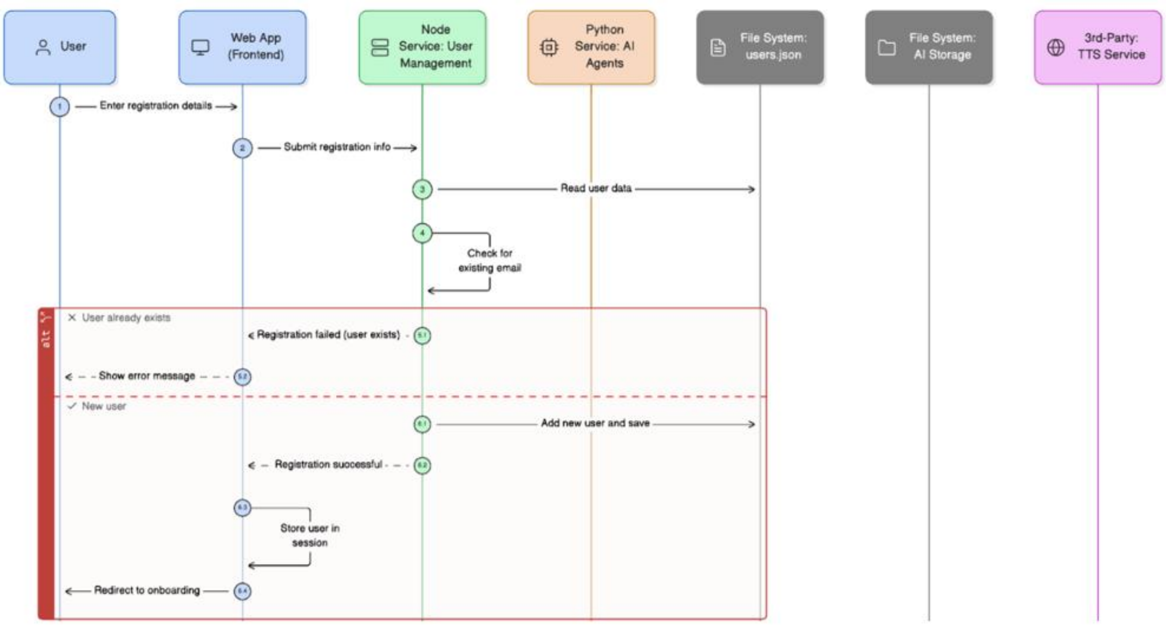


Figure 1: Sequence Diagram illustrating the registration process

Additionally, the process for generating AI-driven content is depicted in Figure 2. This sequence begins when the Web App forwards a user's request for an answer or a summary to the Python-based AI Backend. The AI Backend, acting as a central orchestrator, then initiates the appropriate intelligent process based on the user's request type. During processing, the system provides real-time progress updates by streaming information back to the user updates on the AI agent and related tool updates, concluding when the final generated content is delivered.

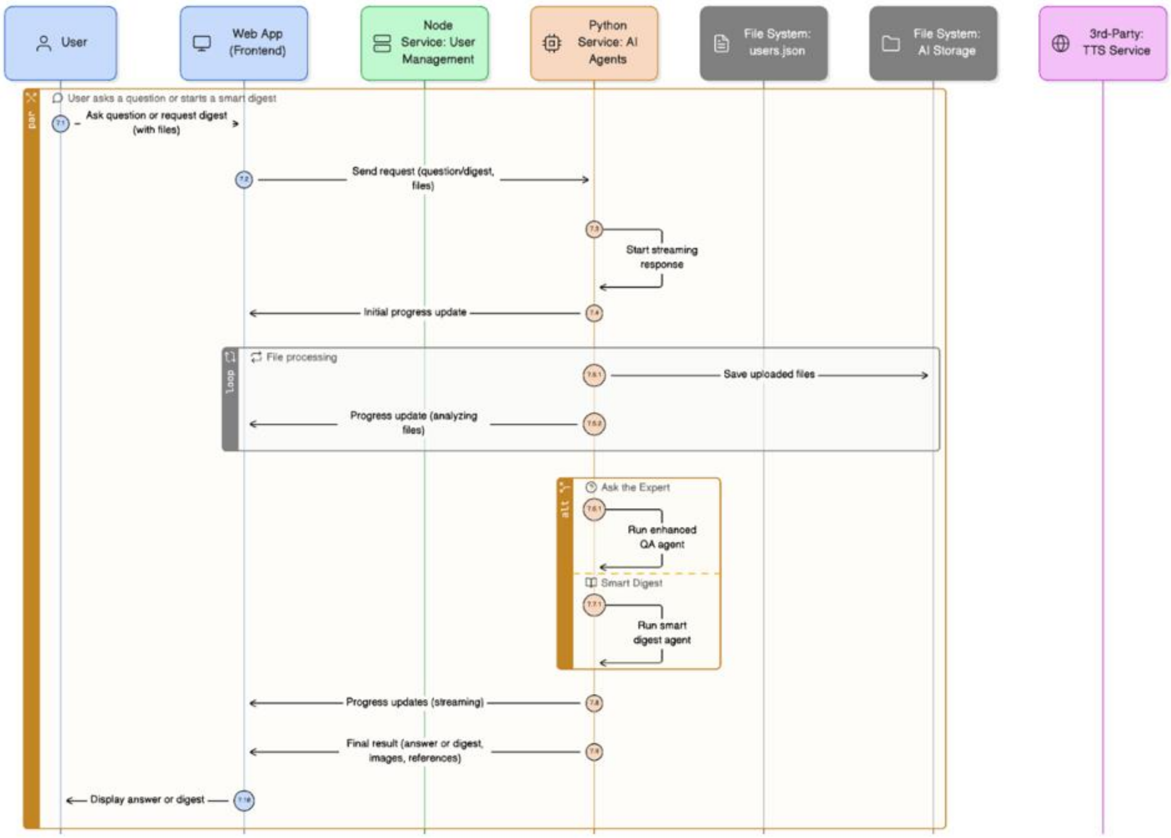


Figure 2: Sequence Diagram of the AI-Driven Content Generation Flow

Furthermore, Figure 3 illustrates the iterative workflow of an interactive assessment, such as a quiz or quest. This turn-based interaction shows the user submitting an answer, which is sent to the Python AI Backend for evaluation. Specifically for the opened ended quest questions and as depicted by this diagram, during the quest process, the user has the ability to request a hint, if support is needed to answer the question. The service provides immediate feedback on the user's submission before presenting the next step in the sequence, a loop that continues until the assessment is complete and the final results are displayed.

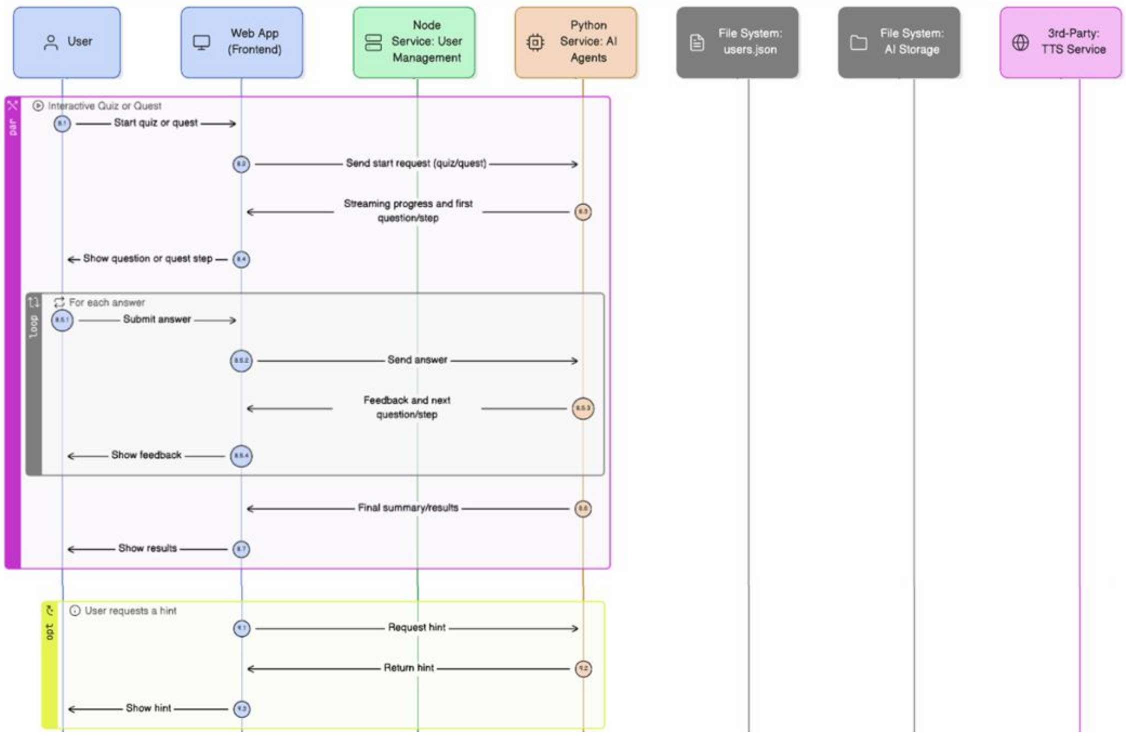


Figure 3: Sequence Diagram for the Interactive Assessment Workflow

Finally, the system's integration with external services is demonstrated in Figure 4. This flow shows the Python AI Backend preparing text content and requesting audio generation from a 3rd-Party TTS Service. Upon receiving the audio data, the backend module saves the file and returns a URL to the Web App, allowing the user to play the generated audio and completing the multimedia output process.

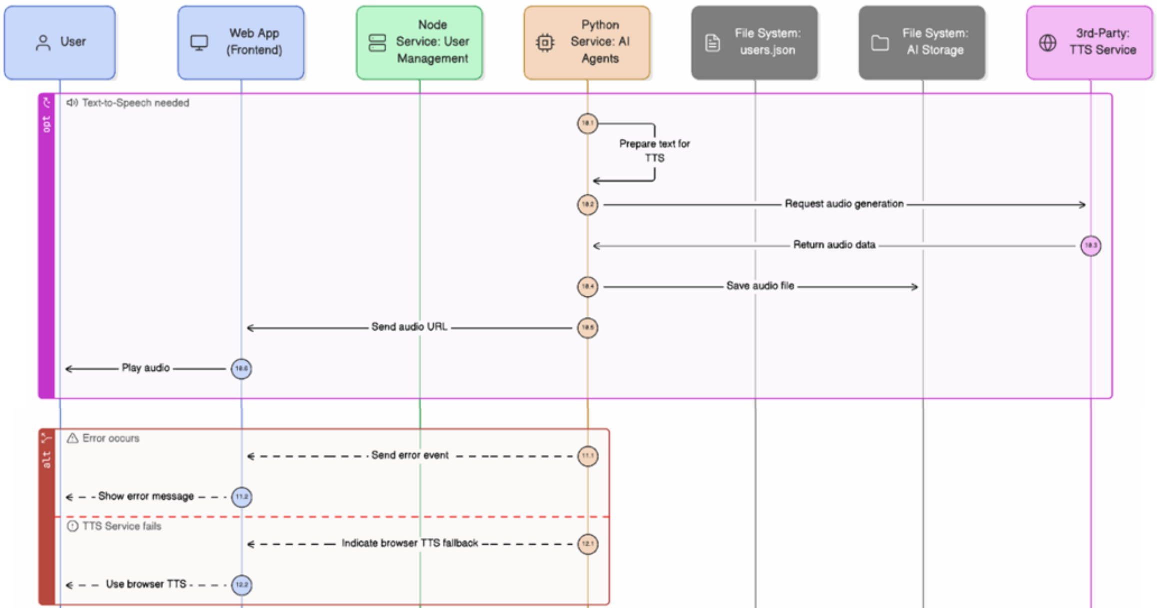


Figure 4: Sequence Diagram for Text-to-Speech Service Integration

4. Implementation Details

To address the research questions and meet the objectives outlined at sections 1.4 and 1.5 respectively, in the course of the present study we created a series of AI-powered tools which can expedite the development of robust educational software solutions that deliver personalized learning experiences and can overcome shortcomings of presently available alternatives. To improve on reliability and factual accuracy of the solution and to address critical limitations of out-of-the-box LLMs such as knowledge staleness and hallucinations, we also implemented a sophisticated Retrieval-Augmented Generation (RAG) pipeline that is being leveraged from our tools to retrieve relevant documents in real-time and use that information as context for generating factually accurate responses. In the next section we provide a foundational overview of our RAG pipeline whereas in the subsequent sections the design principles and architectural interplay of the developed tools are presented.

4.1 Retrieval-Augmented Generation (RAG) pipeline

Retrieval-Augmented Generation (RAG) is often employed to tackle factual accuracy issues, because it combines parametric memory (knowledge encoded in model's parameters) and non-parametric memory (information from external document collections) (Lewis et al., 2021). The fundamental approach behind RAG is that LLMs do not rely only on memorized training data, but instead the system first searches for relevant information in a knowledge base and then uses the context retrieved to produce the final answers. This architecture provides several benefits, such as reduction of hallucinations via the generation of grounded responses that rely on reliable sources as well as the enablement of tactical knowledge updates without retraining the entire model, since updating the document collection immediately makes new information available to the system (Gao et al., 2024). A schematic overview of the way RAG accomplishes the grounding of the responses in provided at Figure 5.

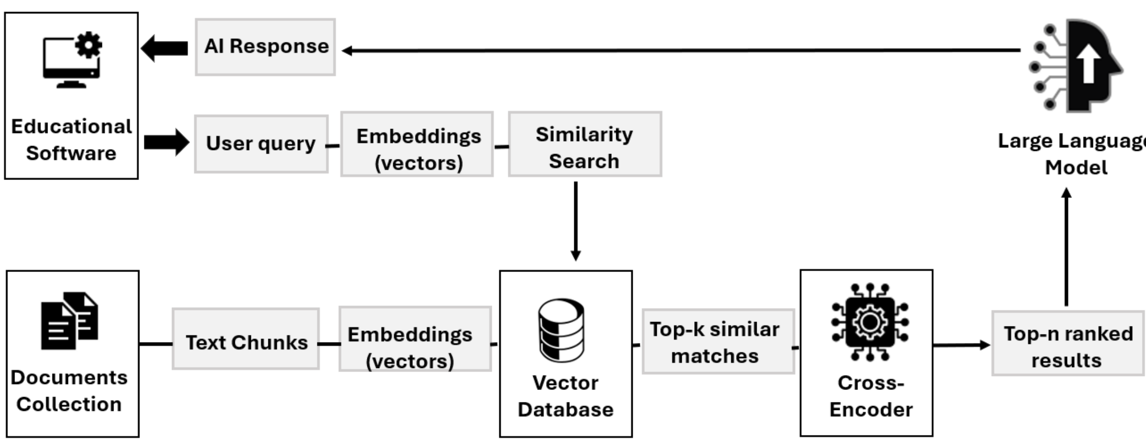


Figure 5: Schematic representation of RAG-based retrieval

An often required first step towards the implementation of RAG-based pipelines is the creation of the vector database based on the available educational material, also referred to as "indexing". Indexing may occur once or periodically and refers to the stage of data preparation, where raw documents are transformed into a structured, searchable representation of knowledge. This step is necessary because LLMs have limited context windows and hence cannot process entire document collections simultaneously (Liu et al., 2023). The indexing phase also decomposes to smaller required steps that are visually presented at the below figure (Figure 6).

Figure 6: Index and vector database creation with RAG-based architecture

A. Parser

The goal of the parser is to extract structured text from a wide variety of document formats (e.g. pdf, word, text files, presentation or spreadsheet files etc.). The quality of parsing plays a crucial role as it strongly influences the subsequent steps and affects the overall effectiveness of the entire system. Poor text extraction may lead to information loss and reduced retrieval accuracy. Additionally, parsers must be able to handle and convert diverse file formats into a uniform text representation and it is important that during extraction process, the documents' structures (e.g. headings, tables, images, lists etc.) are maintained as accurately as possible during the extraction step (n.d.).

For the needs of the present study, we employed the Docling open-source parsing library developed by IBM (Auer et al., 2024), because it has been reported to outperform similar solutions in RAG-based scenarios and due to its inherent ability to handle complex file layouts, while preserving structural elements, and maintaining figures and tables during extraction.

B. Chunker

Following parsing, the next step involves breaking down the extracted text in smaller, semantically coherent pieces of text (also called “chunks”) to overcome inherent limitations of language models and embedding systems, which can only process a fixed number of tokens at a time (Liu et al., 2023). As described earlier, maintaining the semantic coherence during the chunking process is tightly tied to the retrieval performance of the system and for this reason the selection of the chunking method is an important decision.

For the meeting our chunking objectives in the present study, we employed Docling’s hierarchical chunker as it respects the natural boundaries of the text such as paragraphs and sections, matches them with the relevant chapter titles, and hence yields better retrieval accuracy (Auer et al., 2024).

C. Tokenizer and Embeddings Encoder

After chunking, the next step involves invoking a tokenizer to convert the text in each chunk into even smaller pieces (called tokens) in order to be fed in a neural encoder that will translate it to its corresponding vectorized representation. During tokenization, the tokenizer maintains semantic information, and produces consistent numerical representations (that are consistent both during the indexing and the query phases) and hence directly affect the performance of the RAG pipeline (Kudo & Richardson, 2018).

As an immediate next step, the embeddings encoder steps in to transform tokens into dense vector representations that preserve semantic relationships of text. This is achieved by converting the discrete text into its continuous vector space equivalent, where semantic relationships can be measured using distance metrics (Reimers & Gurevych, 2019). These vectors are called "embeddings" and manages to preserve semantic similarity among texts (i.e. similar texts are placed closer together in the vector space), enabling thus sharper retrieval accuracy even against searches with low text (character)-based similarity.

In the context of the current analysis, the all-MiniLM-L6-v2 tokenization and encoding model was employed. This is a lightweight transformer with only 6 layers and ~22 million parameters, able to produce 384-dimensional embeddings that enable rapid inference and low-latency search results. Its compact footprint makes it ideal for scenarios with limited hardware resources or use cases demanding near for real-time responses (e.g. chatbots, high-throughput APIs, and edge deployments) (2024a).

D. Vector Database

As a last step, the generated vector embeddings are stored in fast, vector database stores that are specialized to enable fast similarity searches using advanced indexing techniques (e.g. graph-based approximate nearest neighbor search algorithm such as HNSW). Vector databases are fundamental for RAG systems as they facilitate the very fast and efficient retrieval of relevant context, even in queries against millions of documents (Johnson et al., 2017) by using distance metrics to determine semantic relevance among documents.

For the present analysis, the Qdrant vector database was used, because it offers several advantages such as its high-performance indexing, support for metadata filtering during search, ability to handle the 384-dimensional embeddings from the all-MiniLM-L6-v2 model, and free availability for datasets up to 2GB, making it cost-effective for educational applications (n.d., 2024).

4.2 Query Phase and Cross-Encoder Integration within the RAG pipeline

Albeit steps A and B described above are typically executed only once or when database updates are required (e.g. for the editing the available documents in the knowledge base), the encoding and retrieval steps (C and D) are also required when a user poses a new question (please refer to the top – query phase – in Figure 5).

Specifically, in the query phase, a user submits a question which in turn is converted into an embedding (always using the same model that was used during the indexing phase, meaning the all-MiniLM-L6-v2 model in the present study), following the exact same embedding mechanisms explained above. The system then executes a similarity search query against the vector database (i.e. Qdrant in the present study) to retrieve a set with the top-k most similar (based on the computed vector distances) and hence likely semantically relevant document chunks. The retrieved chunks can then be provided as "additional context" to the LLMs and AI agents, which will consider that provided information to consequently formulate a factually accurate response that is not only based on the parametric memory of the linguistic model but also on the semantically related retrieved documents.

Although many RAG-based systems rely exclusively on aforementioned mechanism for grounding LLMs with retrieved context, studies have indicated that the use of an additional "cross-encoder" model as a post processing step may enhance the quality of retrieved information (Gao et al., 2024) and consequently often yields results of better quality. Cross-encoders are transformers that process both the query and each candidate document together as a single input, enabling the model to understand how the two texts relate to each other and by these means it may produce more accurate relevance scores (Reimers & Gurevych, 2019).

The way that the encoder and cross-encoder work with each other results in implementing a two-stage reranking approach. In the first stage, the bi-encoder rapidly retrieves a set of candidate matches (configured to 25 matching hits by default for the needs of the present study) and in the second stage, the cross-encoder re-ranks these candidates to identify the shortlist the finally selected (higher scoring) relevant results (44) (configured to select the top-8 final results, out the 25 retrieved in the first step).This two-stages approach helps in further reducing hallucination issues and improve the overall reliability of the RAG-based components in the solution (Reimers & Gurevych, 2019).

For the needs of our study, the ms-marco-MiniLM-L-6-v2 model was used as cross-encoder. This is an open-source, lightweight encoder (~22 million parameters), which has received specialized training against the MS MARCO dataset that contains >8.8 million passages and real search queries (Shi et al., 2023).

4.3 “VARK Assessment” tool: Assessing Users’ Learning Styles

The first tool we developed is aiming to capture and assess the students’ learning preferences across four sensory modalities, meaning Visual, Aural (Auditory), Read/Write, and Kinesthetic, and can help users understand how they learn best, encouraging the adoption of strategies tailored to their preferred modalities for more effective information retention and academic performance.

It is based on the VARK model, initially developed by Neil Fleming in 1992 (Fleming & Mills, 1992) , and through a series of multiple-choice situational questions aims to classify learners based on their dominant modes of processing information.

The learner is presented a series of questions, each of which contains four options corresponding to the four sensory modalities and has to choose the answer which best explains their preferences. The answers are then grouped per sensory modality (i.e. total number of Vs, As, Rs and Ks selected) and the aggregated statistics are analyzed to determine their dominant(s) learning styles.

For the needs of our implementation, we used the questionnaire available at (2014) whereas for the VARK results analysis we developed a custom LLM chain (see Figure 03 below). An LLM chain is a structured pipeline composed of sequential operations that enables developers to produce contextual, domain-specific responses by connecting models, and workflow logic into unified, customizable processes.

In our case, the gpt-4o-mini large language model from OpenAI was employed (n.d.) which was instructed to analyze the VARK results across various dimensions (learning style interpretations and dominant styles inference, personalized learning recommendations, key insights for the user and suggestions for study strategies to the learner) as depicted at Table 1 (in the Appendix). As programming language of preference, we decided to use python (due to its wealth in AI libraries availability within the open-source community) and more specifically the leveraged the LangChain and LangGraph open-source frameworks because of their inherent capabilities to handle complexity, accelerating the development of powerful agent-based applications with ease (Mavroudis, 2024).

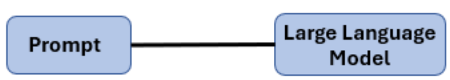


Figure 7: Schematic overview of the VARK Assessment AI Chain

4.4 “Enhanced QA” Agent: Interactive Question-Answering

The second tool in our portfolio is an AI agent equipped with a set of powerful (also AI-enabled) tools that allows the learner to ask open-ended questions to an AI agent and receive back grounded replies which rely on leveraging the tools that the agent has at its disposal as well as the information contained in the provided learning material. Unlike traditional question-answering systems that provide direct responses, agents can create plans to decompose complex requests into manageable subtasks, invoke external tools, maintain context across multiple interactions, and adapt their strategies based on intermediate results (Wang et al., 2024).

One key element of AI agents is their inherent ability for autonomy and behavioral versatility that enables to plan their execution steps on invoke time. For example, an agent may decide to use none, one or more of its available tools (even sometimes reuse a tool multiple times in context of a single question handling) in order to address the user's inquiry. This characteristic of agents is very important for determining the successful behavior of the present tool, since learners may interact with the agent across a variety of topics, include multimedia input in the questions and/or follow-up in the conversation by asking for additional clarifications.

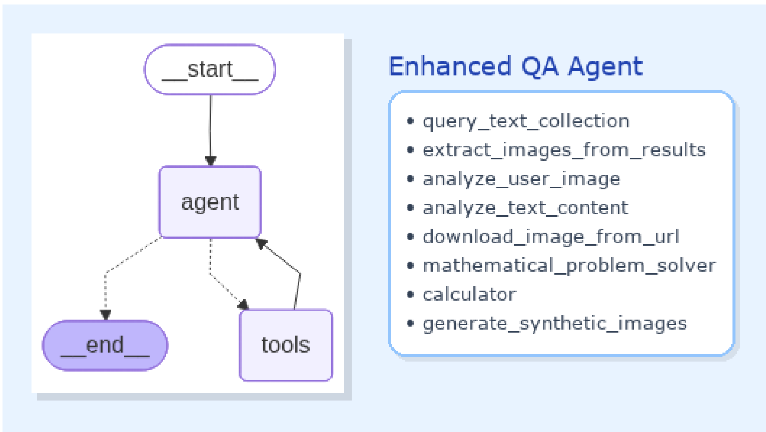


Figure 8: Visual representation of the "Enhanced QA" agent and its attached tools

As shown in the image above, the "Enhanced QA" agent is equipped with eight additional tools:

A. Query text collection tool

When this tool is called by the agent, it leverages the RAG-based pipeline that was described in sections 4.1 and 4.2, that facilitates the retrieval of the top-n most semantically similar documents from the vector database that match the input question (as originally posed by the asker or as rephrased by the agent, e.g. to provided additional context from the existing chat history). The code corresponding to the "Query text collection" tool is provided at Table 2 (in the Appendix).

B. Extract images from results tool

This tool is responsible for previewing the (raw) document pages to extract images (if existing) from a given page in the document. This tool is called by the agent right after the Query text collection tool, so that together the textual information deriving from the RAG-based pipeline, all images are also extracted from the pages of the documents where the (RAG-based) matches were shortlisted. By this tool, the Enhanced QA agent does not only ensure factual accuracy in the generated responses but also can provide visual extracts of the retrieved context which may appear to be even more helpful for learners with string visual preferences. The code corresponding to the "extract images from results" tool is provided at Table 3 (in the Appendix).

C. Analyze user image/ Analyze text content/ Download image from URL tools

The "Enhanced QA" agent enables the user to also provide additional information (e.g. attach accompanying files, images or web links to image files) to be leveraged in the context of question answering. For example, the learner may ask the agent to explain in simple terms the relationships depicted in a complex network diagram (image) or the information contained on a file locally stored on the user's device. These tools are also AI-based (e.g. the analyze_user_image leverages the image-to-text capabilities of the gpt-4o model from OpenAI) and serve to enable the agent to extend to also these capabilities and the agent may autonomously decide to use (one or more of these tools) based on its planning to best address the user's question. The code corresponding to these tools is provided at Table 4 (in the Appendix).

D. Mathematical problem solver tool

This is an AI-based tool (i.e. "agent as a tool") that is also provided to the "Enhanced QA" agent. This tool enables the agent to dynamically generate and execute python code that can help address learners' inquiries (e.g. "please help me solve this mathematical problem please). The tool itself leverages another LLM model, namely gpt-4o from OpenAI (n.d.) and contains instructions that direct the LLM to think and generate tailored python code that can be executed (in a safe python runtime environment) in order to address the received mathematical challenge. The code corresponding to the "mathematical problem solver" tool is provided at Table 5 (in the Appendix).

E. Calculator tool

Similarly to the "mathematical problem solver" tool, the "calculator" tool is a tool that can receive single line mathematical expressions and calculate their equivalent result using Python's numexpr library. This can be triggered by the agent when simpler mathematical results need to be computed. The code corresponding to the "calculator" tool is provided at Table 6 (in the Appendix).

F. Generate synthetic images tool

The last tool provided to the "Enhanced QA" agent is the "generate synthetic images" tool. This tool is using the Pollinations AI web service (selected for its open-source availability and free access), to automatically generate custom synthetic visualizations that relate to the user inquiry and retrieved results (deriving from the "query text collection" tool described above). This tool is optional (the agent will trigger it only if the caller enables the synthetic image generation) and will enrich the agent's generated response with synthetic (AI) related images. The code corresponding to the "generate synthetic images" tool is provided at Table 7 (in the Appendix).

The tools leveraged by the "Enhanced QA" agent have been implemented in a shareable and reusable way so that can be leveraged (if and when needed) by the remaining agents in our study.

4.5 "Smart Digest" Agent: Effective summary generation

The third learning capability explored in the present study is a graph-based AI flow that enables learners create effective summaries on top of their selected subject areas and in context of a preferred topic they would wish to improve on. This agent is very effective in supporting microlearning engagements and can be particularly valuable for adult learners with fractured and time-constrained learning schedules, to whom it may provide concise, targeted education content summaries, tailored to align with their individual schedules and learning preferences. For example, a learner that is currently challenged to thoroughly review a learning subject due to existing work and personal commitments, may revert to this agent to retrieve a condensed summary of that subject area and topic.

The reason for choosing to implement this capability as an AI graph-based flow instead of an AI agent relies on the fact that in contrast to AI agents, which can autonomously and reason themselves about which actions or tools to invoke at each step, the graph-based AI flows provide enhanced transparency and improved governance on every step, node transition and decision being made during the agent's execution. This is a very important detail for the "Smart Digest" agent, since in order for it to deliver a successful condensed summary, several thinking and reasoning steps need to be executed (in proper order in via parallel calls to speed up the summary generation) as depicted at Figure 9.

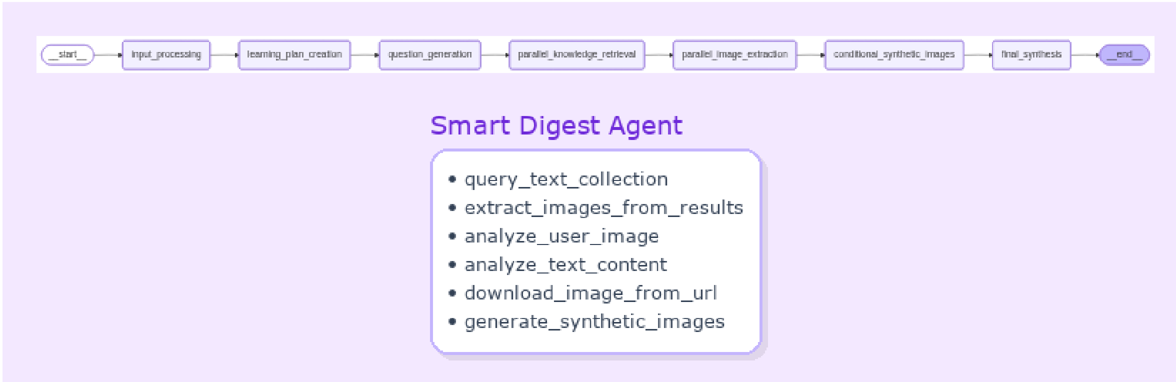


Figure 9: Visual representation of the "Smart Digest" graph-based flow

As visualized in Figure 9, the "Smart Digest" agent also leverages six tools (i.e. query text collection, extract images from results, analyze user image, analyze text content, download image from URL and generate synthetic images) which are shared with the "Enhanced QA" agent and are described in detail at section 4.4. These tools are being called during the graph's "start to end" traversal. In particular, when the "Smart Digest" agent receives a request to generate a summary for a given topic of interest and subject areas, it will:

1. First analyze the provided input (i.e. selected subject areas, educational topic, additionally provided information from attached documents etc.)
2. Elaborate on a detailed learning plan (i.e. leverage a reasoning LLM chain to reflect on all learning objectives that would need to be covered in the generated summary). For example, if the learner wants to learn about "sorting algorithms" in programing, this step may generate a learning plan as the one provided below:

Introduction to Sorting Algorithms

Objectives: Understand the definition and purpose of sorting algorithms., Identify different types of sorting algorithms and their use cases.

Key Concepts: Sorting algorithm definition, Use cases for sorting algorithms

Search Focus: introduction to sorting algorithms, types of sorting algorithms

Common Sorting Algorithms

Objectives: Learn about popular sorting algorithms such as Bubble Sort, Selection Sort, and Insertion Sort., Compare the efficiency and performance of these algorithms.

Key Concepts: Bubble Sort, Selection Sort, Insertion Sort

Search Focus: Bubble Sort, Selection Sort, Insertion Sort, algorithm efficiency

Advanced Sorting Algorithms

Objectives: Explore more complex sorting algorithms like Merge Sort, Quick Sort, and Heap Sort., Understand the divide-and-conquer strategy used in some sorting algorithms.

Key Concepts: Merge Sort, Quick Sort, Heap Sort, Divide-and-conquer strategy

Search Focus: Merge Sort, Quick Sort, Heap Sort, divide-and-conquer

Algorithm Complexity and Big O Notation

Objectives: Understand the concept of algorithm complexity and how to analyze it., Learn about Big O notation and its significance in evaluating sorting algorithms.

Key Concepts: Algorithm complexity, Big O notation, Time complexity, Space complexity

Search Focus: algorithm complexity, Big O notation, time complexity, space complexity

Practical Implementation of Sorting Algorithms

Objectives: Implement various sorting algorithms in a programming language of choice., Debug and optimize sorting algorithms for better performance.

Key Concepts: Code implementation, Debugging techniques, Optimization strategies

Search Focus: implementing sorting algorithms, debugging sorting algorithms, optimizing sorting algorithms

Real-World Applications of Sorting Algorithms

Objectives: Identify real-world scenarios where sorting algorithms are applied., Understand the impact of sorting algorithms on data processing and retrieval.

Key Concepts: Data processing, Data retrieval, Applications in databases and search engines

Search Focus: real-world applications of sorting algorithms, sorting in databases, sorting in search engines

3. Based on the generated learning plan, the agent will then develop (in parallel) AI generated inquiries that will be executed (also in parallel) against the RAG-based retrieval pipeline (described in sections 4.1 and 4.2). An example of such generated question for the above exemplary learning plan is provided below:

Q1
introduction to sorting algorithms, types of sorting algorithms Sorting algorithm definition Use cases for sorting algorithms
Q2
Bubble Sort, Selection Sort, Insertion Sort, algorithm efficiency Bubble Sort Selection Sort Insertion Sort
Q3
Merge Sort, Quick Sort, Heap Sort, divide-and-conquer Merge Sort Quick Sort Heap Sort Divide-and-conquer strategy
Q4
algorithm complexity, Big O notation, time complexity, space complexity Algorithm complexity Big O notation Time complexity Space complexity
Q5
implementing sorting algorithms, debugging sorting algorithms, optimizing sorting algorithms Code implementation Debugging techniques Optimization strategies
Q6
real-world applications of sorting algorithms, sorting in databases, sorting in search engines Data processing Data retrieval Applications in databases and search engines
Q7
sorting algorithms [Programming] [Programming]

4. If the user desires additional (synthetic) images related to the study subjects, the AI graph-based agent will also trigger (in parallel) the synthetic image tool to generate such synthetic images for each of the learning areas on the generated learning plan

5. When all parallel calls have returned results, the top-n matching results for each retrieval inquiry coupled with the respective extracted or synthetically generated images will be then fed into the last step of the agent that will analyze them to generate a final, condensed summary (tailored to the difficulty level and length defined by the learner).

4.6 "Quiz" Agent: Efficient knowledge quiz generation

The fourth learning capability developed enables the creation of quizzes based on user defined learning objectives and curriculum requirements. This module can be used by learners for practice purposes or for running personal assessment tests and can help them identify learning gaps and areas for improvement in a pleasant, interactive way. Similarly to the "Smart Digest" agent, for the needs of the "Quiz" agent we also relied on the usage of graph-based AI flows for achieving improved governance and performance in the quiz generation process. Figure 10 provides a visual overview of the steps governing the execution of the "Quiz generation" agent.

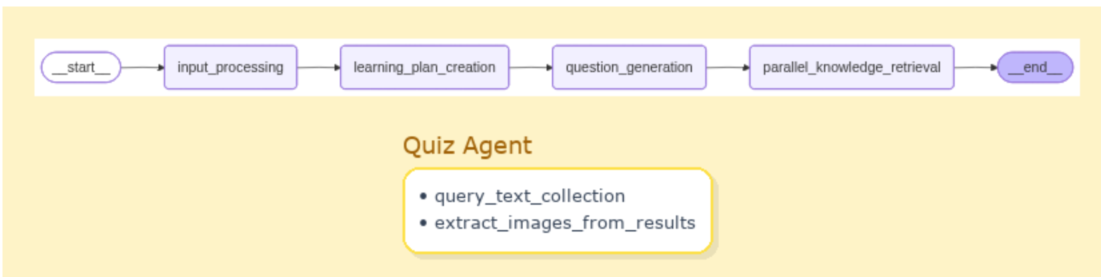


Figure 10: Visual representation of the "Quiz generation" graph-based flow

As visualized in the above figure, the "Quiz generation" also relies on the shared RAG-based tools (i.e. query text collection and extract images from results tools) to extract semantically related information from the knowledge base and traverses the nodes in a graph as follows:

1. Like in the case of the "Smart Digest" agent, the "Quiz generation" agent will also start by analyzing the provided input and then proceed with creating a detailed learning plan and a set of retrieval questions that will be used to extract content from the knowledge base that is highly relevant to the formulated learning plan and objectives therein (please refer to the examples in section 4.5 for a more detailed explanation of these steps)
2. Then, based on the knowledge based extracted content, it will reflect and generate M (where is M is a user defined integer number) of multiple-choice questions, each containing C (with C being another user defined integer number ranged between 2-6) possible answers among which only one is correct whereas the remaining C-1 options are wrong. The difficulty of the erroneous answers can also be defined as a parameter passed to the agent.
3. Last, it will iteratively present these questions one by one to the user, and after the learner submits their answer to each of the presented questions, the agent will undergo another reasoning step, where it will contrast the user selected choice to the correct one and provide detailed feedback as per the correctness of the chosen answer (or similarly proper justification - supported by the retrieved references as well learning recommendations, in case of a mistaken user selection).

When all multiple-choice questions have been displayed and answered, the agent will compute and log (in the form of locally hosted json files) the quiz footprint. This contains information around the quiz such as the user's score, the date the quiz was taken, the learning subjects and topics examined, the user defined difficulty level, the exact questions that were included in the quiz, the multiple choices displayed per question, user provided answers and correct or mistaken answered results coupled with the AI generated justifications). The storage of this information enables the creation of AI-driven summary statistics per each learner, that students can preview to monitor their progress over time and obtain insights as per the learning areas and material they may need to further focus to address currently identified weaknesses. An example of such a quiz summary report is provided at Figure 11.

4.7 "Quest generation" Agent: Learning via gamification

The last learning capability that was implemented bears high resemblance to the "Quiz generation" agent described earlier, with the notable difference being that the "Quest generation" agent generates knowledge assessment test through a gamified learning experience. It follows the same steps as the "Quiz" agent to formulate a comprehensive learning plan and then proceeds by generating a coherent story around a user-defined theme, via which it assesses the learner's understanding on the study subjects by means of multiple choice or open-ended questions that are presented in the context of the story. Figure 12 graphically represents the flow behind the "Quest generation" agent.

Albeit all proposed agents are coupled with mechanisms to facilitate the automated Text-to-Speech (TTS) synthesis and reproduction of all generated outcomes coming from the agents, what is a unique characteristic of the "Quest generation" agent is the AI-enhanced generation of the quests' audio narrations, which allow for the definition of a variety of voices, tones and personalities resulting in a richer and a completer gamified learning experience.

Last, and in alignment with the "Quiz generation" agent, this agent also implements mechanisms for capturing and logging the detailed statistics on top of the learners' executed assessments (please refer to the examples provided at section 4.6 for more details).

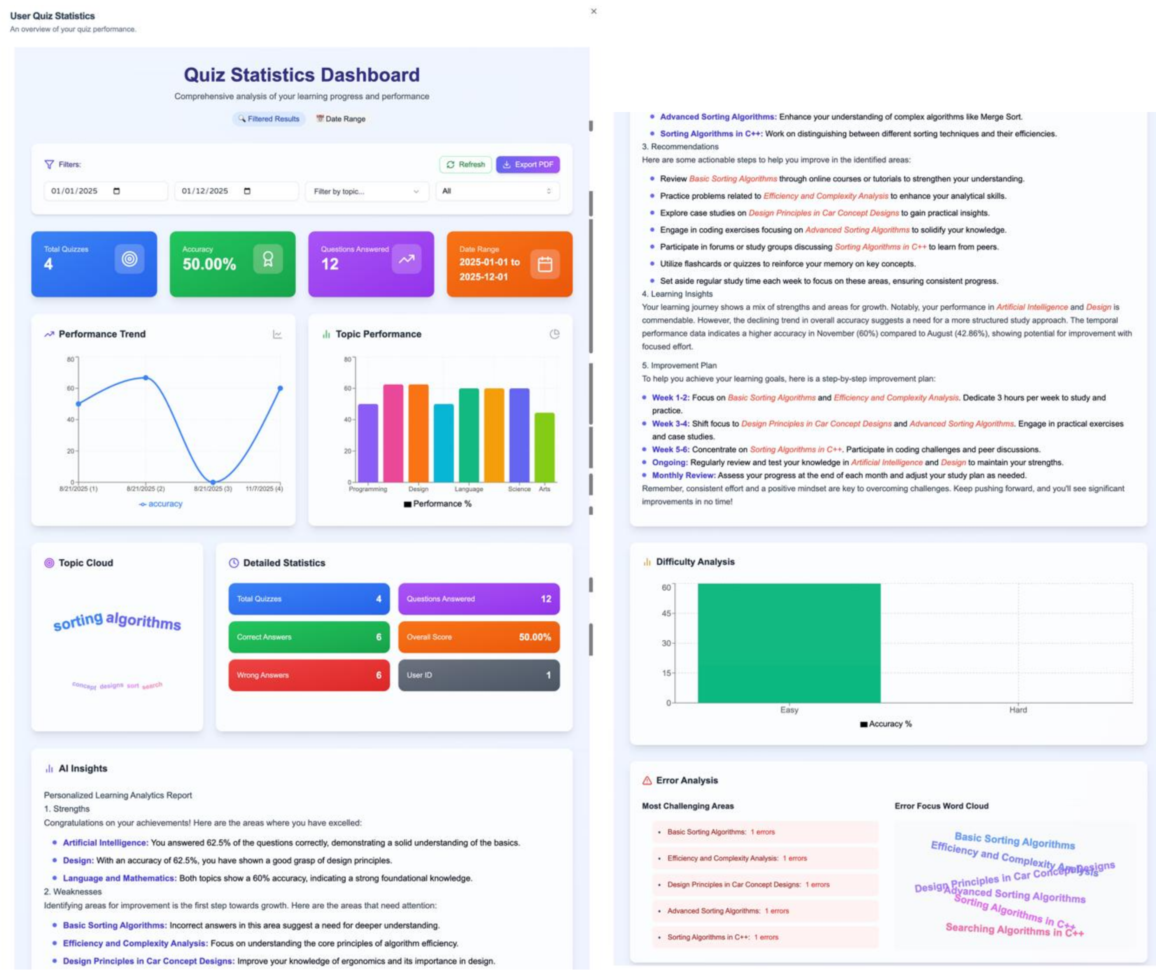


Figure 11: Visual representation of the "Quiz Statistics Dashboard" summarizing the learner's performance over time and providing detailed AI driven insights for further improvement.

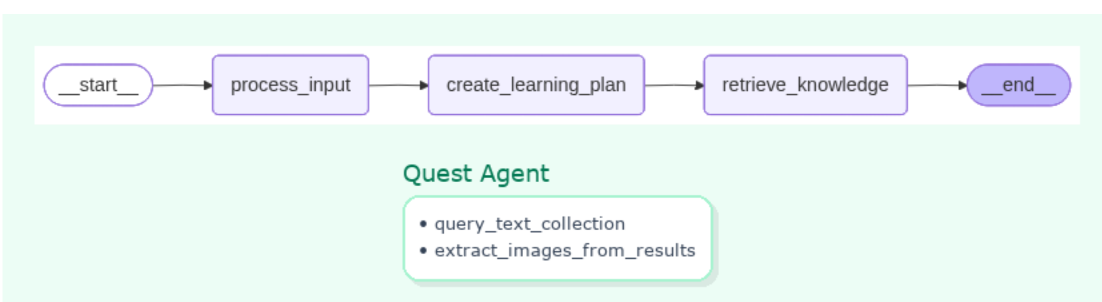


Figure 12: Visual representation of the "Quest generation" graph-based agent

4.8 Entity relationship diagrams and data layer interactions

Entity Relationship Diagrams (ERDs) are very useful for the designing and visualization of the data layer interactions and underlying system architecture. The majority of the AI agents outlined in the previous sections do not interact with structured or unstructured data, with the exception being the “Quiz” and “Quest” generation agents which enable a detailed logging mechanism of the quizzes and quests executed by the learner, aiming to facilitate the generation of progress updates throughout the student’s learning journey. A detailed overview of the Entity relationships among the data entities of those agents is provided below.

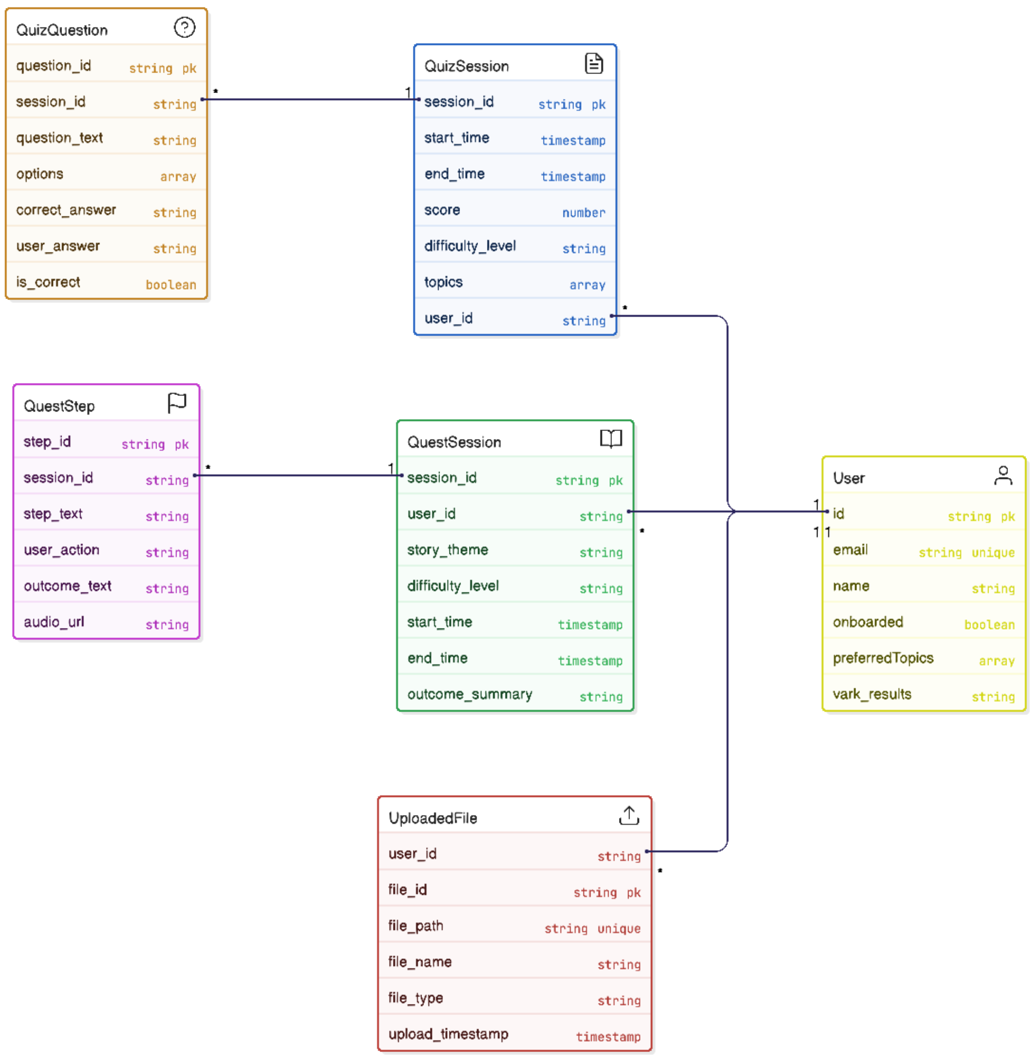


Figure 13: Entity Relationship Diagram (ERD) reflecting the data layer interactions of the “Quiz” and “Quest” generation agents

4.9 Implementation Challenges

The development of the AI-powered tools described in the previous sections presented several technical challenges that required careful analysis and repeated optimization cycles. The first critical challenge emerged during initial testing of the retrieval mechanism. When retrieval operations relied solely on the user-provided topic, the system executed a direct semantic search against the vector database using only the topic keywords. This narrow approach risked incomplete or fragmented responses because the LLM generated answers were based exclusively on chunks retrieved through this limited topic-based search, failing to meet the holistic learners' needs for in depth subject understanding. For instance, a learner asking to learn about "sorting algorithms", if only the entered keywords are used for the RAG-retrieval, may end up receiving content on a few specific algorithms, lacking nonetheless foundational knowledge governing the "sorting algorithms" topic, such as algorithm complexity, Big O notation, or practical implementation strategies.

We addressed this challenge by developing a concrete "learning plan generation" mechanism (sections 4.5, 4.6, and 4.7), which instructs the LLM to analyze the user-selected topic and identify all related areas needed for comprehensive understanding. Via this mechanism, an AI-generated structured learning plan with multiple objectives and key concepts can be generated, covering more broadly the learning subject. Based on the elaborated learning plan, the system then generates new, targeted retrieval queries for each learning objective, which are executed against the vector database to retrieve relevant results. This ensures the retrieved context covers all necessary foundational and advanced concepts, enabling educationally complete responses.

The learning plan generation mechanism solved the completeness problem but introduced new performance challenges. More specifically, the RAG pipeline (sections 4.1 and 4.2), which serves as the foundation for all agents, retrieves semantically relevant document chunks and provides factually grounded context to the LLMs. The multi-step process of query embedding, vector similarity search, cross-encoder reranking, and context assembly introduces latency. Combined with learning plan generation, initial tests showed that creating a detailed plan and executing retrieval operations sequentially could take up to five minutes, an unacceptable delay that would undermine the tools' educational value.

We addressed this challenge through extensive parallelization and critical architectural decisions. Instead of implementing all capabilities as fully autonomous agents reasoning through each step sequentially, we used graph-based flows for the "Smart Digest," "Quiz," and "Quest generation" agents (sections 4.5, 4.6, and 4.7). This choice enabled the parallel execution of multiple RAG inquiries, for each of learning plan topics that were identified. While we identified that a purely agent-based approach could require introduce additional (reasoning) delays, or at times could even lead to sequential execution (based on the inferred agent's execution plan), whereas the graph-based structure allowed for the parallel execution of retrieval queries as soon as individual objectives were formulated. This strategy substantially reduced the total processing time and made interactions feel more immediate and responsive.

Additionally, ensuring retrieval reliability required careful evaluation and selection of parsing and reranking components. For example, the RAG pipeline's quality depends fundamentally on accurate document parsing during indexing (section 4.1.A). Towards this end, we tested several open-source parsing libraries, such as pdfplumber, Docling and PyMuPDF (Auer et al., 2024, n.d., n.d.), each with distinct trade-offs between parsing accuracy, structural preservation, and computational cost. After evaluating these parsers across diverse document types (textbooks with complex layouts, presentation slides, technical papers with mathematical notation), we selected Docling for its ability to preserve structural elements like headings, tables, and figures while keeping processing costs manageable. Our findings are also in alignment with similar studies reported in online technical blogs and practitioner-oriented articles (Eversberg, 2025).

We further enhanced the retrieval accuracy by integrating the ms-marco-MiniLM-L-6-v2 cross-encoder (2025, n.d.) as a reranking step (section 4.2). This two-stage approach, where the bi-encoder retrieves 25 candidate matches and the cross-encoder reranks them to select the top-8 results, filtered out less relevant chunks and ensured the AI kernel receives the highest quality of input context.

The fourth challenge was balancing performance, completeness, and accuracy with low implementation and operational costs, a challenge that was particularly relevant to be addressed for this academic study. Academic research and educational institutions operate under tight budget constraints and cannot rely on expensive proprietary models or extensive computational infrastructure. This constraint a compass during the selection our architectural components and decisions. For example, the selection of which Large Language Model to be employed for which part of the AI execution was particularly critical. To mitigate cost overruns, we used gpt-4o-mini/ gpt-4.1-mini for most agent operations and gpt-4o/ gpt-4.1 for complex tasks requiring more advanced reasoning (such as the mathematical problem solver in section 4.4). These GPT-4 family models deliver strong accuracy and reasoning while maintaining significantly lower costs than full-scale counterparts, making them suitable for educational applications with high query volumes. Similarly, with regards to the embeddings and cross-encoding requirements, we chose the all-MiniLM-L6-v2 (2024b) embedding model and ms-marco-MiniLM-L-6-v2 (2025, n.d.) cross-encoder (section 4.2) for their lightweight architecture and strong performance on commodity hardware, eliminating expensive GPU requirements.

Towards further reducing operational costs while providing robust agent development capabilities, we also employed cost-cutting strategies whenever an effective free option was available. For example, for the needs of vectorized information preservation, the Qdrant vector database (n.d.) offered a cost-effective free tier for datasets up to 2GB (section 4.1.D), the Pollinations AI service served us nicely for coping with the synthetic image generation task (section 4.4), whilst we chose to rely on open-source AI development frameworks such as LangChain and LangGraph (n.d., n.d.) (section 4.3) for our core Agentic development needs. These decisions kept the system powerful and economically viable, making it accessible for replication in other academic and educational settings with similar constraints.

5. Functional Demonstration

In the previous section, we presented the architectural framework and a series of tools that can be integrated to educational software in order to transform static educational content into multimodal, engaging, and personalized learning experiences. To demonstrate how these tools, AI agents and agentic workflows can complete their tasks in practice, we developed a functional demonstration prototype. The prototype focuses on showcasing the backend agents and their intelligent decision-making processes operating in real time, supported by a lightweight frontend interface built with Vite and Tailwind. These capabilities are presented in full detail in the subsequent sections whereas at Figure 14 a visual depiction of the learner's dashboard screen.

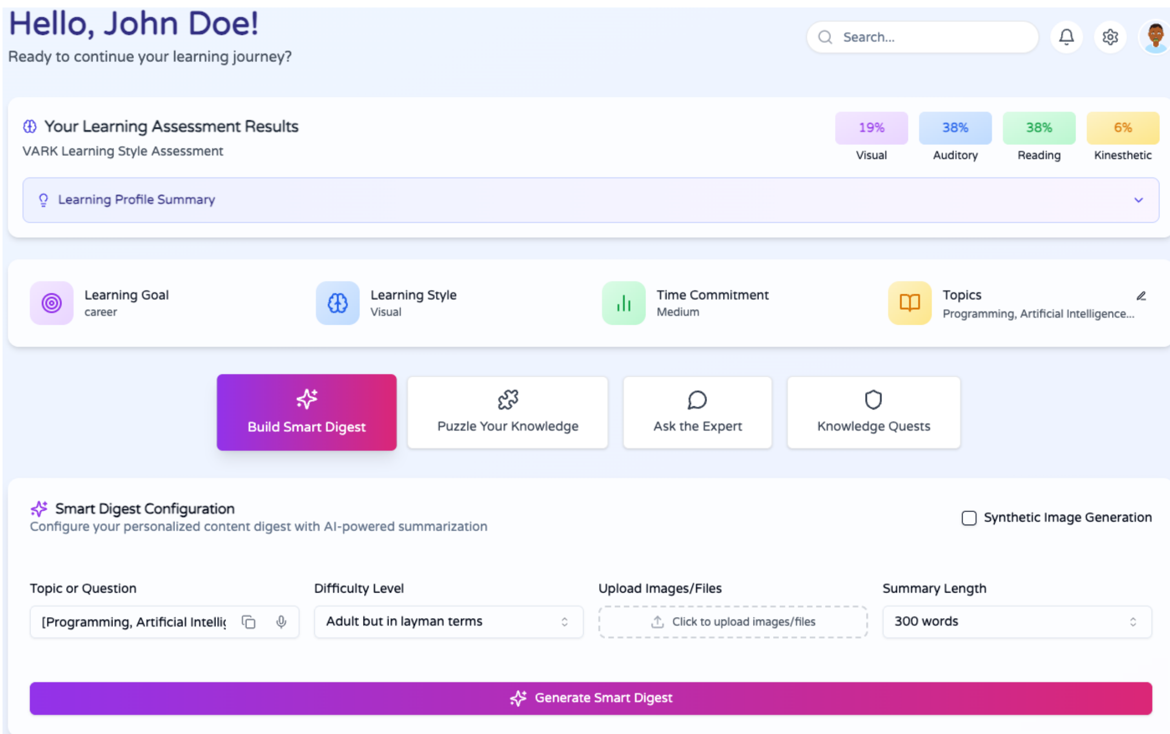


Figure 14: Screenshot of the learner's dashboard for the prototype UI that was built to disseminate the agents' capabilities in action. It enables the use to easily access their VARK assessment results and navigate and test the implemented agentic capabilities ("Smart Digest" agent, "Quiz generation" agent, "Enhanced QA agent, and "Quest generation" agent.

5.1 Assessing students' learning styles

As outlined in section 4, understanding how users prefer to learn is essential for receiving a personalized learning experience. Towards this direction and to capture the learning preferences of students, in section 4.3 we introduced the VARK assessment chain that can help identify the learners' dominant learning style(s) across the four modalities: visual, auditory, read/write, and kinesthetic.

In our prototype implementation, we developed a guided wizard that navigates the user through the VARK assessment questions and when all questionnaire answers have been received, it triggers the AI-powered VARK assessment tool that analyzes the results and generates a personalized learning profile summary, which includes the dominant learning styles, personalized recommendations based on cognitive patterns, and customized study strategies. Figure 15 and 16 below visually depict a sample of the structured multiple-choice questionnaire appearing through the wizard, as well as the GenAI produced analysis and recommendations from the VARK chain upon receiving the learner's answers.

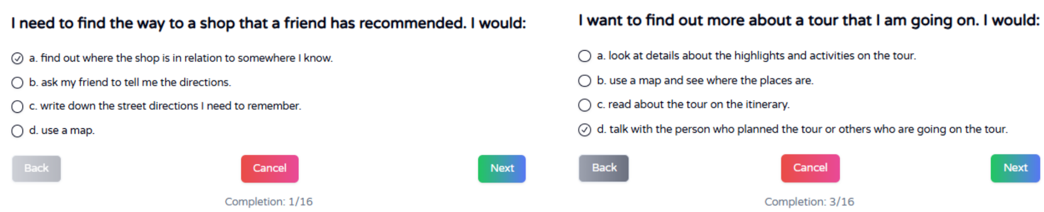


Figure 15: Graphical representation of the VARK questionnaire as presented to users. Four options are provided per question (in random ordering), corresponding to the four learning modalities (visual, auditory, read/write, kinesthetic).

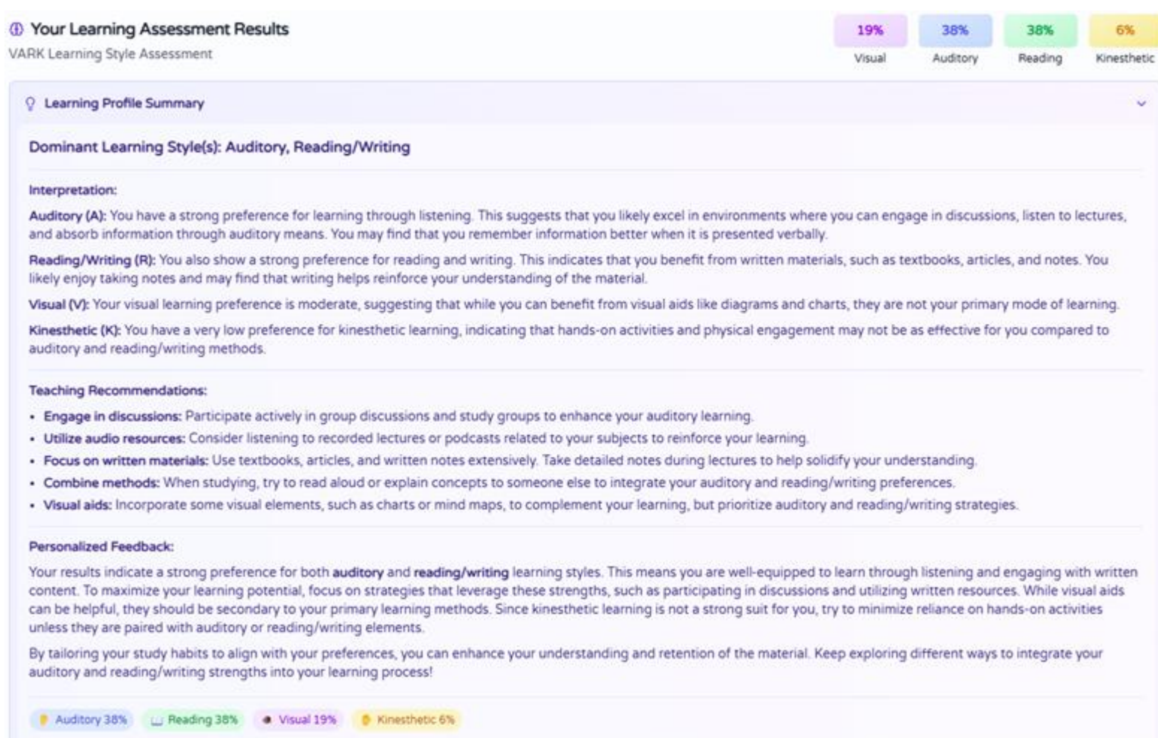


Figure 16: Visualization of AI-generated analysis including:(a) identification of dominant learning styles, (b) detailed interpretation of cognitive patterns, (c) personalized teaching recommendations, and (d) individualized feedback.

5.2 Enhanced QA Agent

Following Section 4.4, where we outlined the implementation details of the “Enhanced QA” agent and its capabilities, we hereby showcase its practical application via the "Ask the Expert" feature that was included in our prototype implementation.

The “Enhanced QA” agent enables users to pose open-ended questions and receive grounded responses through the RAG-based pipeline Sections 4.1 and 4.2, ensuring hence that the answers remain accurate and contextually appropriate. In the screenshot at Figure 17 for example, the user asked the agent "What are the differences between bubble sort and merge sort?”. The “Enhanced QA” agent leveraged the information contained in the knowledge base and provided a detailed response explaining both algorithms and their key differences. The agent also included explicit references to the sources that support its answer, maintaining transparency and factual accuracy.

Figure 17: Visual representation of the “Ask the Expert” interface leveraging the “Enhanced QA” agent.

Figure 18 also comes to demonstrate the agent’s transparency and answer reliability. In this example, the user is asking about a question that cannot be addressed by the agent’s available tools, and hence the agent fails to answer the question, yet transparently provides a detailed justification as per the reasons behind it.

Complementarily, the system's semantic search capabilities enable it to provide accurate responses even when the exact terminology used in the question does not appear in the knowledge base. This represents a significant advantage over systems that rely solely on text similarity search. In the example shown in Figure 19, the user asks "what is kalasthetics in Arts?". While the exact term "Kalasthetics" is not explicitly defined in the knowledge base, the agent performs semantic search and identifies that it is closely related to the concept of "Somaesthetics," a philosophical discipline introduced by Richard Shusterman (Shusterman, 1999). The system provides a comprehensive explanation of Somaesthetics, describing it as the critical and meliorative study of the experience and use of one's body as a locus of sensory-aesthetic appreciation. This example demonstrates how semantic understanding allows the agent to bridge conceptual gaps and provide meaningful answers even when exact keyword matches are not available.

Additionally, for visual learners, the system can retrieve and display relevant images from the educational materials and/or can also generate AI-powered synthetic images upon user selection. The image extraction and synthesis capabilities of the “Enhanced QA” agent are demonstrated at Figure 20.

Beyond text-based queries, users can attach diagrams or images they do not fully understand and ask related questions. The "Enhanced QA" agent selects the appropriate tools—such as image analysis—retrieves relevant information from the knowledge base and identifies how the provided image or diagram applies in the areas selected by the user (for example, business or design).

In the example shown in Figure 21, the user attaches a mathematical graph and asks "can you explain this graph please?". The system analyzes the quadratic function graph and provides a comprehensive explanation of its key features: the vertex, axis of symmetry, and intercepts. Beyond the mathematical explanation, the agent demonstrates practical applications across multiple domains selected by the user. It shows how the parabola's properties apply in design and arts for aesthetic purposes, in language education for explaining mathematical concepts, in business for profit maximization models, and in artificial intelligence for optimization problems. This approach helps learners connect theoretical knowledge with real-world usage, enabling them to "connect the dots" and making the learning process more engaging.

The "Enhanced QA" agent can also handle questions requiring mathematical analysis by leveraging specialized mathematical tools. In the example shown in Figure 22, the user poses a practical question: "I ordered a large pizza with a diameter of 28 cm, but they brought me three smaller ones with a diameter of 16 cm each. Did I eat less pizza than what I originally ordered?". The agent recognizes that this question requires mathematical computation and invokes the appropriate mathematical analysis tool. The system calculates the area of both the large pizza and the three smaller pizzas, compares them, and provides a clear answer that the user ate slightly less pizza with the three smaller pizzas than the large one originally ordered. The response includes detailed calculations showing the radius, area formulas, and final comparison, making the mathematical reasoning transparent and educational.

Lastly, the prototype was enhanced to also leverage Text-to-Speech (TTS) capabilities which enable the conversion of the agent’s textual responses (questions and answers) into audio narration to further satisfy the needs of learners with strong auditory learning preferences.

Finally, Figure 23 visually depicts all possible user interactions with this module in the form of an activity diagram.

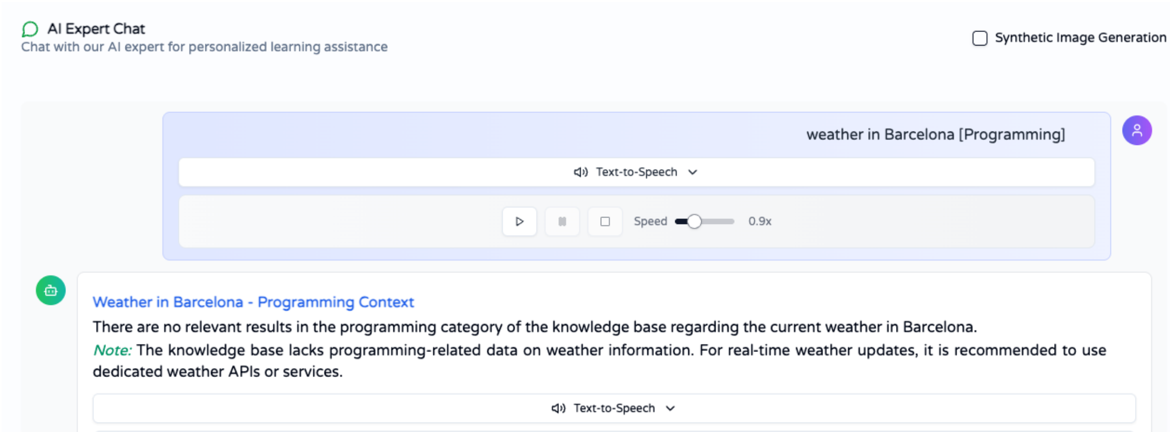


Figure 18: An example of a user question that cannot be addressed by the "Enhanced QA" agent due to lack of relative educational material in the knowledge base.

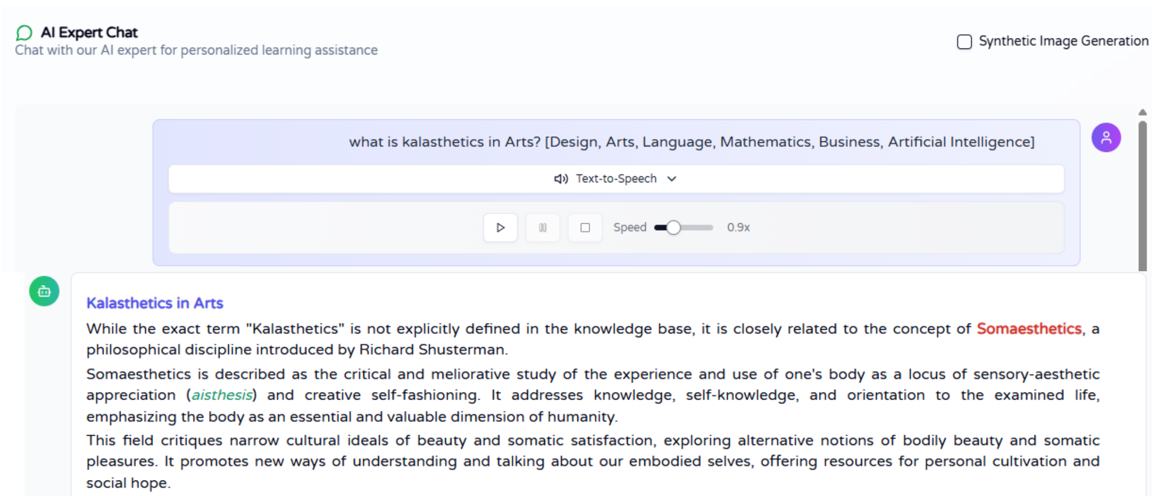


Figure 19: An example of semantic search capabilities where the "Enhanced QA" agent provides an accurate response by identifying conceptually related terms rather than relying on exact keyword matching.

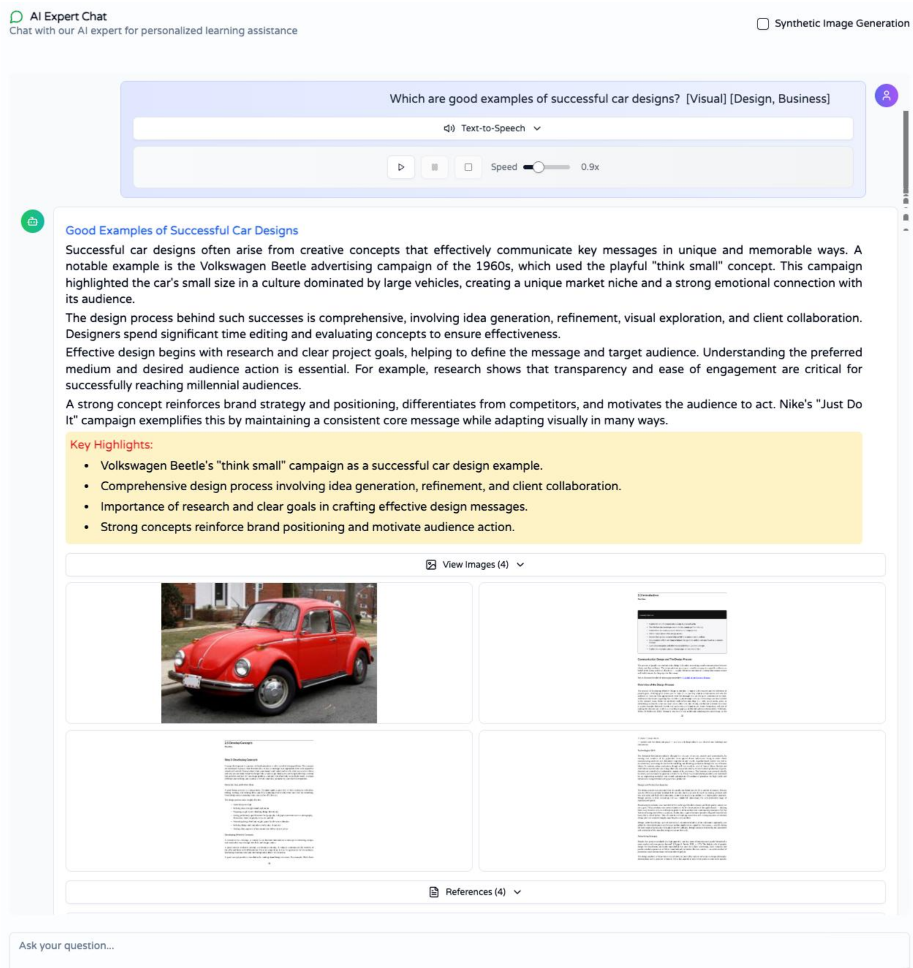


Figure 20: An example of image extraction and synthesis capabilities of the "Enhanced QA" agent to improve the learning experience of students with stronger visual preferences.

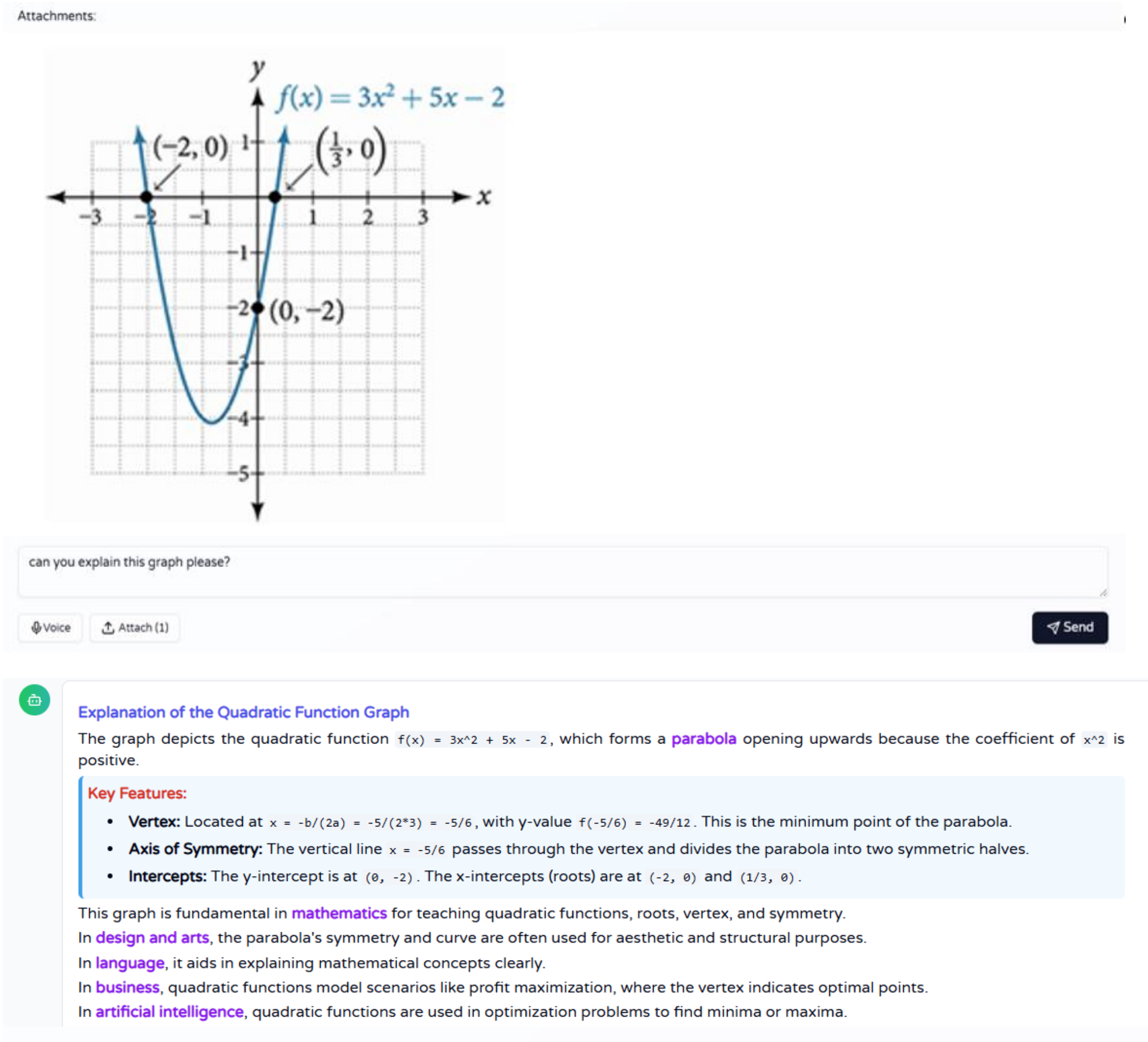
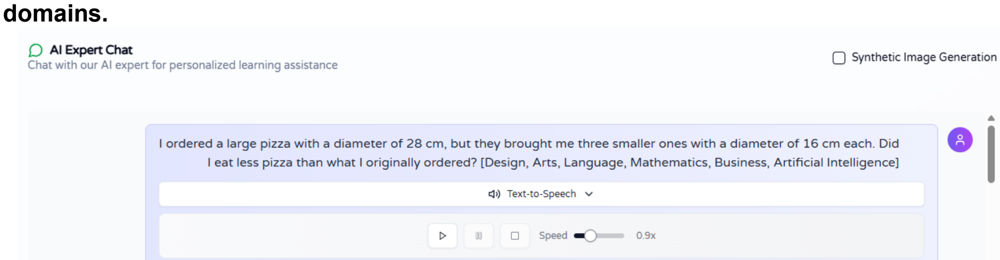


Figure 21: [Top image]: An example of an image input provided by the learner in the "Enhanced QA" and [bottom image] a demonstration of the image analysis capabilities of the "Enhanced QA" agent which explains the mathematical graph provided and demonstrates its practical applications across multiple domains.



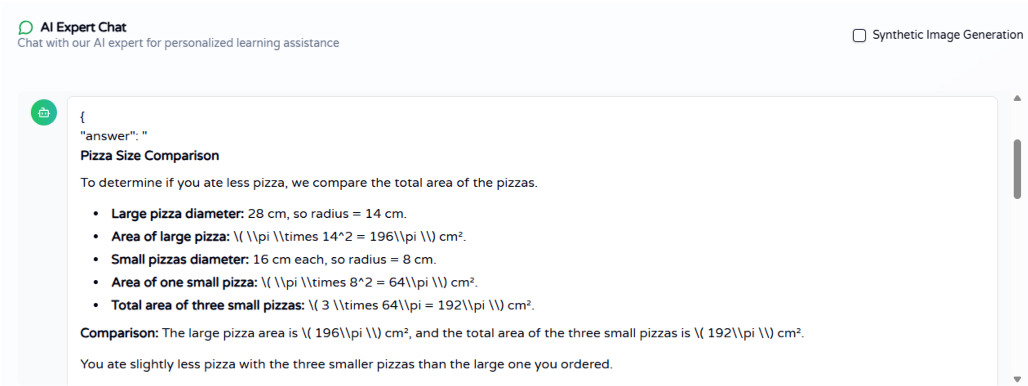


Figure 22: An example of mathematical analysis capabilities where the "Enhanced QA" agent solves a practical problem requiring area calculations and comparison.

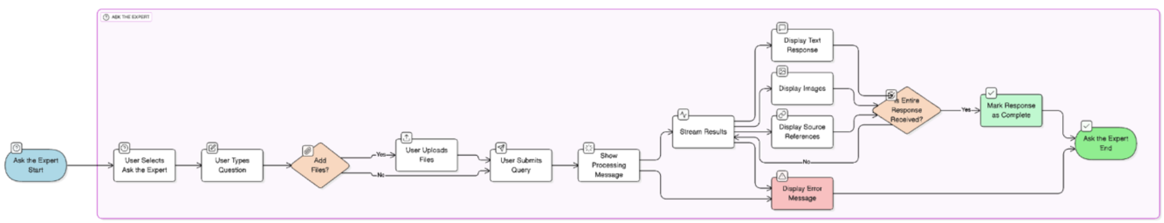


Figure 23: Activity diagram representing the possible user interactions with the “Enhanced QA” agent.

5.3 Smart Digest Agent

In the dissemination prototype a separate screen was developed to reflect the capabilities of the “Smart Digest” agent described over Section 4.5. This agent allows for the dynamic generation of summaries that leverage the educational material migrated to the knowledge base while accounting for the leaner’s defined settings (e.g. subject material, lengthy of the summary, difficulty level of the summary etc.). Figures 24, 25 and 26 display an example the “Smart Agent” agent in action, where it was requested to generate a short, simplified summary about “sorting algorithms and binary trees” with simultaneous synthetic image generation. The solution also provides access to the references utilized to compile the generated summary as well as the detailed learning plan (see section 4.5) that guided the retrieval steps of the agent.

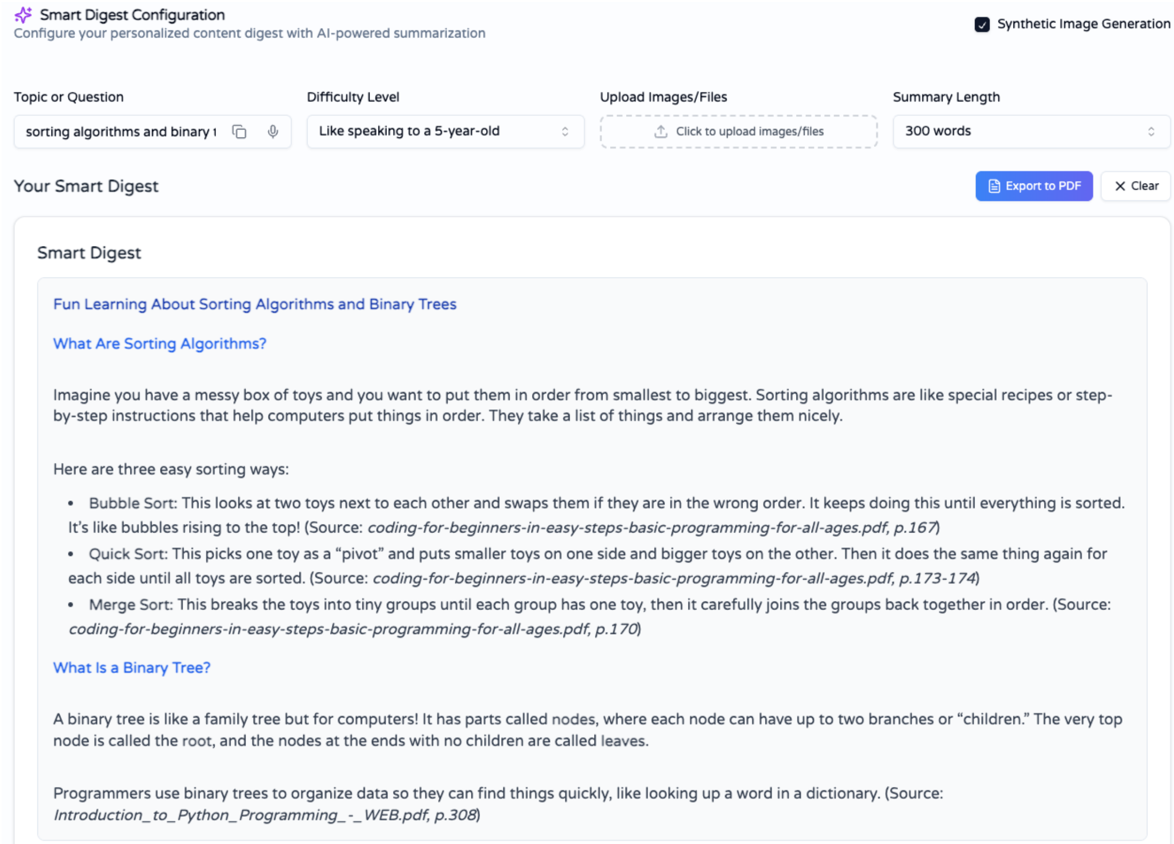
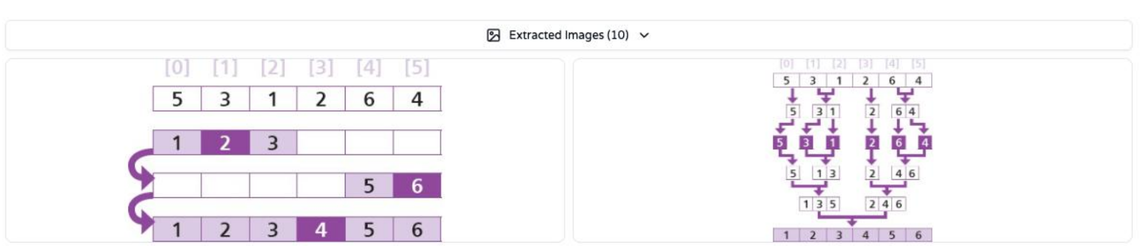


Figure 24: An example (only a part) of the summary generated by the "Smart Digest" agent. The agent provides a condensed (up to 300 words), simplified ("like speaking to a 5-year old") summary regarding sorting algorithms and binary trees.



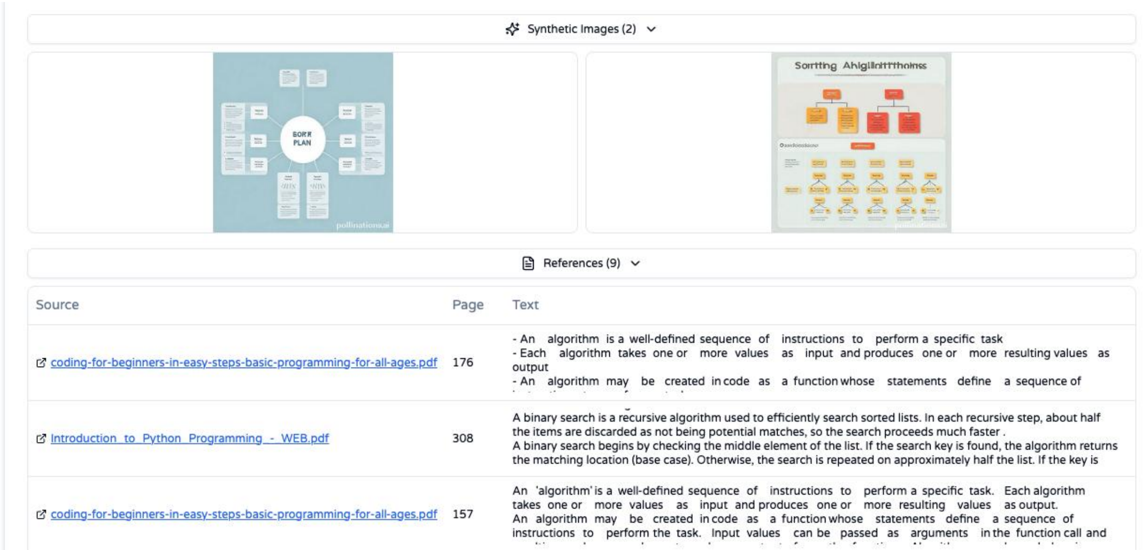


Figure 25: A sample of the extracted and AI-synthesized images as well as the references used to formulate the summary depicted at Figure 24.

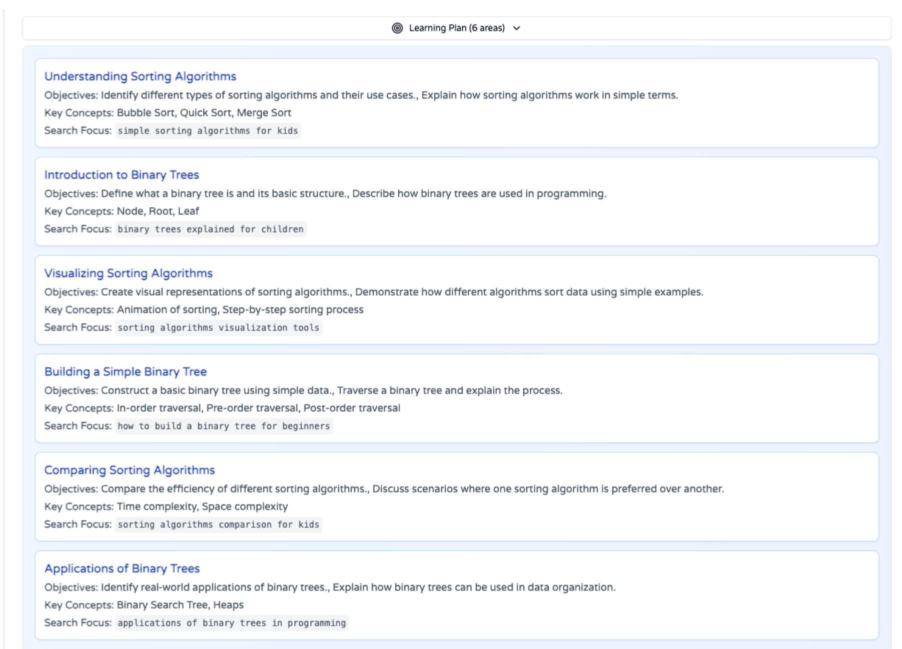


Figure 26: Learning plan generated by the "Smart Digest" agent for the generation of the summary depicted at Figure 24.

Similarly to the "Enhanced QA" agent, also in this scenario, TTS-based audio generated narration is supported for delivering an enhanced auditory experience to the learners. Lastly, Figure

27 graphically depicts the possible user interactions with the “Smart Digest” agent in the form of an activity diagram.

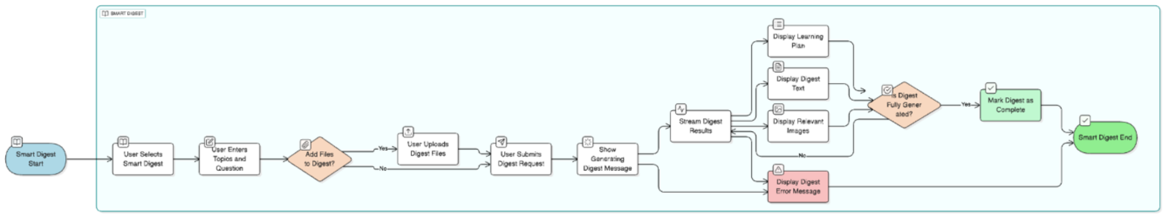


Figure 27: Activity diagram representing the possible user interactions with the “Smart Digest” agent.

5.4 Quiz Agent

As discussed at Section 4.6, the “Quiz generation” agent enables the learner to experience an interactive learning journey where the educational content is being provided in the form of multiple-choice based questions of varying difficulty level. To test the capabilities of this agent in action, we extended the prototype with an additional panel that hosts the quiz generation and execution. Figure 28 demonstrates the implemented interface to facilitate the employment of the “Quiz generation” agent, whereas in Figures 29 and 30, examples of the AI generated feedback in the cases of correct and mistaken user replies are provided.

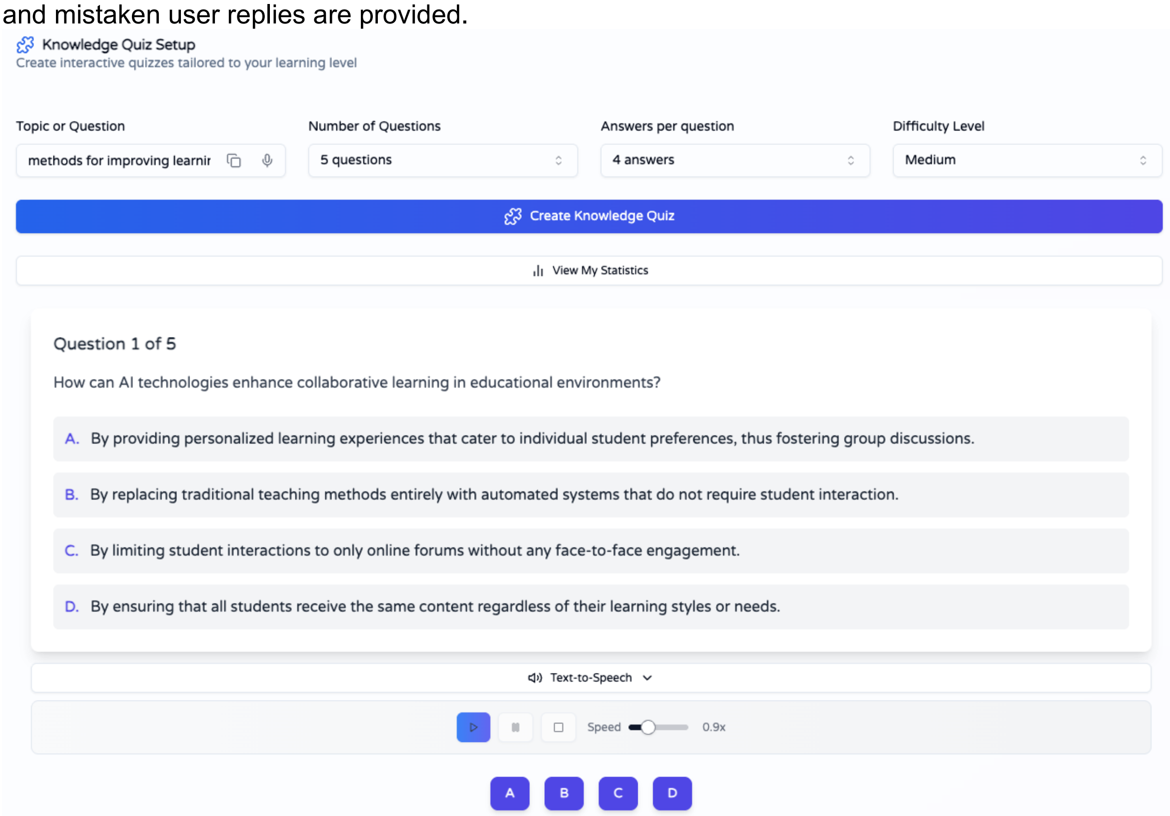


Figure 28: [Top image]: Visual representation of the user interface options for setting up a new Quiz around "methods for improving learners' experience" and [bottom image] preview of a sample of the quiz questions generated by the agent.

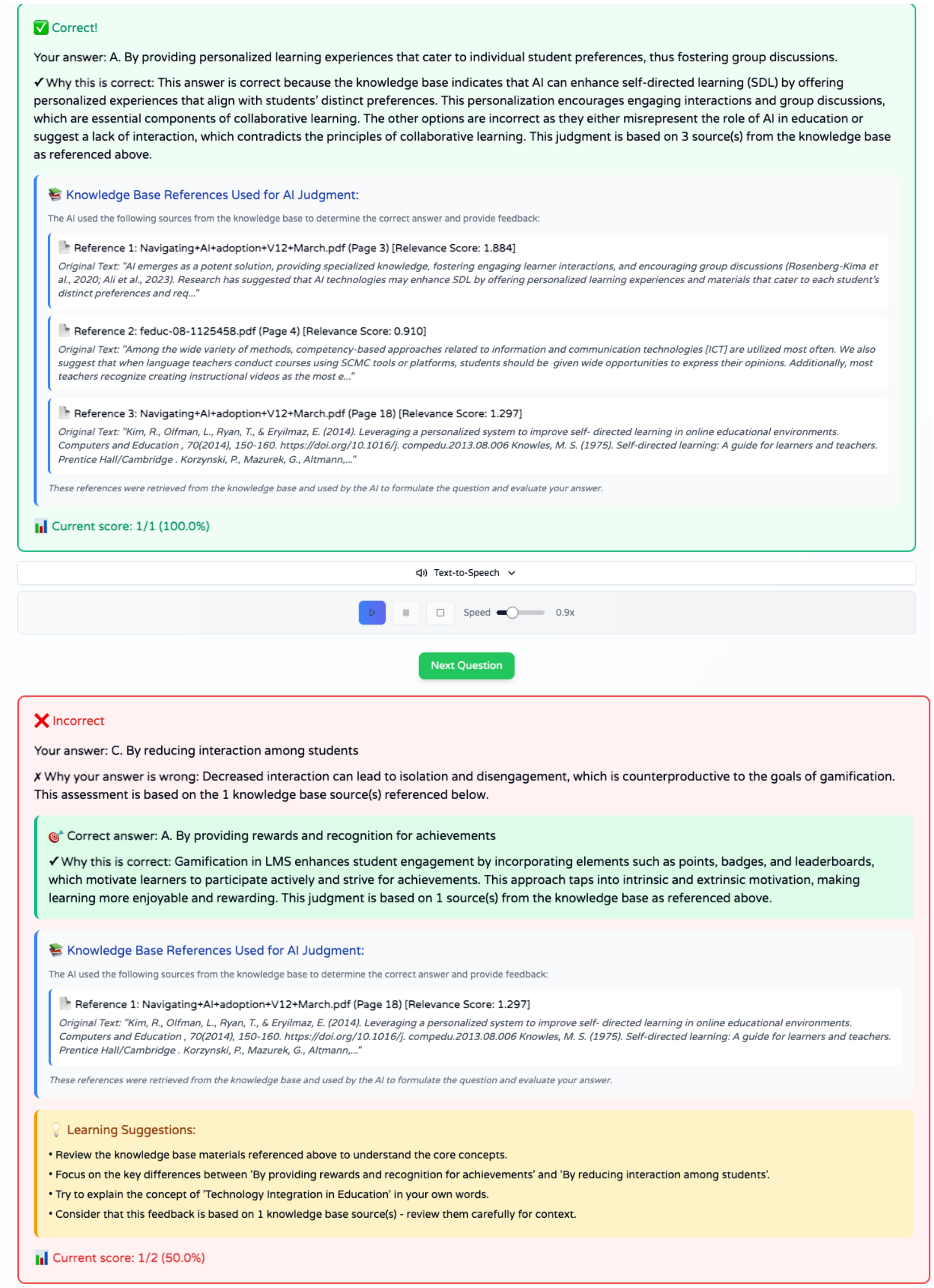


Figure 29: Example of the agent's feedback to a correctly (top image) and incorrectly (bottom image) answered quiz question.

As described at section 4.6, the “Quiz generation” agent supports the logging of the learner’s performance statistics collected during the practice assessments. Upon completion of a quiz, the student receives a summary of their performance (Figure 30) or can choose to preview their performance statistics over time (please refer to Figure 11 for an example of the Quizzes performance overview dashboard). Figure 31 shows the possible user interactions with the “Quiz Generation” agent in the form of an activity diagram.

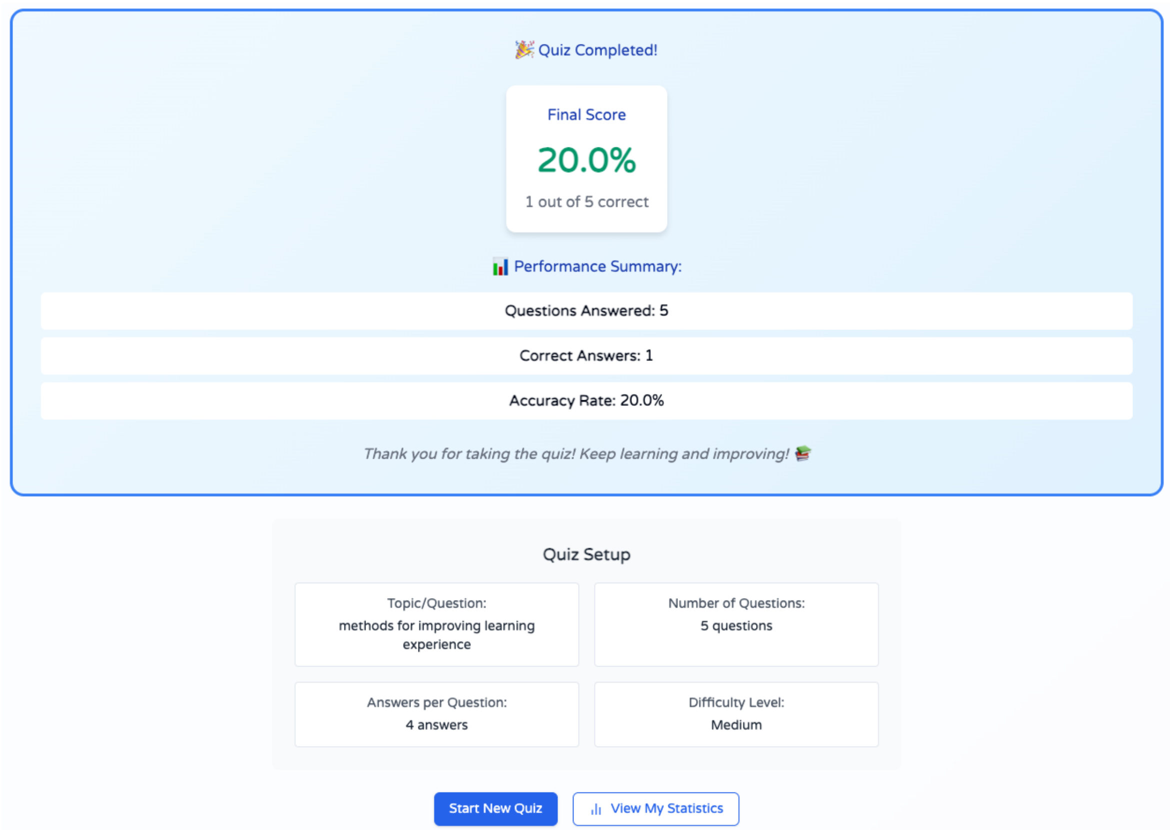


Figure 30: Results summary after the learner has completed a quiz assessment.

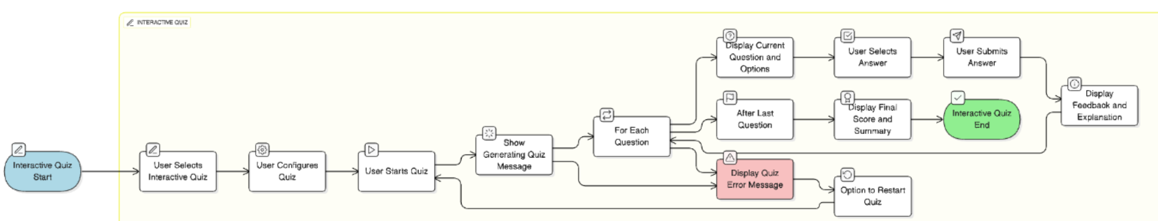
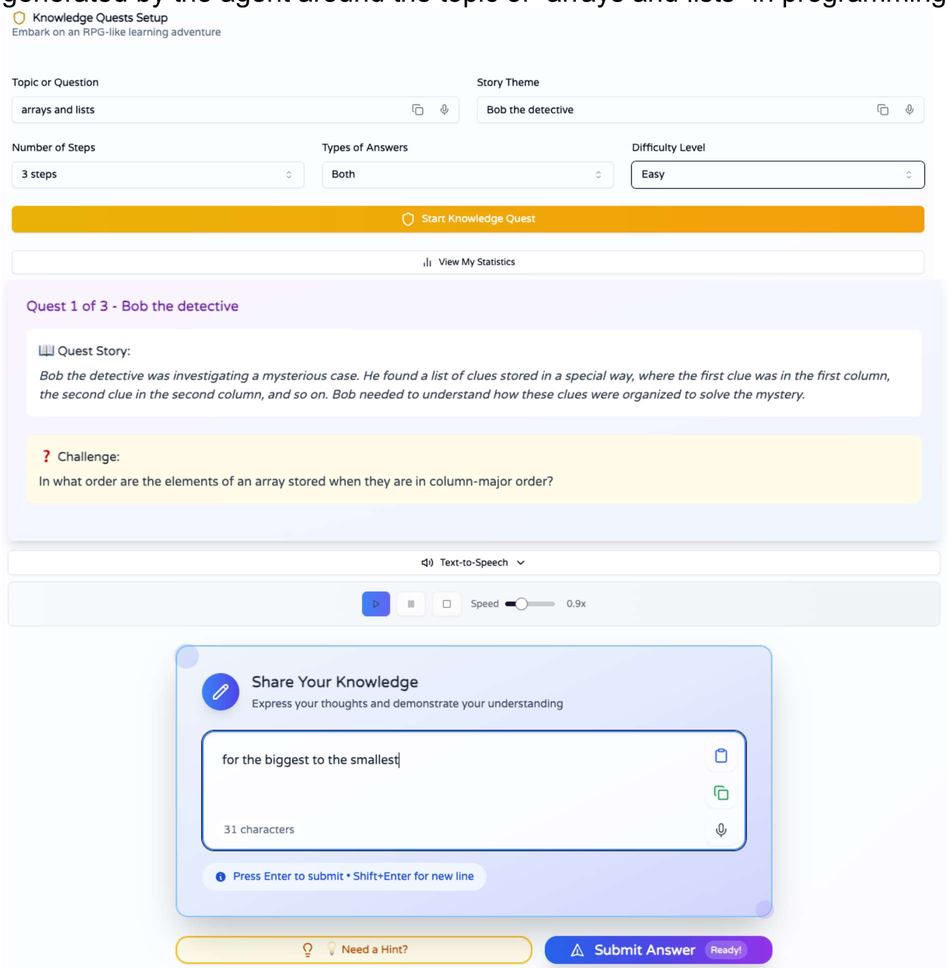


Figure 31: Activity diagram with the possible user interactions with the “Quiz generation” agent.

5.5 Quest Agent

The last agentic capability that was described at Section 4.7 enables the creation of interactive quests allowing the user to learn through a gamified learning experience. Similarly to the "Quiz generation" agent, it retrieves learning content from the encoded knowledge base and then generates a story around a user-defined theme, via which the learner may assess their understanding on the study subjects. Figure 32 displays an example of such a quest as generated by the agent around the topic of "arrays and lists" in programming.



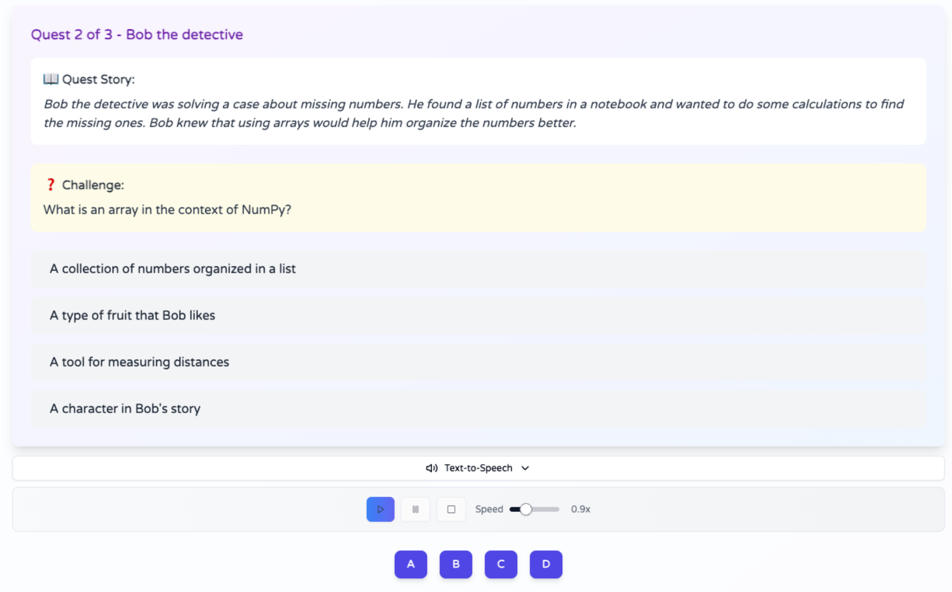


Figure 31: Example of a quest setup (top image) and generated open-ended (middle image) and multiple choice (bottom image) questions of the "Quest generation" agent when asked to generate a story about the "arrays and lists" using a "detective" theme.

Similarly to the "Quiz generation" agent, the "Quest" agent also can provide comprehensive feedback to the user for the correct or mistaken answers (Figure 33) and logs the performance statistics across the assessment tests to formulate a comprehensive summary. Like in the case of the previously described agents, the "Quest generation" agent also supports AI-driven narrations for enhanced auditory experience, however in contrast to the remaining agents, the Quest agent incorporates instructions to leverage a "storyteller" style with tailored tone and personality, to deliver a richer gamified learning experience. Figure 34 shows the possible user interactions with the "Quiz Generation" agent in the form of an activity diagram.

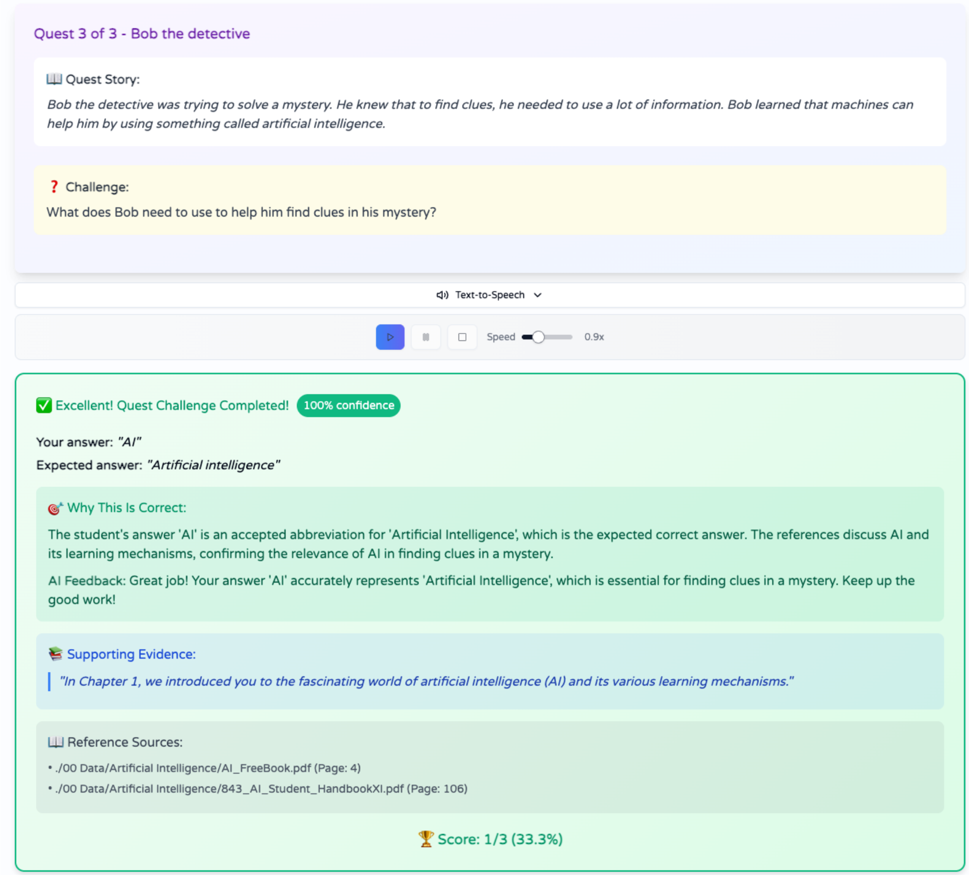


Figure 32: Example of the feedback provided by the “Quest” agent to the learner’s answer to an open-ended question. The agent leverages reasoning to properly decide on the correctness of the answer provided.

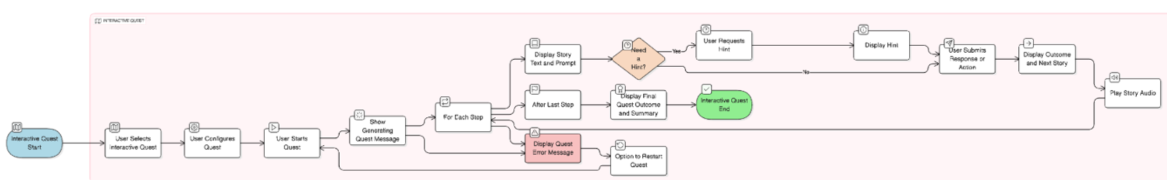


Figure 33: Activity diagram with the possible user interactions with the “Quest generation” agent.

6.User Guide

To provide clarity on the prototype's core functionalities, this section includes a comprehensive overview of its available features. Each modality is described in terms of how users engage with it and how it supports their personalized learning experience.

VARK: Learning Style Assessment
The VARK Assessment modality enables users to identify their personal learning preferences through a structured questionnaire based on the VARK model (Visual, Auditory, Read/Write, and Kinesthetic). Following initiation of the assessment, users navigate through 16 situational questions, each presenting a real-world learning scenario (see Figure 34). The interface displays one question at a time with four response options, allowing users to select the approach that best reflects their natural learning tendencies. Navigation controls enable users to move forward, return to previous questions, or exit the assessment at any point. A progress indicator tracks completion status throughout the assessment process.

Upon completion, users receive an immediate, comprehensive learning profile generated through AI analysis. The results interface presents scores across all four modalities with visual indicators, identifying the user's dominant learning style(s) in ranked order (see Figure 35). The system provides detailed interpretations of what each score means for the user's learning approach, along with personalized recommendations and customized study strategies tailored to their cognitive patterns.

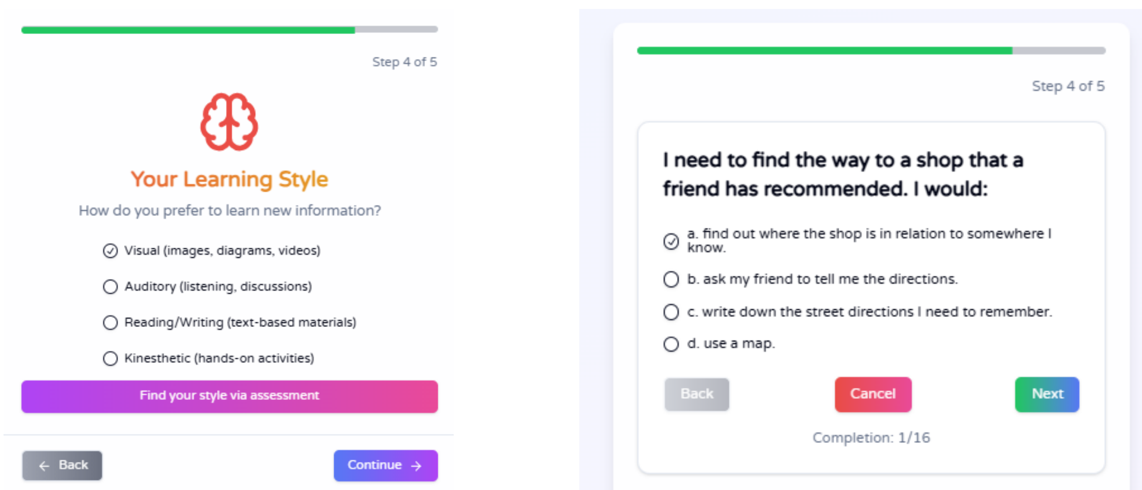


Figure 34: VARK assessment user interface illustrating two key components: the introductory screen explaining the four learning modalities and a sample question interface demonstrating the assessment format with response options and navigation controls.

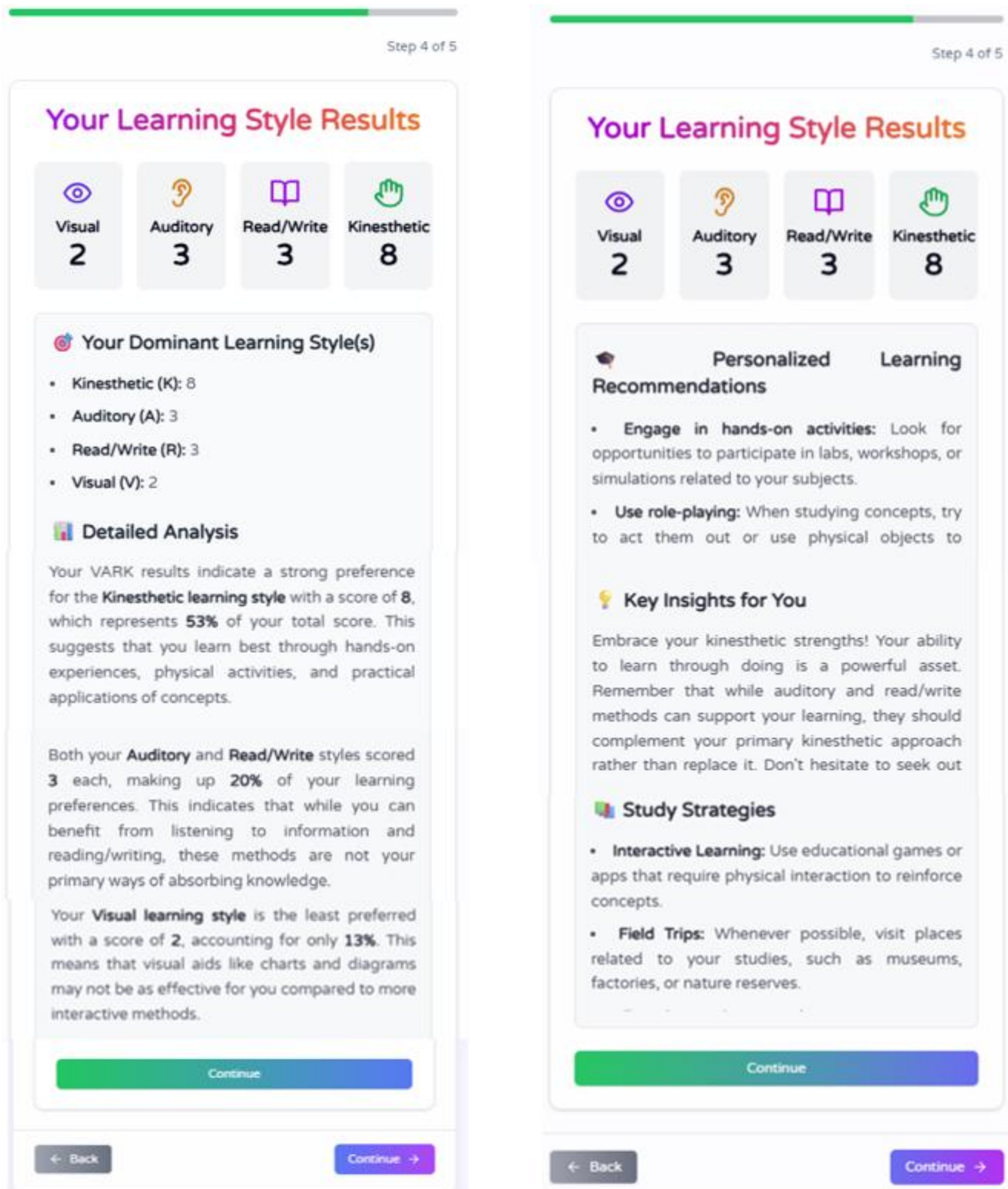


Figure 35: VARK assessment results interface presenting the complete learning profile, including score visualization across four modalities, detailed analysis of the dominant learning style, personalized learning recommendations, and tailored study strategies.

Smart Digest: AI-Powered Learning Summaries

The Smart Digest modality provides an interface through which learners interact with the AI-powered summarization capabilities described in Section 3.5. The configuration screen (Figure 36) presents users with input fields and controls for specifying their learning requirements. The topic or question field accepts both typed and voice input through an integrated microphone button, while a file upload area enables users to supplement the knowledge base with additional documents or images relevant to their study needs.

Three primary configuration controls shape the generated output. The difficulty level dropdown offers three distinct modes of explanation tailored to different expertise levels, the summary length selector (Figure 37) provides granular control over content volume ranging from brief 100-word overviews to comprehensive 4,000-word analyses, and the synthetic image generation toggle allows users to request AI-created visual representations alongside text-based content. This configuration approach enables learners to adapt the summarization process to their specific context, whether preparing for examinations under time pressure, conducting initial exploration of unfamiliar topics, or seeking comprehensive review of complex subject matter.

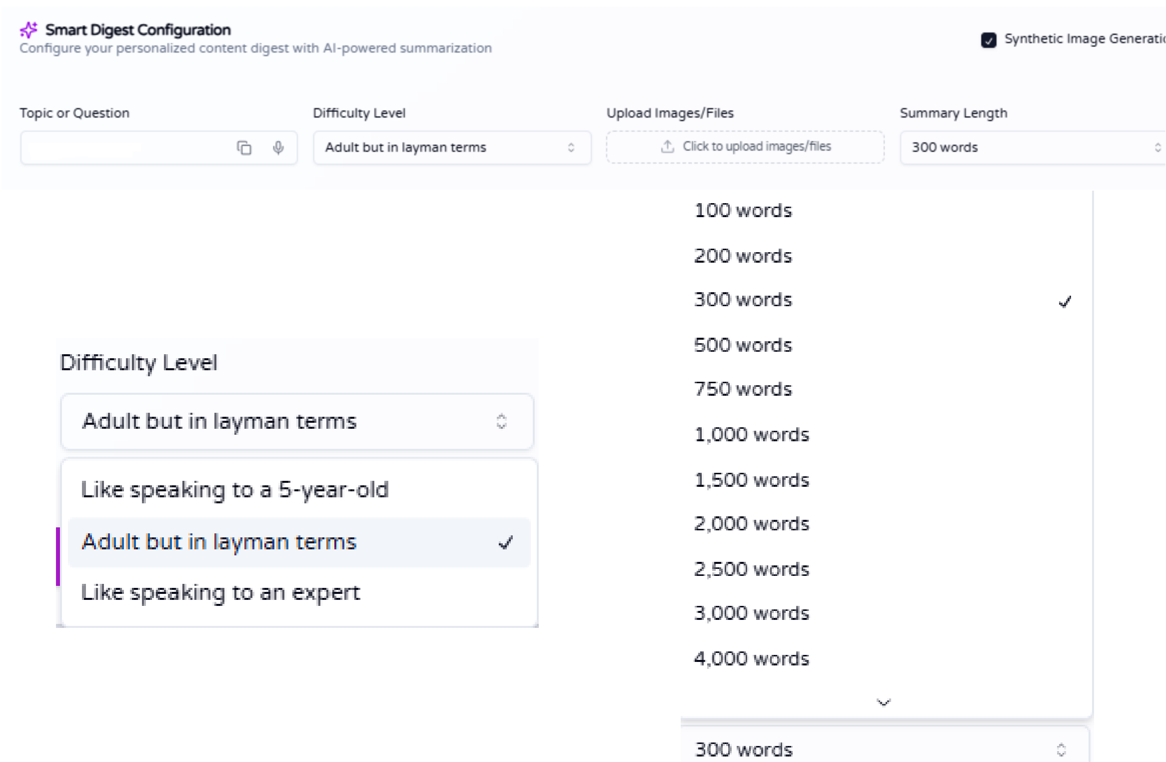


Figure 36: Smart Digest configuration interface displaying difficulty level and summary length selection options.

Puzzle Your Knowledge: Personalized Quiz Assessments

The Quiz modality enables learners to generate and complete personalized knowledge assessments based on the AI-powered capabilities described in Section 4.6. The configuration screen (Figure 37) presents users with input fields and controls for specifying their assessment requirements. The topic or question field accepts both typed and voice input through an integrated microphone button, allowing users to define the subject matter they wish to be tested on. Four primary configuration controls shape the generated quiz. The number of questions dropdown (Figure 38) offers options ranging from 1 to 20 questions, enabling users to create brief knowledge checks or comprehensive assessments. The answers per question selector (Figure 37) provides choices between 2 and 6 multiple-choice options for each question, allowing users to adjust the complexity of decision-making required. The difficulty level dropdown (Figure 37) presents three distinct modes (Easy, Medium, Hard) that determine the sophistication of both questions and incorrect answer options. This configuration approach enables learners to tailor the assessment experience to their specific learning context, whether conducting quick self-checks, preparing for examinations, or challenging themselves with advanced material.

During quiz execution, questions appear sequentially with clearly labeled answer choices (A, B, C, D, etc.) as shown in Figure 28 (bottom image). Users select their response by clicking the corresponding button, with text-to-speech functionality available through playback controls that include adjustable speed settings. Upon submitting each answer, the system generates immediate, comprehensive feedback grounded in the knowledge base. For correct responses (Figure 29, top image), users receive confirmation with detailed explanations of why their selection aligns with the source material, including knowledge base references with document names, page numbers, and relevance scores. For incorrect responses (Figure 29, bottom image), the feedback explains why the chosen answer was erroneous, serves the correct option, and provides reasoning supported by specific text excerpts from the source materials. Incorrect answers trigger additional personalized learning suggestions that guide users towards concepts requiring further review.

The current score displays continuously as both a fraction and percentage. Upon completing all questions, users receive a summary (Figure 30) showing their final score, total questions answered, correct answers, and accuracy rate. From this completion screen, users can initiate a new quiz or access the Quiz Statistics Dashboard (Figure 11), which provides performance analytics including trend charts, topic-specific accuracy breakdowns, difficulty analysis, AI-generated insights about strengths and weaknesses, and error pattern identification with the ability to filter results by date range or export statistics as PDF reports.

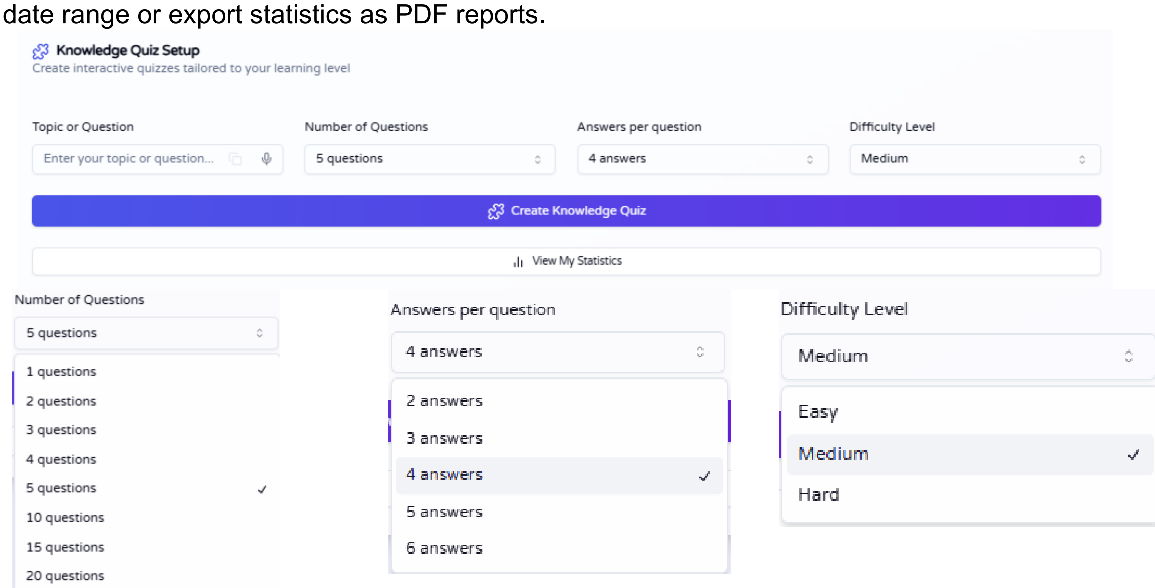


Figure 37: Knowledge Quiz Setup interface showing configuration options for quiz parameters: question count, answer options per question, and difficulty level selection.

Ask the Expert: Interactive AI Learning Assistant

The Ask the Expert modality provides a conversational interface where learners can ask questions and receive answers grounded in their course materials. The main screen (Figure 38) displays a chat area where the conversation unfolds, with a text input field at the bottom for typing questions. Users can formulate questions in their own words without needing specific phrasing or keywords. The interface includes a microphone button for speaking questions aloud and an attachment button for adding files or images that relate to their inquiry. When users want the system to explain a diagram, analyze a chart, or work with visual content from their own files, they can upload these materials directly through the attachment feature. The question appears in the chat area once submitted, and the system begins processing the request.

The AI agent's response appears below the question with a structured layout that presents the answer followed by supporting materials. When relevant visual content exists in the course materials, thumbnail images appear within the response, which users can click to view in full size (Figure 20). Below the main answer, expandable source sections show where the information came from, displaying excerpts from the actual course documents. Users can click these source references to examine the original context. If a question involves mathematical calculations, the agent works through the problem step-by-step and shows the solution process. When users ask about content that doesn't exist in the uploaded course materials, the agent clearly states it cannot answer and explains what information is missing (Figure 18), preventing confusion about whether the response is reliable. Users can continue asking related questions, with each new inquiry building on the previous conversation. The agent remembers what was discussed earlier in the session, so users can ask follow-up questions like "Can you explain that differently?" or "What about the second point you mentioned?" without repeating context. This back-and-forth exchange continues until users find the clarity they need or choose to start a fresh conversation.

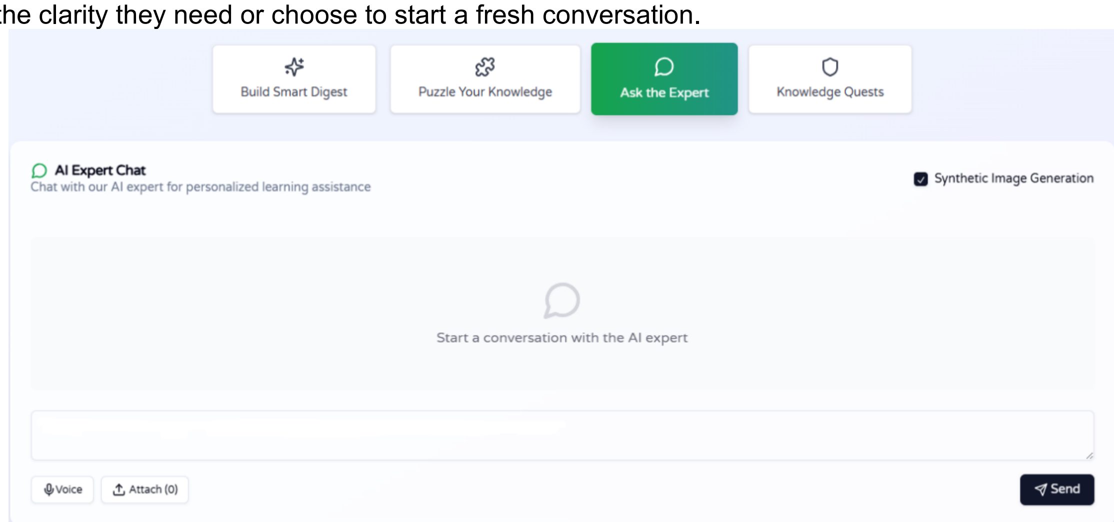


Figure 38: Initial screen of the Ask the Expert modality prompting users to begin their interaction with the AI learning assistant through multiple input methods.

Knowledge Quests: Gamified Learning Through Interactive Storytelling

The Knowledge Quests modality transforms educational content into an RPG-style learning adventure that combines assessment with narrative engagement. The user begins by configuring the quest through the setup interface (Figure 39), where the learning topic is specified and a story theme is defined that will frame the learning journey, for example, "Dragons and Magicians" or "Detective

Mystery." The interface allows customization of the quest length by selecting between 1 and 10 steps, choosing the preferred answer format (text answering, multiple choice, or both), and setting the difficulty level (Easy, Medium, or Hard) to match the current knowledge level. Like other modalities in the system, voice input is available for both fields, enabling hands-free configuration. Once "Start Knowledge Quest" is clicked, the system retrieves relevant content from the uploaded educational materials through the RAG pipeline and generates a coherent narrative that weaves assessment questions into the storyline, creating an experience where learning feels like progressing through an adventure rather than taking a traditional test.

During the quest, each step presents a "Quest Story" section that advances the narrative, followed by a "Challenge" section containing the actual assessment question. For multiple-choice questions (Figure 32, bottom), the user selects from labeled options (A, B, C, D) displayed as interactive buttons at the bottom of the screen. For open-ended questions (Figure 32, middle), the response is typed in the "Share Your Knowledge" text area, which encourages expression of reasoning and demonstration of understanding beyond simple answers. When difficulty is encountered, clicking "Need a Hint?" generates contextual AI-powered guidance that nudges toward the solution without revealing the answer directly, the hint appears in a highlighted box marked "AI Generated." After submitting an answer, the system provides comprehensive feedback (Figure 33) that includes the user's response, the correct answer, a "Why this is incorrect" explanation detailing the conceptual misunderstanding, AI-generated feedback offering guidance for improvement, and a "Learning Opportunity" section that clarifies the correct approach. The system also displays reference sources citing specific pages from the uploaded materials and shows a confidence score (e.g. "80% confidence") indicating how certain the AI is about its evaluation.

Similar to other modalities, all quest content, stories, questions, hints, and feedback, can be converted to speech using text-to-speech controls with adjustable playback speed. What distinguishes the Quest modality is that the AI narration adopts a storyteller style with tailored tone and personality, delivering a richer gamified learning experience that enhances engagement and accessibility.

Upon completing all quest steps, a "Quest Completed!" summary screen is displayed showing the final score (as a percentage), the number of correct answers out of total questions attempted, and a "Quest Performance Summary" that includes the story theme used, total quests completed, successful challenges, and success rate. Below this summary, the quest configuration settings can be reviewed and the user can choose to either "Start New Quest" to begin another learning adventure or click "View My Statistics" to access the comprehensive Quest Statistics Dashboard for deeper performance analysis and AI-driven insights into learning patterns across multiple quests. An example of such a performance analysis review is provided at Figure 11.

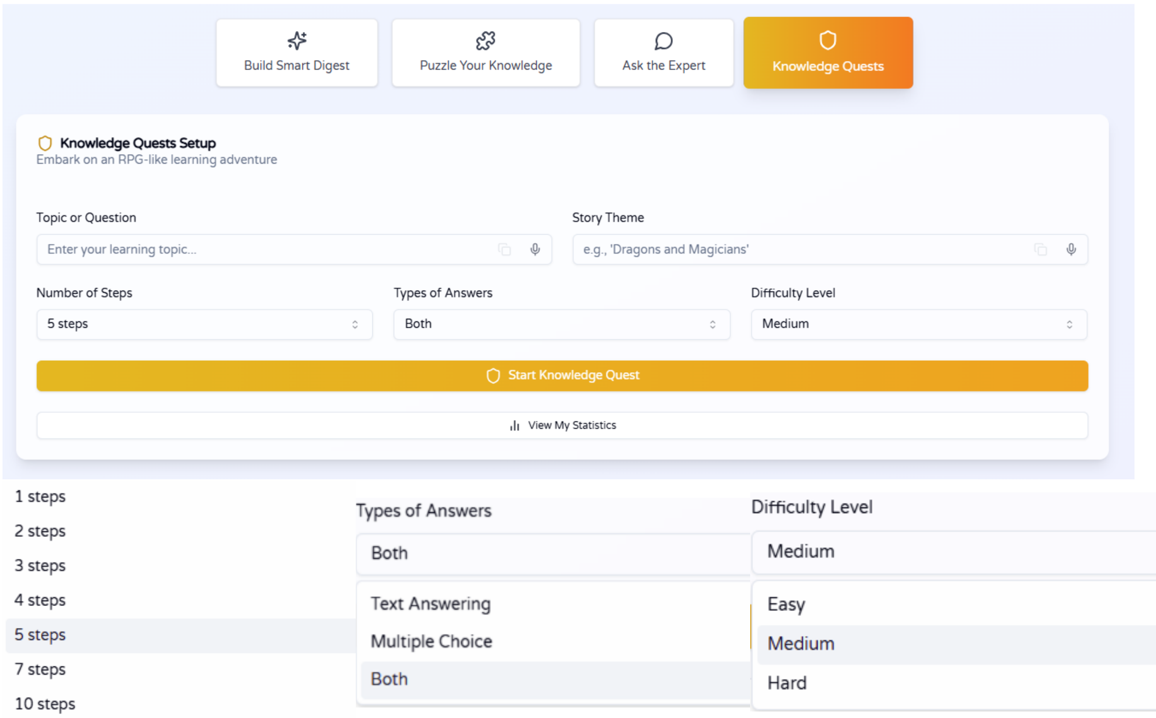


Figure 39: Configuration interface for the Knowledge Quests modality with customizable parameters for quest length, answer types, and difficulty levels.

7.Conclusions and Future Perspectives

In the present study, we investigated two fundamental research questions: how effectively generative AI can support personalized learning for both students and educators, and in what ways agentic AI architectures can enable the conversion of static educational content into engaging, interactive learning experiences.

The motivation for this research stems from the breakthroughs generative AI has introduced across all aspects of everyday life and has fundamentally affected how people consume information. Students, especially of younger generations, increasingly prefer digital, interactive formats over traditional materials. Yet time bottlenecks hinder educators from updating their curriculum to meet these evolving needs. Current educational AI solutions come to bridge for this vacuum but may be challenged by inherent customization and integration limitations, resulting in a fragmented landscape that may block quick adoption in the standard procedures of educational institutions.

To address these challenges, we developed a series of AI-based tools that dynamically adjust the learning content into different formats, meeting the learners' individual learning styles and preferences. The proposed solution benefits from the use of Large Language Models (LLM) combined with Retrieval Augmented Generation (RAG) techniques, exposed through AI agents and workflows that extract relevant curriculum data and generate grounded, multimodal responses. In particular, we implemented four different agentic architectures, each showcasing specific AI capabilities: identifying learning styles and giving tailored recommendations, generating content in multiple formats through parallel processing, creating quests and quizzes with adjustable difficulty and enabling conversational Q&A with feedback mechanisms and able to manage conversations while automatically picking the right tools. The proposition combines several AI features, including but not limited to text-to-speech synthesis, suitable for students with stronger auditory learning preferences, image extraction and generation, aiming to cover for the needs of students preferring concepts to be explained visually, AI-generated tailored study plans to support for automated summaries on user select learning subjects, and mechanisms for tracking progress with personalized suggestions for improvement. Additionally, we demonstrated the power that these tools can bring to the educational process via a quick prototype that utilizes these agents in practice.

Within the course of this study, we provided several hands-on ways by which generative AI with agent-based architectures can indeed support and enhance the learning process, e.g. by creating summaries that save time, generating quizzes that reinforce knowledge, making personalized study plans, giving immediate feedback, providing expert help with visual aids, and creating gamified experiences. We aspire that this research will set the foundations for designing the next generation of educational solutions that are fully versatile, customizable, scalable and easy to integrate, and hence provide a revolutionized and fully personalized learning experience for students and educators.

7.1 Future Perspectives

This study sets the ground across several directions, interesting to be further explored. The current prototype evolved through two development phases, each with different technical approaches. In our preliminary implementation attempt we relied on using the Gradio open-source library that enables rapid interface prototyping. This allowed us to quickly test the core AI functionalities. For the second version, we wanted to create a more appealing and engaging experience for students, so we completely redesigned the solution, implementing the latest innovations in the Generative AI space (in terms of backend enhancements) as well as updated the user interface (frontend enhancements) by using Node.js with Vite and Tailwind CSS. Additionally, we implemented logging mechanisms, unlocking capabilities to store user credentials and keep track of the student learning progress over time with the simultaneous provision of powerful AI-driven insights.

However, although this prototype serves to meet the posed research questions and objectives, several additional aspects need to be explored and considered to pave the road for productization of this solution. Specifically, important aspects related to the user and data tracking outlined at Section 2 are still subject to further improvement. These include the implementation of enterprise-grade databases, strong security measures to protect student data and proprietary materials, and scalable infrastructure to serve thousands of concurrent users, while at the same time accounting for compliance with international regulations and directives on personal data and ethical use of AI. Moving from the present prototype to a production-ready system involves tackling these concerns and addressing deployment challenges, optimizing performance, and building robust, scalable infrastructure.

Secondly, exploring improved ways to enhance the retrieval accuracy of the RAG-based pipeline also constitutes and interesting future research direction. In the present implementation we are employing the current state-of-the art ways for document processing, chunking and information encoding however there is still plenty of room for improvement. For example, adding Hierarchical Navigable Small World (HNSW) indexing would make retrieval both faster and more accurate (Malkov & Yashunin, 2018). Last, also knowledge graphs (KG) could be employed in the process to more accurately capture how concepts relate to each other (Pan et al., 2024). Research on domain-specific knowledge graph construction demonstrates that combining Named Entity Recognition (NER) with graph-based representations can reveal knowledge gaps and optimize content organization for improved learning outcomes (Panagoulias et al., 2024). Recent work on knowledge space reduction using graph-based representations shows that structuring domain knowledge as interconnected nodes and relationships enables more precise information retrieval by reducing the dimensional space from which AI systems extract responses (Panagoulias et al., 2025). This approach, termed Sequential Language Model Integration, demonstrates that feeding structured KG into LLM pipelines significantly improves accuracy by constraining the model's reasoning to relevant conceptual pathways.

Integrating knowledge graphs into the RAG pipeline could enable the system to understand not just individual concepts, but the broader conceptual framework of a subject, supporting more intelligent content sequencing and personalized learning path generation. An equally interesting perspective lies in augmenting and strengthening the ways complex document elements (e.g. tables, diagrams, videos, and image-heavy pages) are handled. Current document parsing may struggle to process such cases, and better extraction techniques with native multi-lingual support would provide the agentic mesh with higher quality content to work with.

Comparable, it is interesting to investigate how other types of multimedia input and extend the user’s learning experience and overall performance of the solution. Future versions could handle PowerPoint presentations, video content, or even databases and web scrappers and education material sources.

Finally, and complete the circle of retrieval-based improvements, a perspective for enhanced ways to evaluate educational AI systems would also be an interesting direction to explore. Libraries like LangTest, Ragas, and DeepEval can evaluate LLM applications (Purwar, 2024), but they need annotated datasets with correct answers already prepared. Developing such datasets for educational purposes and on top of educators’ selected learning material takes a lot of time and expert work. Future research could contribute in extending present methods for systematically evaluating retrieval accuracy (how well the system finds relevant information), content quality (whether generated materials are correct and pedagogically sound), and learning effectiveness (whether students actually learn better) in an easier and more flexible way. This would enable a more rigorous assessment on whether AI-generated educational content is fully and exclusively being used in the context for the AI-powered solutions.

8. Bibliography

Akçayır, M., & Akçayır, G. (2017). Advantages and challenges associated with augmented reality for education: A systematic review of the literature. Educational Research Review, 20, 1–11. https://doi.org/10.1016/j.edurev.2016.11.002

Alepis, E., Stathopoulou, I.-O., Virvou, M., Tsihrintzis, G., & Kabassi, K. (2010). Audio-lingual and visual-facial emotion recognition: Towards a bimodal interaction system. Proceedings of the IEEE International Conference on Tools with Artificial Intelligence, 274–281. https://doi.org/10.1109/ICTAI.2010.111

Alepis, E., & Virvou, M. (2014). Object-Oriented User Interfaces for Personalized Mobile Learning. Springer. https://doi.org/10.1007/978-3-642-53851-3

Anderson, T., & Dron, J. (2011). Three generations of distance education pedagogy. The International Review of Research in Open and Distributed Learning, 12(3), 80–97. https://doi.org/10.19173/irrodl.v12i3.890

Auer, C., Lysak, M., Nassar, A., Dolfi, M., Livathinos, N., Vagenas, P., Ramis, C. B., Omenetti, M., Lindlbauer, F., Dinkla, K., Mishra, L., Kim, Y., Gupta, S., Lima, R. T. de, Weber, V., Morin, L., Meijer, I., Kuropiatnyk, V., & Staar, P. W. J. (2024). Docling technical report (No. 2408.09869). arXiv. https://doi.org/10.48550/arXiv.2408.09869

Bingimlas, K. A. (2009). Barriers to the Successful Integration of ICT in Teaching and Learning Environments: A Review of the Literature. Eurasia Journal of Mathematics, Science and Technology Education, 5(3), 235–245. https://doi.org/10.12973/ejmste/75275

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., 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., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

Capp, M. J. (2017). The effectiveness of universal design for learning: A meta-analysis of literature between 2013 and 2016. International Journal of Inclusive Education, 21(8), 791–807. https://doi.org/10.1080/13603116.2017.1325074

Chen, L., Chen, P., & Lin, Z. (2020). Artificial Intelligence in Education: A Review. IEEE Access, 8, 75264–75278. https://doi.org/10.1109/ACCESS.2020.2988510

Chrysafiadi, K., & Virvou, M. (2012). Evaluating the integration of fuzzy logic into the student model of a web-based learning environment. Expert Systems with Applications, 39(18), 13127–13134. https://doi.org/10.1016/j.eswa.2012.05.089

Chrysafiadi, K., & Virvou, M. (2013). Student modeling approaches: A literature review for the last decade. Expert Systems with Applications, 40(11), 4715–4729. https://doi.org/10.1016/j.eswa.2013.02.007

cross-encoder. (2025, March 31). Ms-marco-MiniLM-L6-v2. Hugging Face. https://huggingface.co/cross-encoder/ms-marco-MiniLM-L6-v2

Dan, Y., Lei, Z., Gu, Y., Li, Y., Yin, J., Lin, J., Ye, L., Tie, Z., Zhou, Y., Wang, Y., Zhou, A., Zhou, Z., Chen, Q., Zhou, J., He, L., & Qiu, X. (2023). EduChat: A large-scale language model-based chatbot system for intelligent education (No. 2308.02773). arXiv. https://doi.org/10.48550/arXiv.2308.02773

Drachsler, H., Verbert, K., Santos, O. C., & Manouselis, N. (2015). Panorama of Recommender Systems to Support Learning. In F. Ricci, L. Rokach, & B. Shapira (Eds.), Recommender Systems Handbook (pp. 421–451). Springer US. https://doi.org/10.1007/978-1-4899-7637-6_12

Duolingo. (2023, March 14). Introducing Duolingo Max, a learning experience powered by GPT-4. Duolingo Blog. https://blog.duolingo.com/duolingo-max/

Ertmer, P. A., Ottenbreit-Leftwich, A. T., Sadik, O., Sendurur, E., & Sendurur, P. (2012). Teacher beliefs and technology integration practices: A critical relationship. Computers & Education, 59(2), 423–435. https://doi.org/10.1016/j.compedu.2012.02.001

Eversberg, D. L. (2025, November 17). PDF to Markdown Conversion Benchmark—Part 3. Medium. https://ai.gopubby.com/pdf-to-markdown-conversion-benchmark-part-3-01c22dce8e9d

Fleming, N. D., & Mills, C. (1992). Not Another Inventory, Rather a Catalyst for Reflection.

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. Proceedings of the National Academy of Sciences, 111(23), 8410–8415. https://doi.org/10.1073/pnas.1319030111

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 (No. 2312.10997). arXiv. https://doi.org/10.48550/arXiv.2312.10997

Genially. (n.d.). Genially | The easiest way to create interactive experiences. Genially. Retrieved November 25, 2025, from https://genially.com/

Järvelä, S., Kirschner, P. A., Hadwin, A., Järvenoja, H., Malmberg, J., Miller, M., & Laru, J. (2016). Socially shared regulation of learning in CSCL: Understanding and prompting individual- and group-level shared regulatory activities. International Journal of Computer-Supported Collaborative Learning, 11(3), 263–280. https://doi.org/10.1007/s11412-016-9238-2

Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with GPUs (No. 1702.08734). arXiv. https://doi.org/10.48550/arXiv.1702.08734

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/10.1016/j.lindif.2023.102274

Khan Academy. (n.d.). Meet Khanmigo: Khan Academy's AI-powered teaching assistant & tutor. Khanmigo. Retrieved November 25, 2025, from https://www.khanmigo.ai

Kopcha, T. J. (2012). Teachers' perceptions of the barriers to technology integration and practices with technology under situated professional development. Computers & Education, 59(4), 1109–1121. https://doi.org/10.1016/j.compedu.2012.05.014

Kudo, T., & Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing (No. 1808.06226). arXiv. https://doi.org/10.48550/arXiv.1808.06226

LangChain. (n.d.). LangChain overview. LangChain Documentation. Retrieved November 30, 2025, from https://langchain-5e9cc07a.mintlify.app/oss/python/langchain/overview

LangChain. (n.d.). LangGraph overview. LangChain. Retrieved November 30, 2025, from https://langchain-5e9cc07a.mintlify.app/oss/python/langgraph/overview

Learning & education. (n.d.). Google Blog. Retrieved November 25, 2025, from https://blog.google/outreach-initiatives/education/

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive NLP tasks (No. 2005.11401). arXiv. https://doi.org/10.48550/arXiv.2005.11401

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 (No. 2307.03172). arXiv. https://doi.org/10.48550/arXiv.2307.03172

LlamaIndex. (n.d.). Parse vs extract: Understanding two fundamental approaches to document processing. LlamaIndex Blog. Retrieved November 5, 2025, from https://www.llamaindex.ai/blog/parse-vs-extract

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs (No. 1603.09320). arXiv. https://doi.org/10.48550/arXiv.1603.09320

Mavroudis, V. (2024). LangChain v0.3. HAL. https://doi.org/10.20944/preprints202411.0566.v1

Mayer, R. E. (2002). Multimedia learning. In Psychology of Learning and Motivation (Vol. 41, pp. 85–139). Academic Press. https://doi.org/10.1016/S0079-7421(02)80005-6

Mayer, R. E., & Moreno, R. (2003). Nine ways to reduce cognitive load in multimedia learning. Educational Psychologist, 38(1), 43–52. https://doi.org/10.1207/S15326985EP3801_6

McCrindle. (2022, July 6). Understanding Generation Alpha. https://mccrindle.com.au/article/topic/generation-alpha/generation-alpha-defined/

Mindgrasp. (n.d.). Mindgrasp—AI learning platform for students and professionals. Mindgrasp. Retrieved November 25, 2025, from https://www.mindgrasp.ai/

Mittal, U., Sai, S., Chamola, V., & Sangwan, D. (2024). A Comprehensive Review on Generative AI for Education. IEEE Access, 12, 142733–142759. https://doi.org/10.1109/ACCESS.2024.3468368

Moreno, R., & Mayer, R. E. (1999). Cognitive principles of multimedia learning: The role of modality and contiguity. Journal of Educational Psychology, 91(2), 358–368. https://doi.org/10.1037/0022-0663.91.2.358

Mougiakou, E., Papadimitriou, S., Chrysafiadi, K., & Virvou, M. (2025). Artificial intelligence in educational games and consent under general data protection regulation. Intelligent Decision Technologies, 19(2), 670–686. https://doi.org/10.1177/18724981251322884

Nearpod. (n.d.). Nearpod: Foster a love of learning in every student. Nearpod. Retrieved November 25, 2025, from http://nearpod.com

OECD. (2019). TALIS 2018 Results (Volume I): Teachers and School Leaders as Lifelong Learners. OECD. https://doi.org/10.1787/1d0bc92a-en

OpenAI. (n.d.). GPT-4o mini: Advancing cost-efficient intelligence. OpenAI. Retrieved November 8, 2025, from https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

OpenAI. (n.d.). Hello GPT-4o. OpenAI. Retrieved November 25, 2025, from https://openai.com/index/hello-gpt-4o/

Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., & Wu, X. (2024). Unifying Large Language Models and Knowledge Graphs: A Roadmap. IEEE Transactions on Knowledge and Data Engineering, 36(7), 3580–3599. https://doi.org/10.1109/TKDE.2024.3352100

Panagoulias, D. P., Tsoureli-Nikita, E., Virvou, M., & Tsihrintzis, G. A. (2025). Dermacen analytica: A novel methodology integrating multi-modal large language models with machine learning in dermatology. International Journal of Medical Informatics, 199, 105898. https://doi.org/10.1016/j.ijmedinf.2025.105898

Panagoulias, D. P., Virvou, M., & Tsihrintzis, G. A. (2024). Evaluating LLM -- generated multimodal diagnosis from medical images and symptom analysis (No. 2402.01730). arXiv. https://doi.org/10.48550/arXiv.2402.01730

Papert, S. (1993). Mindstorms: Children, computers, and powerful ideas (2nd ed.). Basic Books.

PDFPlumber. (n.d.). PDFPlumber [Computer software]. Retrieved November 30, 2025, from https://www.pdfplumber.com/

Purwar, A. and others. (2024). Evaluating the efficacy of open-source LLMs in enterprise-specific RAG systems: A comparative study of performance and scalability (No. 2406.11424). arXiv. https://doi.org/10.48550/arXiv.2406.11424

PyMuPDF. (n.d.). PyMuPDF documentation. PyMuPDF. Retrieved November 30, 2025, from https://pymupdf.readthedocs.io/en/latest/

Qdrant. (n.d.). Qdrant documentation. Qdrant. Retrieved November 30, 2025, from https://qdrant.tech/documentation/

Quizlet Inc. (n.d.). Digital flashcards & revision cards for students. Quizlet. Retrieved November 25, 2025, from https://quizlet.com/gb

Ralabate, P. K. (2011). Universal Design for Learning: Meeting the Needs of All Students. The ASHA Leader Archive, 16(10), 14–17. https://doi.org/10.1044/leader.FTR2.16102011.14

Reckase, M. D. (2009). Multidimensional Item Response Theory Models. In M. D. Reckase (Ed.), Multidimensional Item Response Theory (pp. 79–112). Springer. https://doi.org/10.1007/978-0-387-89976-3_4

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-Networks (No. 1908.10084). arXiv. https://doi.org/10.48550/arXiv.1908.10084

Roblyer, M. D., & Hughes, J. E. (2019). Integrating educational technology into teaching: Transforming learning across disciplines (8 e, student edition). Pearson.

Seemiller, C., & Grace, M. (2017). Generation Z: Educating and Engaging the Next Generation of Students. About Campus: Enriching the Student Learning Experience, 22(3), 21–26. https://doi.org/10.1002/abc.21293

Sentence-Transformers. (n.d.). MS MARCO cross-encoders. Sentence Transformers Documentation. Retrieved November 30, 2025, from https://www.sbert.net/docs/pretrained-models/ce-msmarco.html

sentence-transformers. (2024, January 5). All-MiniLM-L6-v2. Hugging Face. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Shi, X., Zhu, Z., Zhang, Z., & Li, C. (2023). Hallucination mitigation in natural language generation from large-scale open-domain knowledge graphs. In H. Bouamor, J. Pino, & K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (pp. 12506–12521). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.770

Shusterman, R. (1999). Somaesthetics: A Disciplinary Proposal. The Journal of Aesthetics and Art Criticism, 57(3), 299–313. https://doi.org/10.2307/432196

Storywizard.ai. (n.d.). Create incredible learning experiences using AI. Storywizard.Ai. Retrieved November 25, 2025, from https://www.storywizard.ai/

Suprayogi, M. N., Valcke, M., & Godwin, R. (2017). Teachers and their implementation of differentiated instruction in the classroom. Teaching and Teacher Education, 67, 291–301. https://doi.org/10.1016/j.tate.2017.06.020

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. Learning and Instruction, 4(4), 295–312. https://doi.org/10.1016/0959-4752(94)90003-5

VanLehn, K. (2011). The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. Educational Psychologist, 46(4), 197–221. https://doi.org/10.1080/00461520.2011.611369

VARK Learn Limited. (2014, July 30). VARK questionnaire: How do you learn best? VARK. https://vark-learn.com/the-vark-questionnaire/

Virvou, M., & Alepis, E. (2005). Mobile educational features in authoring tools for personalised tutoring. Computers & Education, 44(1), 53–68. https://doi.org/10.1016/j.compedu.2003.12.020

Virvou, M., Katsionis, G., & Manos, K. (2005). Combining Software Games with Education: Evaluation of its Educational Effectiveness. Educational Technology & Society, 8(2), 54–65.

Virvou, M., & Tsihrintzis, G. A. (2023a). A novel trust state-chart model for requirements engineering of trustful AI-empowered software. The 14th International Conference on Information, Intelligence, Systems & Applications (IISA), 2023, 1–6. https://doi.org/10.1109/IISA59645.2023.10345934

Virvou, M., & Tsihrintzis, G. A. (2023b). Is ChatGPT beneficial to education? A holistic evaluation framework based on intelligent tutoring systems. The14th International Conference on Information, Intelligence, Systems & Applications (IISA), 2023, 1–8. https://doi.org/10.1109/IISA59645.2023.10345949

Virvou, M., & Tsihrintzis, G. A. (2023c). Pre-made empowering artificial intelligence and ChatGPT: The growing importance of human AI-experts. The14th International Conference on Information, Intelligence, Systems & Applications (IISA), 2023, 1–8. https://doi.org/10.1109/IISA59645.2023.10345880

Virvou, M., Tsihrintzis, G., Sotiropoulos, D., Chrysafiadi, K., Sakkopoulos, E., & Tsichrintzi, E.-A. (2023). ChatGPT in Artificial Intelligence-Empowered E-Learning for Cultural Heritage: The case of Lyrics and Poems. The 14th International Conference on Information, Intelligence, Systems & Applications (IISA), 2023, 1–9. https://doi.org/10.1109/IISA59645.2023.10345878

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.-R. (2024). A Survey on Large Language Model based Autonomous Agents. Frontiers of Computer Science, 18(6), 186345. https://doi.org/10.1007/s11704-024-40231-1

Wayground. (n.d.). Create quizzes, lessons and more with Wayground AI. Wayground. Retrieved November 25, 2025, from https://wayground.com/quizizz-ai

Appendix I Tables

Table 1: Source code implementation (snippet) of the VARK Assessment AI Chain

```
def get_vark_results_assessment_fn(vark_results: str) -> str:
    """Enhanced VARK assessment with better formatting and analysis."""
    vark_assessment_prompt_template = """
    You are an expert in VARK learning style assessment. Analyze the student's results
and provide a comprehensive, personalized assessment.

    VARK Results: {results}

    Provide a detailed assessment following this structure:

    <h2>🎯 Your Dominant Learning Style(s)</h2>
    [Identify and list the dominant styles with scores]

    <h2>📊 Detailed Analysis</h2>
    [Provide thorough interpretation for each learning style, starting with dominant
ones]

    <h2>🎓 Personalized Learning Recommendations</h2>
    [Specific, actionable strategies tailored to their learning preferences]

    <h2>💡 Key Insights for You</h2>
    [Personalized feedback and advice directly addressing the student]

    <h2>📚 Study Strategies</h2>
    [Concrete study methods and techniques based on their profile]

    Requirements:
    - Address the student directly using "you" and "your"
    - Use HTML formatting with proper tags (h2, h3, p, ul, li, strong, em)
    - Include specific scores and percentages where relevant
    - Provide actionable, practical advice
    - Use engaging, encouraging tone
    - Add relevant emojis for visual appeal
    - Ensure proper HTML structure without code blocks
    """

    from langchain_core.prompts import PromptTemplate
    vark_assessment_prompt = PromptTemplate(
        input_variables=["results"],
        template=vark_assessment_prompt_template
    )
    vark_assessment_agent_chain = vark_assessment_prompt | llm_gpt_4o_mini

    vark_assessment_res = vark_assessment_agent_chain.invoke({"results":
vark_results})
    return vark_assessment_res.content
```

Table 2: Source code implementation (snippet) of the “Query text collection” tool

```
def query_text_collection(
    query_text: str,
    text_collection_name: str = text_collection,
    k: int = 25,
```

```
    n: int = 8,
    rerank: bool = True,
    parent_folder_filters: list = [],
    score_threshold: float = -5.0
) -> dict:
    """
    Enhanced text collection search with optimized performance and detailed reporting.

    Args:
        query_text: The search query text
        text_collection_name: Name of the Qdrant collection to search
        k: Number of initial results to retrieve from vector search
        n: Number of top results to return after filtering
        rerank: Whether to apply cross-encoder re-ranking
        parent_folder_filters: List of folder names to filter results by
        score_threshold: Minimum score threshold - results below this score will be
filtered out

    Returns:
        dict: Search results with scores, sources, and text content, plus metadata
about retrieval quality
    """
    try:
        print(f"□ Starting enhanced text collection search for:
'{query_text[:50]}...'")

        client = qdrant_client.QdrantClient(
            url=os.environ.get("QDRANT_URL"),
            api_key=os.environ.get("QDRANT_API_KEY"),
            timeout=60
        )

        try:
            client.create_payload_index(
                collection_name=text_collection_name,
                field_name="parent_folder",
                field_schema=models.PayloadSchemaType.KEYWORD,
                wait=True
            )
        except Exception:
            pass

        query_embedding =
Common_Toolset_Core.hf_text_model.encode(query_text).tolist()

        # Build the filter condition
        filter_conditions = []
        if parent_folder_filters:
            filter_conditions.append(
                models.FieldCondition(
                    key="parent_folder",
                    match=models.MatchAny(any=parent_folder_filters)
                )
            )

        search_filter = models.Filter(must=filter_conditions) if filter_conditions
else None

        # Perform search
        search_results = client.search(
            collection_name=text_collection_name,
            query_vector=query_embedding,
```

```
            limit=k,
            with_payload=True,
            query_filter=search_filter
        )

        if not search_results:
            return {
                "results": [],
                "status": "No results found",
                "retrieval_quality": "insufficient",
                "quality_details": "No matches found in knowledge base"
            }

        # Enhanced re-ranking with parallel processing
        if rerank and Common_Toolset_Core.cross_encoder_model:
            pairs = [
                [query_text, point.payload.get('original_text', '')]
                for point in search_results
                if point.payload.get('original_text')
            ]
            if pairs:
                # Batch processing for better performance
                batch_size = 32
                all_scores = []
                for i in range(0, len(pairs), batch_size):
                    batch = pairs[i:i+batch_size]
                    batch_scores =
Common_Toolset_Core.cross_encoder_model.predict(batch)
                    all_scores.extend(batch_scores)

                for point, score in zip(search_results, all_scores):
                    point.score = score
                search_results.sort(key=lambda p: p.score, reverse=True)

        top_results = search_results[:n]

        results = []
        high_quality_results = 0
        for point in top_results:
            score = float(point.score)
            if score >= score_threshold:
                results.append({
                    "score": score,
                    "source": point.payload.get("source"),
                    "parent_folder": point.payload.get("parent_folder"),
                    "page": point.payload.get("page"),
                    "original_text": point.payload.get("original_text"),
                    "enriched_text": point.payload.get("enriched_text")
                })
                if score > 0.5:  # High confidence threshold
                    high_quality_results += 1

        # Assess retrieval quality
        if len(results) == 0:
            retrieval_quality = "insufficient"
            quality_details = "No results met the score threshold"
        elif high_quality_results >= 3:
            retrieval_quality = "excellent"
            quality_details = f"Found {high_quality_results} high-confidence matches"
        elif len(results) >= 2:
            retrieval_quality = "good"
            quality_details = f"Found {len(results)} relevant matches"
```

```
        else:
            retrieval_quality = "limited"
            quality_details = f"Only {len(results)} relevant match(es) found"

        return {
            "results": results,
            "status": "success",
            "retrieval_quality": retrieval_quality,
            "quality_details": quality_details,
            "total_searched": len(search_results),
            "filtered_results": len(results)
        }

    except Exception as e:
        error_msg = f"Error in enhanced query_text_collection: {str(e)}"
        return {
            "error": error_msg,
            "results": [],
            "retrieval_quality": "failed"
        }
```

Table 3: Source code implementation (snippet) of the "extract images from results" tools

```
def extract_images_from_results(
    search_results: dict,
    max_images: int = 10
) -> dict:
    """
    Enhanced image extraction with optimized parallel processing for multiple
references.
    Processes all unique pages from references in parallel for maximum efficiency.
    """
    if not search_results or "results" not in search_results:
        return {"error": "No search results provided", "image_paths": []}

    try:
        import fitz
        from PIL import Image
        import io

        print(f"🖼️ Starting parallel image extraction from
{len(search_results.get('results', []))} references...")

        saved_image_paths = []
        processed_pages = set()

        # Collect ALL unique pages for processing (not limited by max_images here)
        pages_to_process = []
        for result in search_results.get("results", []):
            source = result.get("source")
            page_num = result.get("page")

            print(f"🔍 DEBUG: Processing result - Source: {source}, Page:
{page_num}")

            if source and page_num is not None:
                page_key = f"{source}_{page_num}"
                if page_key not in processed_pages:
                    processed_pages.add(page_key)
                    page_index = page_num - 1 if page_num > 0 else 0
```

```
                    # Enhanced path resolution for actual PDF files
                    pdf_path = None

                    if source.startswith('./00 Data'):
                        source = source.replace('./', 'AI/')

                    # Strategy 1: Use the source path directly if it exists
                    if os.path.exists(source):
                        pdf_path = source
                        print(f"✅ Found PDF using direct path: {pdf_path}")
                    else:
                        # Strategy 2: Look for "00 Data" in the path and construct
relative path
                        data_index = source.find("00 Data")
                        if data_index != -1:
                            relative_path = source[data_index:]

                            # Try multiple relative path strategies
                            possible_paths = [
                                f"./{relative_path}",  # From root directory
                                f"Agents_Dev/{relative_path}",  # From Agents_Dev
subdirectory
                                relative_path  # Direct relative path
                            ]

                            for possible_path in possible_paths:
                                print(f"🔍 Trying path: {possible_path}")
                                if os.path.exists(possible_path):
                                    pdf_path = possible_path
                                    print(f"✅ Found PDF at: {pdf_path}")
                                    break
                                else:
                                    print(f"❌ Path doesn't exist: {possible_path}")

                        # Strategy 3: Try just the filename in various 00 Data
locations
                        if pdf_path is None:
                            filename = os.path.basename(source)

                            # Search in multiple possible locations
                            search_locations = [
                                "./00 Data",
                                "Agents_Dev/00 Data"
                            ]

                            for search_location in search_locations:
                                if os.path.exists(search_location):
                                    print(f"🔍 Searching in: {search_location}")
                                    for root, dirs, files in os.walk(search_location):
                                        if filename in files:
                                            pdf_path = os.path.join(root, filename)
                                            print(f"✅ Found PDF by filename search:
{pdf_path}")
                                            break
                                    if pdf_path:
                                        break

                            if pdf_path is None:
                                print(f"❌ Could not find PDF file: {filename}")

                    if pdf_path and os.path.exists(pdf_path):
```

```
                        pages_to_process.append((pdf_path, page_index, page_num,
source))
                        print(f"✅ DEBUG: Added page to process: {pdf_path}, page
{page_num}")
                    else:
                        print(f"❌ DEBUG: Could not find PDF file for source:
{source}")

        print(f"📄 Found {len(pages_to_process)} unique pages to process in
parallel...")

        # Enhanced parallel processing function
        def extract_from_page(page_info):
            pdf_path, page_index, page_num, source = page_info
            local_image_paths = []

            try:
                print(f"🔍 Processing page {page_num} from
{os.path.basename(source)}...")
                doc = fitz.open(pdf_path)

                if page_index < len(doc):
                    page_obj = doc[page_index]
                    image_list = page_obj.get_images(full=True)
                    images_found_on_page = False

                    # Extract embedded images with higher limits for parallel
processing
                    for img_index, img_info in enumerate(image_list):
                        if len(local_image_paths) >= 3:  # Increased limit per page
for parallel processing
                            break

                        try:
                            xref = img_info[0]
                            pix = fitz.Pixmap(doc, xref)

                            if pix.n - pix.alpha < 4:  # Not CMYK
                                pil_img = Image.open(io.BytesIO(pix.tobytes("png")))

                                # More lenient size requirements for better extraction
                                if pil_img.width > 100 and pil_img.height > 100:
                                    image_filename =
f"extracted_{os.path.basename(source).replace('.pdf',
'')}_p{page_num}_img{img_index}_{uuid.uuid4().hex[:6]}.png"
                                    save_path = os.path.join(IMAGES_DIR,
image_filename)

                                    # Optimize image before saving
                                    if pil_img.width > 1024 or pil_img.height > 1024:
                                        pil_img.thumbnail((1024, 1024),
Image.Resampling.LANCZOS)

                                    pil_img.save(save_path, format='PNG',
optimize=True)
                                    local_image_paths.append(save_path)
                                    images_found_on_page = True
                                    print(f"✅ Extracted embedded image:
{image_filename}")

                            pix = None
```

```
                        except Exception as img_e:
                            print(f"⚠️ Error extracting embedded image {img_index}
from page {page_num}: {img_e}")
                            continue

                    # Always render the page as fallback (especially important for
mock PDFs)
                    if not images_found_on_page or len(local_image_paths) == 0:
                        try:
                            pix = page_obj.get_pixmap(dpi=200)  # Higher DPI for
better quality
                            pil_img = Image.open(io.BytesIO(pix.tobytes("png")))

                            # For mock PDFs, always extract the rendered page
                            if "mock_" in pdf_path or len(local_image_paths) == 0:
                                image_filename =
f"rendered_page_{os.path.basename(source).replace('.pdf',
'')}_p{page_num}_{uuid.uuid4().hex[:6]}.png"
                                save_path = os.path.join(IMAGES_DIR, image_filename)

                                # Optimize rendered page
                                if pil_img.width > 1200 or pil_img.height > 1200:
                                    pil_img.thumbnail((1200, 1200),
Image.Resampling.LANCZOS)

                                pil_img.save(save_path, format='PNG', optimize=True)
                                local_image_paths.append(save_path)
                                print(f"✅ Rendered full page: {image_filename}")

                            pix = None

                        except Exception as render_e:
                            print(f"⚠️ Error rendering page {page_num}: {render_e}")

                doc.close()
                print(f"📊 Page {page_num} processing complete:
{len(local_image_paths)} images extracted")
                return local_image_paths

            except Exception as e:
                print(f"❌ Error processing page {page_num} from {source}: {e}")
                return []

        # Use optimized ThreadPoolExecutor with increased workers for better
parallelization
        max_workers = min(8, len(pages_to_process))  # Scale workers based on pages to
process
        print(f"🚀 Starting parallel extraction with {max_workers} workers...")

        with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as
executor:
            # Submit all pages for parallel processing
            future_to_page = {
                executor.submit(extract_from_page, page_info): page_info
                for page_info in pages_to_process
            }

            # Collect results as they complete
            completed_pages = 0
            for future in concurrent.futures.as_completed(future_to_page):
```

```
                try:
                    page_images = future.result()
                    saved_image_paths.extend(page_images)
                    completed_pages += 1

                    page_info = future_to_page[future]
                    print(f"✅ Completed {completed_pages}/{len(pages_to_process)}
pages. Total images so far: {len(saved_image_paths)}")

                    # Early termination if we have enough images
                    if len(saved_image_paths) >= max_images * 2:  # Allow some buffer
for selection
                        print(f"🎯 Reached sufficient images
({len(saved_image_paths)}), continuing with remaining parallel tasks...")

                except Exception as e:
                    page_info = future_to_page[future]
                    print(f"❌ Error in parallel image extraction for {page_info[3]}:
{e}")

        # Sort by creation time and select best images
        final_images = saved_image_paths[:max_images]

        print(f"🎉 Parallel image extraction completed!")
        print(f"📊 Processed {len(pages_to_process)} pages in parallel")
        print(f"🖼️ Extracted {len(saved_image_paths)} total images")
        print(f"✨ Returning {len(final_images)} final images")

        return {
            "image_paths": final_images,
            "status": "success" if final_images else "no_images_found",
            "total_extracted": len(saved_image_paths),
            "pages_processed": len(pages_to_process),
            "parallel_workers": max_workers
        }

    except Exception as e:
        return {"error": f"Error in enhanced parallel image extraction: {str(e)}",
"image_paths": []}
```

Table 4: Source code implementation (snippet) of the “analyze user image and text content” tools

```
def analyze_user_image(
    image_path: str,
    query_text: str = "",
    analysis_focus: str = "general"
) -> dict:
    """
    Enhanced image analysis with better processing and error handling.
    """
    try:
        if image_path.startswith(('http://', 'https://')):
            # Fix: call the function directly, not as a tool with .invoke()
            download_result = download_image_from_url(image_url=image_path)
            if download_result.get("status") != "success":
                return download_result
            image_path = download_result["local_path"]

        if not os.path.exists(image_path):
```

```
            return {"error": f"Image file not found: {image_path}", "status":
"failed"}

        with Image.open(image_path) as img:
            if img.mode != 'RGB':
                img = img.convert('RGB')
            max_size = (1024, 1024)
            img.thumbnail(max_size, Image.Resampling.LANCZOS)
            buffer = io.BytesIO()
            img.save(buffer, format='PNG')
            img_base64 = base64.b64encode(buffer.getvalue()).decode()

        from langchain_core.messages import HumanMessage
        message = HumanMessage(
            content=[
                {"type": "text", "text": f"Analyze this image in detail regarding the
user's query: '{query_text}'. Focus on: {analysis_focus}. Provide comprehensive
analysis including key visual elements, context, and relevance to the query."},
                {"type": "image_url", "image_url": {"url":
f"data:image/png;base64,{img_base64}"}}
            ]
        )

        vision_response = llm_gpt_4o.invoke([message])

        return {
            "detailed_analysis": vision_response.content,
            "image_path": image_path,
            "status": "success"
        }

    except Exception as e:
        return {"error": f"Error in enhanced image analysis: {str(e)}", "status":
"failed"}


def analyze_text_content(
    file_content: str,
    query_text: str = "",
    filename: str = "user_file.txt"
) -> dict:
    """
    Analyzes text content provided directly as a string, in relation to a user's
query.
    Use this tool when a user uploads a file and you need to understand its content.
    """
    try:
        # Content is already provided as an argument.
        content = file_content

        # Truncate if content is too long for the prompt
        if len(content) > 8000:
            content = content[:8000] + "\n\n[Content truncated for analysis...]"

        analysis_prompt = f"""Analyze the following file content in relation to the
user's query: '{query_text}'

File: {filename}
Content:
{content}

Provide a comprehensive analysis including:
```

```
1. Key information relevant to the query.
2. Main concepts and topics covered.
3. How this content relates to the user's question.
4. Any important details or insights.
"""

        analysis_response = llm_gpt_4o_mini.invoke(analysis_prompt)

        return {
            "content_summary": analysis_response.content,
            "filename": filename,
            "content_length": len(file_content), # Report original length
            "status": "success"
        }

    except Exception as e:
        return {"error": f"Error in text content analysis: {str(e)}", "status":
"failed"}
```

Table 5: Source code implementation (snippet) of the "mathematical problem solver" tool

```
def mathematical_problem_solver(
    problem_statement: str
) -> dict:
    """
    Enhanced mathematical problem solver with better error handling and detailed
solutions.
    """
    print(f"🧮 Starting enhanced mathematical problem solver for:
'{problem_statement[:100]}...'")

    try:
        from langchain_experimental.tools.python.tool import PythonREPLTool
        from sympy import symbols, Eq, solve, simplify, expand, factor

        python_tool = PythonREPLTool(
            globals={
                "symbols": symbols,
                "Eq": Eq,
                "solve": solve,
                "simplify": simplify,
                "expand": expand,
                "factor": factor
            }
        )

        solver_prompt = f"""
        You are an expert mathematician. Solve this problem step-by-step using Python
and SymPy.

        Problem: "{problem_statement}"

        Instructions:
        1. **Analyze**: Identify the mathematical concepts and what's being asked
        2. **Plan**: Outline your solution approach
        3. **Code**: Write clear, well-commented Python code using SymPy
        4. **Execute**: Show all calculations and intermediate steps
        5. **Verify**: Check your answer makes sense
        6. **Explain**: Provide a clear explanation of the solution

        Use proper mathematical notation and show your work clearly.
        """
```

```
        solution_response = llm_gpt_4o.invoke(solver_prompt)
        solution_explanation = solution_response.content

        # Extract and execute Python code
        import re
        python_code_matches = re.findall(r"```python\n(.*?)```", solution_explanation,
re.DOTALL)

        code_outputs = []
        for i, python_code in enumerate(python_code_matches):
            try:
                print(f"🐍 Executing Python code block {i+1}...")
                code_output = python_tool.invoke({"query": python_code})
                code_outputs.append(f"Code Block {i+1} Output:\n{code_output}")
                print(f"💻 Code Output {i+1}:\n{code_output}")
            except Exception as e:
                code_outputs.append(f"Code Block {i+1} Error: {str(e)}")

        return {
            "problem": problem_statement,
            "solution_explanation": solution_explanation,
            "python_code_blocks": python_code_matches,
            "code_outputs": code_outputs,
            "status": "success"
        }

    except Exception as e:
        return {"error": f"Error in enhanced mathematical problem solver: {str(e)}",
"status": "failed"}
```

Table 6: Source code implementation (snippet) of the “calculator” tool

```
def calculator(expression: str) -> str:
    """Calculate expression using Python's numexpr library.

    Expression should be a single line mathematical expression
    that solves the problem.

    Examples:
        "37593 * 67" for "37593 times 67"
        "37593**(1/5)" for "37593^(1/5)"
    """
    local_dict = {"pi": math.pi, "e": math.e}
    return str(
        numexpr.evaluate(
            expression.strip(),
            global_dict={},  # restrict access to globals
            local_dict=local_dict,  # add common mathematical functions
        )
    )
```

Table 7: Source code implementation (snippet) of the “generate synthetic images” tool

```
def generate_synthetic_images(
    user_question: str,
    response_content: str
) -> dict:
    """
    Enhanced synthetic image generation with full parallelization and better error
handling.
```

```
    """
    try:
        from langchain_core.messages import HumanMessage

        print(f"🎨 Starting enhanced synthetic image generation for:
'{user_question[:50]}...'")

        generated_image_paths = []

        # Enhanced parallel description generation
        def generate_conceptual_description():
            return llm_gpt_4o_mini.invoke([
                HumanMessage(content=f"""Create a detailed description for a
CONCEPTUAL OVERVIEW image:

                User Question: {user_question}
                Response Content: {response_content[:500]}...

                Focus on HIGH-LEVEL CONCEPTS and BIG PICTURE understanding:
                - Main visual elements for conceptual understanding
                - Layout showing relationships between ideas
                - Clean, professional design with clear labels
                - Conceptual diagram, mind map, or framework approach
                - Colors: professional blues, greens, and neutrals

                Provide a concise, clear description (max 150 words) suitable for
image generation.""")
            ])

        def generate_detailed_description():
            return llm_gpt_4o_mini.invoke([
                HumanMessage(content=f"""Create a detailed description for a
PRACTICAL/DETAILED image:

                User Question: {user_question}
                Response Content: {response_content[:500]}...

                Focus on SPECIFIC DETAILS and PRACTICAL APPLICATION:
                - Step-by-step visual elements or specific examples
                - Process flow or implementation details
                - Detailed annotations and specific data
                - Workflow, tutorial, or case study approach
                - Colors: warm oranges, reds, and practical earth tones

                This should be COMPLETELY DIFFERENT from a conceptual overview.
                Provide a concise, clear description (max 150 words) suitable for
image generation.""")
            ])

        # Generate descriptions in parallel
        print("🖼️ Generating image descriptions in parallel...")
        with concurrent.futures.ThreadPoolExecutor(max_workers=2) as executor:
            future_conceptual = executor.submit(generate_conceptual_description)
            future_detailed = executor.submit(generate_detailed_description)

            conceptual_response = future_conceptual.result()
            detailed_response = future_detailed.result()

        print("✅ Both image descriptions generated successfully")

        # Enhanced parallel image generation
        def generate_single_image(description_response, image_number, image_type):
```

```
            try:
                image_filename =
f"synthetic_image_{uuid.uuid4().hex[:8]}_{image_number}.png"
                image_path = os.path.join(IMAGES_DIR, image_filename)

                # Create optimized prompt for image generation
                image_prompt = description_response.content[:180].strip()
                image_prompt = ' '.join(image_prompt.replace('\n', ' ').split())

                # Add style enhancements
                if image_type == "CONCEPTUAL":
                    image_prompt += " professional diagram style, clean layout,
educational infographic"
                else:
                    image_prompt += " detailed illustration, step-by-step visual,
practical guide style"

                encoded_prompt = quote(image_prompt)
                pollinations_url =
f"https://image.pollinations.ai/prompt/{encoded_prompt}"

                print(f"🌐 Requesting {image_type} image from Pollinations API...")
                response = requests.get(pollinations_url, timeout=45)

                if response.status_code == 200:
                    img = Image.open(io.BytesIO(response.content))
                    # Optimize image size
                    if img.width > 1024 or img.height > 1024:
                        img.thumbnail((1024, 1024), Image.Resampling.LANCZOS)
                    img.save(image_path, 'PNG', optimize=True)
                    print(f"✅ Successfully generated {image_type} image:
{image_path}")
                    return image_path
                else:
                    print(f"❌ Pollinations API error for {image_type} image:
{response.status_code}")
                    _create_placeholder_image(image_path,
description_response.content, user_question, image_number)
                    return image_path

            except Exception as e:
                print(f"❌ Error generating {image_type} image: {str(e)}")
                image_filename =
f"synthetic_image_{uuid.uuid4().hex[:8]}_{image_number}.png"
                image_path = os.path.join(IMAGES_DIR, image_filename)
                _create_placeholder_image(image_path, description_response.content,
user_question, image_number)
                return image_path

        # Generate both images in parallel
        print("🖼️ Generating both images in parallel...")
        with concurrent.futures.ThreadPoolExecutor(max_workers=2) as executor:
            future_conceptual_image = executor.submit(
                generate_single_image,
                conceptual_response,
                1,
                "CONCEPTUAL"
            )
            future_detailed_image = executor.submit(
                generate_single_image,
                detailed_response,
```

```
                2,
                "DETAILED"
            )

            # Collect results
            conceptual_image_path = future_conceptual_image.result()
            detailed_image_path = future_detailed_image.result()

            if conceptual_image_path:
                generated_image_paths.append(conceptual_image_path)
            if detailed_image_path:
                generated_image_paths.append(detailed_image_path)

        print(f"🎉 Enhanced parallel image generation completed! Generated
{len(generated_image_paths)} images")

        return {
            "synthetic_images": generated_image_paths,
            "status": "success",
            "count": len(generated_image_paths),
            "generation_method": "parallel_optimized"
        }

    except Exception as e:
        return {
            "synthetic_images": [],
            "status": "failed",
            "error": f"Error in enhanced synthetic image generation: {str(e)}"
        }
```