POLITECNICO DI TORINO

Department of Automation and Computer Science
Class LM-32

Master's Thesis in Computer Engineering

Development of an AI-enhanced microlearning platform for enterprise knowledge transfer

In collaboration with Reply S.p.A.

Tutor:
Prof. Laura Farinetti
Company tutor:
Andrea Lupo

Candidate:
Veronica Mattei

Academic Year 2024/2025

SUMMARY

1. INTRODUCTION 1

1.1 Context and motivation 1

1.2 Research objectives 1

1.3 Thesis structure 2

2. STATE OF THE ART 3

2.1 Microlearning definition and context 3

2.1.1 Benefits of microlearning 3

2.1.2 Limitations of microlearning 5

2.2 Existing technologies for enterprise training 7

2.2.1 E-learning platforms 7

2.2.3 Microlearning platforms 7

2.2.3 Learning Management Systems 7

2.2.4 Authoring tools 8

2.2.5 AI-powered tools and chatbots 8

2.2.6 Key limitations of existing technologies 8

2.3 Application of Artificial Intelligence in e-learning 9

2.3.1 Personalized and adaptive learning 9

2.3.2 Intelligent Tutoring Systems and Virtual Assistants 9

2.3.3 AI in learning analytics and assessment 9

2.3.4 Enhancing accessibility and inclusion 10

2.3.5 Recent trends and innovations 10

2.3.6 Ethical considerations and challenges 11

2.4 Standards and formats for digital training (SCORM, xAPI) 12

2.4.1 SCORM use cases, advantages, and limitations 12

2.4.2 xAPI use cases, advantages, and limitations 13

2.4.3 Comparison of technical architectures 14

2.4.4 Applications in corporate and academic settings 15

3. PLATFORM DESIGN AND ARCHITECTURE 16

3.1 Functional and non-functional requirements 16

3.1.1 Functional requirements 17

3.1.2 Non-functional requirements 18

3.2 User personas and use cases 19

3.2.1 Training Manager 19
3.2.2 Domain Expert 19
3.2.3 User profiles summary 19
3.3 Survey of AI techniques for automated content generation 20
3.3.1 AI for content summarization 20
3.3.2 Document parsing and preprocessing 22
3.3.3 Online resource retrieval 24
3.3.4 Slide generation techniques and quiz generation 25
3.3.5 Image generation 25
3.3.6 Multilingual support 26
3.4 Technology stack selection 26
3.4.1 Backend architecture and technologies 27
3.4.2 AI models 32
3.4.3 Frontend architecture and technologies 33
3.4.4 Cloud infrastructure and security layer 37
4. PLATFORM IMPLEMENTATION AND FEATURES 43
4.1 State management 43
4.1.1 *React Context Providers* 44
4.1.2 Custom hooks for stateful logic 44
4.1.3 State transitions and side effect handling 45
4.1.4 Benefits of the architecture 45
4.2 Content upload and management 45
4.2.1 Supported resource types 46
4.2.2 Resource structure and metadata 46
4.2.3 Upload workflow 46
4.2.4 Resource management features 46
4.2.5 User interface and experience 46
4.2.6 File management logic 48
4.2.7 Security and validation mechanisms 48
4.2.8 UI state management 48
4.2.9 Utility functions 48
4.2.10 Backend integration 49
4.3 Online resource retrieval 49
4.4 Automated generation of learning modules 50
4.5 Editing and customization of content 52
4.6 Integration of a chatbot for user assistance 53

4.7 Multilingual support 54
4.8 Development tools 54
5. EXPERIMENTAL RESULTS AND FUTURE WORK 56
5.1 Manual testing methodology 56
5.1.1 Example of user interaction flow: from user query to microlearning slides 57
5.1.2 Aggregated metrics from the interaction flow 67
5.1.3 Aggregated metrics from repeated interaction experiments 69
5.1.4 Discussion and analysis 70
5.2 Automated testing methodology 70
5.2.1 Planned performance evaluation 71
5.2.2 User experience and usability testing 71
5.2.3 Conclusions 72
6. CONCLUSIONS AND FUTURE DEVELOPMENTS 73
6.1 Limitations and challenges encountered 73
6.1.1 Limited input modalities 73
6.1.2 Inefficiencies in image generation 73
6.1.3 Lack of a visual editing interface 73
6.1.4 Basic authentication scheme 73
6.1.5 File size limitations and lack of persistent storage 74
6.1.6 Absence of structured evaluation 74
6.1.7 Fixed slide template and limited customization 74
6.1.8 Inaccessibility of source files 74
6.1.9 Dependency on external APIs and third-party services 74
6.1.10 Summary of key limitations 75
6.2 Potential enhancements and future work 75
6.2.1 Support for expanded input modalities 75
6.2.2 Advanced visual asset handling 75
6.2.3 Parallel image generation 75
6.2.4 Visual slide editing interface 76
6.2.5 Flashcard module for spaced repetition 76
6.2.6 Formal evaluation and benchmarking 76
6.2.7 LLM model selection and customization 76
6.2.8 Enhanced authentication and security 76
6.3 Strengths of the developed application 78
6.4 Conclusion 78
BIBLIOGRAPHY 79

Chapter 1

1. INTRODUCTION

This thesis is the result of an internship carried out at *Reply S.p.A.*, during which I developed the proposed application.

1.1 Context and motivation

In the contemporary landscape, which is characterized by very rapid technological changes, there is an ever-increasing need for companies to train their employees effectively in order to maintain their competitiveness.

However, traditional methods are beginning to show their limitations, often offering long learning sessions that do not fit well with the current pace of the working world and do not fully meet the learners' needs, not considering their pre-existing skills and learning methods. Uniform content for all, not tailored to individual preferences, fails to stimulate and motivate learners, who also struggle to maintain high levels of concentration. Traditional courses are poorly scalable, require physical presence at pre-established times, potentially creating conflicts with workloads and limiting participation. In addition to these aspects, using traditional methods for knowledge transfer also presents significant economic disadvantages: the costs of qualified teachers, equipped classrooms, and the production of learning materials must all be considered. This lack of customization and flexibility, therefore, reduces the effectiveness of learning.

In this context, microlearning aims to overcome these limitations, offering more flexible and dynamic learning experiences. It allows the learner to access the necessary information exactly when needed and to test the acquired knowledge through practical exercises and quizzes. This is possible thanks to the division of the teaching material into small "pills" that can be consumed in a few minutes, so that learners maintain high concentration and can focus on learning one topic at a time or on acquiring one skill at a time. Even from an economic point of view, microlearning is advantageous because it reduces the required time and resources to create content compared to traditional training formats.

At the same time, Artificial Intelligence (AI) is revolutionizing the e-learning sector as it offers companies powerful tools to automate educational content generation, quickly analyze data, and provide real-time support to students, thereby greatly increasing system efficiency and student performance, while reducing costs for companies.

1.2 Research objectives

The present thesis work studies this synergy between microlearning and generative AI with the aim of creating a web application that allows companies to easily convert complex business

content into concise and personalized learning modules, complete with descriptive slides and interactive quizzes for learner self-assessment. Before actually implementing the platform, an analysis of the state of the art in the e-learning sector was conducted, focusing on microlearning. The research highlighted that current tools offer limited customization and often require advanced technical skills for content creation. To address these limitations, the introduction of a chatbot assistant was proposed to enhance user experience and reduce manual workload.

In short, the main goal of the project is to develop an intelligent and user-friendly platform for enterprise knowledge transfer. This platform aims to reduce the manual effort required for learning content preparation through the use of AI models. It is designed to be accessible even to users without technical skills and integrates chatbot assistance to guide users during the creation and editing of slides. Additionally, it offers multilingual support to allow companies to effortlessly translate training materials.

1.3 Thesis structure

This thesis is divided into six chapters. After the introduction, the second chapter offers an overview of the state of the art, examining the benefits and the limitations of microlearning, the main technologies currently used in corporate training, and the potential offered by Artificial Intelligence tools in the educational field. The third chapter describes the design of the platform, defining the functional and non-functional requirements, the user personas, and use cases; after an overview of the AI techniques currently used for automated content generation, the selection of the adopted software technologies is presented. The fourth chapter illustrates in detail the implementation of the system, analyzing the developed functionalities, the workflows, and the architectural logics underlying the prototype. The fifth chapter is dedicated to the experimental evaluation of the platform: the manual test results are presented, and the generated outputs are analyzed to demonstrate the capabilities of the system and identify its strengths and limitations. After the first phase of manual testing, a planned automated test implementation is also presented. Finally, the sixth chapter offers a conclusive reflection on the results achieved, highlighting strengths, limitations, and possible directions for future development of the application.

This path aims to demonstrate how the integration of microlearning and Artificial Intelligence can be a concrete, innovative, and effective solution to address the challenges of modern corporate training, contributing to the improvement of knowledge transfer processes in increasingly complex and interconnected organizations.

Chapter 2

2. STATE OF THE ART

This chapter emphasizes the significant role of microlearning training in boosting learner performance across both professional and educational environments. In particular, the first paragraph introduces the concept of microlearning, providing the definition and the context and then focusing on its benefits and limitations. The second and third paragraphs examine the current technologies used for enterprise training purposes and the integration of Artificial Intelligence in e-learning. The fourth paragraph, finally, illustrates the standards and formats used for digital training.

2.1 Microlearning definition and context

Microlearning, also known as bite-sized learning, is a teaching strategy designed to deliver content in easily digestible chunks, typically through digital platforms [1]. This method focuses on one specific concept or skill at a time, enabling learners to assimilate “pills” of knowledge easily. The small learning units involve learning activities that can be completed in a matter of minutes (usually, a session lasts between 2 and 10 minutes), and can include videos, slides, quizzes, infographics, games, diagrams, interactive elements, simulations, or short reading materials to make learning more engaging and provide learners with quick and relevant information they can apply immediately.

This approach is particularly useful in an enterprise environment for staff training or rapid skills development. It also proves valuable in educational settings, such as schools or universities, where microlearning courses can complement traditional ones, offering a more flexible and dynamic study experience.

An extensive review of literature conducted through the *Scopus* database and *Google Trends*, *Leong et al.* [2] found that microlearning, although a relatively new concept, is emerging as an educational tool of global significance and has the potential to soon become an established and significant trend.

2.1.1 Benefits of microlearning

One of the defining characteristics of the microlearning approach is its on-demand nature, which aligns closely with the concept of "Just-In-Time" (JIT) knowledge [3]. JIT knowledge, defined as “delivering the right knowledge at the right time,” emphasizes the practical value of microlearning, enabling learners to access specific content whenever and wherever needed [4], also because the content is accessible via mobile devices. This flexibility ensures that learning can be tailored to individual schedules, making it especially useful for professionals managing

a busy workload or students requiring immediate clarification on a topic. It is clear, therefore, that in work situations with time constraints, this flexibility is particularly advantageous compared to traditional long and demanding training courses [5].

Several empirical studies show that learning in short and targeted units facilitates more effective learning. As an illustrative example, *Gross et al.* [6] conducted experimental studies that examined the impact of microlearning in the context of Crew Resource Management (CRM) training. In this study, participants were divided into two groups: one viewed a concise practical video demonstrating CRM principles in action, while the other watched a traditional lecture on the same topic. The group engaged with microlearning-based courses not only demonstrated better knowledge retention immediately after the training, but also retained that knowledge two weeks later. This effect is largely due to the way microlearning reduces cognitive overload by presenting content in manageable and focused chunks, allowing learners to internalize one concept at a time.

The ability to retrieve content at the exact moment of need improves not only knowledge retention but also supports the immediate application of new skills in real-life contexts. Studies conducted by *Branzetti et al*. [7], *Cheng et al.* [8], and *Gross et al.* [6] highlight how microlearning promotes skill acquisition more effectively than traditional methods.

Closely linked to flexibility is the higher level of learner engagement that microlearning fosters. Research by *Sawarynski et al.* [9] examined the implementation of an online module system, and, since its introduction, students were able to personalize their learning paths through a library of microlearning modules. The impact was significant: the percentage of students engaging with the modules rose from just 10% to 71% in two years. This data highlights how short and interactive content is more effective in keeping the attention of students than traditional long sessions. In addition, microlearning makes use of multimedia resources (such as videos, quizzes, animations, and simulations), which contribute to making the learning experience more dynamic and stimulating. This type of interactive approach stimulates learners' motivation and also facilitates the immediate application of the knowledge learned, strengthening learning through practical exercise and real-time feedback. Studies [5], [10] confirm that learners report greater satisfaction and motivation when interacting with microlearning content, a crucial factor in professional training contexts where intrinsic motivation plays a major role in continuous learning.

Even from an economic point of view, the microlearning approach presents many advantages. Due to the fact that microlearning modules are relatively short and focused, they require less time and fewer resources to produce compared to traditional training formats. This makes them inherently more economical, reducing costs associated with physical infrastructure, instructor time, and printed materials [11], [12]. This economic advantage makes microlearning particularly attractive to businesses looking for scalable and efficient employee training solutions. According to findings in [1], companies can reduce overall training costs while still achieving solid educational outcomes, thus maximizing return on investment.

Furthermore, microlearning offers opportunities for personalization, which is increasingly recognized as a key factor in successful education and training. Content can be tailored to the specific needs, goals, and learning styles of individual users, often with the support of AI-based tools. Several studies confirm the validity of this approach. *Hamdan Alamri et al.* argue that to better engage students and improve knowledge retention, it is necessary to consider individual preferences and adapt learning content accordingly [13]. In particular, adjusting the complexity,

format, and pace of lessons enables the creation of personalized experiences that maximize effectiveness. These adaptive capabilities also promote inclusiveness and accessibility, making educational materials more widely usable and equitable.

Another significant advantage of this methodology is the immediacy with which students can put the knowledge they have just acquired into practice. This derives from the fact that the small learning modules are focused on developing a single competency or skill at a time. This targeted method not only reinforces retention but also leads to measurable performance improvements. Studies [14] highlight the benefits of this immediate application, particularly in dynamic business settings where employees must constantly adapt to evolving tools, processes, or market conditions. By integrating learning directly into daily tasks, microlearning fosters a seamless transition from education to execution, making it an essential strategy for modern workplaces.

In these sectors where knowledge changes quickly (such as technology, healthcare, and finance), microlearning is a powerful tool for continuous education, satisfying the need of employees to update their skills frequently and stay informed about emerging trends to remain competitive. The microlearning short lessons help to reinforce what has already been learned and to introduce new information quickly and in a non-invasive way. The research [5] emphasizes its value in competitive and dynamic business environments where having access to updated information can make the difference.

Microlearning also has positive effects on self-confidence and self-perception, particularly because it enables learners to achieve small frequent successes that reinforce their sense of competence and progress [15], [16]. Breaking information into simple units reduces cognitive overload and allows learners to gradually increase confidence in their abilities.

In summary, microlearning is perfectly suited to the needs of modern businesses and the behaviors of today's workers. It is flexible, engaging, and helps people remember information better, thus promoting continuous learning. Furthermore, it is affordable, easy to adapt, and allows immediacy in knowledge application, making it very effective for companies. As industries continue to evolve and the demand for personalized training solutions increases, microlearning is set to become always more important. The use of Artificial Intelligence will make it even more powerful, offering customized and easily accessible learning paths. Companies that choose this path can count on employees who are better prepared, motivated, and ready for change.

2.1.2 Limitations of microlearning

Despite its growing popularity and proven benefits in modern educational and corporate contexts, microlearning is not without limitations. While it offers a flexible, efficient, and engaging learning experience, this approach is not universally suitable across all learning scenarios. The effectiveness of microlearning depends heavily on the nature of the content, the complexity of the subject matter, the learning objectives, and the learners' technological proficiency.

Microlearning is most effective when applied to topics that can be broken down into self-contained units, such as procedural knowledge, terminology, or best practices. However, for subjects that are inherently complex, abstract, or interdependent (such as strategic decision-making, critical thinking, or comprehensive systems analysis), microlearning may not provide

sufficient depth or scaffolding [17]. These topics often require extensive exploration, more in-depth explanations, iterative learning cycles, and sustained engagement, which are better facilitated through traditional learning methodologies [18]. The microlearning model may oversimplify such topics, leading to a fragmented learning experience and superficial understanding.

While microlearning helps maintain focus for learners with short attention spans and is very helpful for those with limited time, sustaining long-term motivation can be difficult. Learners who prefer immersive educational experiences may find microlearning's brevity disengaging, particularly when rich discussions, reflective practices, and deeper analysis are minimized or absent. Furthermore, completing short modules may give a false sense of accomplishment, where learners believe they have mastered a subject without sufficient reinforcement or application. This illusion of mastery can discourage further study, potentially compromising long-term retention and skill development [17].

Additionally, not all learners respond equally well to technology-driven approaches. A significant proportion of the workforce, particularly those less familiar with digital tools or online learning platforms, may experience discomfort, skepticism, or disengagement when confronted with microlearning formats. This resistance can be attributed to several factors, including lack of exposure to digital learning environments, low confidence in navigating AI-based interfaces, or a simple preference for the interpersonal and structured dynamics of traditional classroom settings [19], [20]. Some learners may perceive microlearning as overly simplistic or lacking the depth and rigor of conventional instruction. Educators and corporate trainers, too, may exhibit hesitancy in adopting microlearning, particularly if they are accustomed to established pedagogical methods that offer predictable outcomes and clearly defined instructional sequences [10].

Although microlearning is intended to reduce cognitive load by delivering content in manageable segments, it can paradoxically contribute to cognitive overload if not carefully designed. Densely packed modules or rapid sequencing without sufficient time for reflection may overwhelm learners. Consuming multiple modules in succession can result in information overload and mental fatigue, diminishing the effectiveness of the learning process. Therefore, pacing, structure, and opportunities for consolidation are critical in preventing learner burnout [17].

Another important issue is related to accessibility and technological equity, which emerges from the fact that the microlearning approach is based on digital platforms (such as Learning Management Systems and web applications). Learners who do not have consistent internet access may not be able to fully engage in microlearning programs and this deepens educational gaps. Technical problems (such as poor interface design, app malfunctions, or connectivity troubles) can disrupt the learning experience, leading to frustration and dropout. These challenges underscore the importance of designing intuitive and inclusive platforms that take into account different levels of technological access and proficiency [17].

Another important limitation lies in the perception that microlearning, especially when delivered via AI-powered web platforms, may diminish opportunities for social interaction and peer-to-peer learning. In traditional settings, learners benefit from discussions, group activities, and real-time feedback, which foster deeper understanding and collaborative skills. Microlearning environments must therefore evolve to incorporate interactive elements, such as discussion boards, chatbots, collaborative tasks, or gamified challenges, to mitigate this

perceived isolation and enhance learner engagement. In designing an AI-powered application for business knowledge transfer, it becomes essential to consider these human-centric elements and ensure that the platform is not only efficient but also intuitive, inclusive, and socially responsive [17].

In summary, while microlearning offers significant advantages in terms of flexibility, adaptability, and scalability, especially when integrated with AI, it is not a solution that works in every context. The successful application of this approach requires a good understanding of the content domain, learner profiles, and organizational goals. Combining the microlearning approach with traditional learning methods can create a balanced system that optimizes training outcomes. With this hybrid model, companies can address a wider range of learner needs and preferences, ensuring that technology enhances rather than replaces the foundational principles of effective learning. Therefore, the proposed AI-driven web application must be designed with versatility and user-centricity in mind, enabling it to adapt to different learning contexts while maximizing the benefits of microlearning for business knowledge transfer.

2.2 Existing technologies for enterprise training

In recent decades, enterprise training methods have undergone profound changes due to the use of digital technologies to improve accessibility, skills development, knowledge retention, efficiency, and personalization of learning experiences, while reducing the time and cost associated with traditional training.

This chapter focuses on presenting the commonly used technologies and AI tools for corporate knowledge transfer, paying particular attention to their strengths and limitations.

2.2.1 E-learning platforms

E-learning refers to education delivered via digital means. It offers a fast and effective way to deliver and share knowledge with learners across the globe [21]. Services such as *LinkedIn Learning*, *Coursera for Business*, and *Udemy Business* offer curated video courses on a wide range of professional topics, including technical skills, leadership, and compliance. These platforms provide high-quality content and support self-paced learning. However, they do not facilitate company-specific knowledge transfer and lack tools for adapting content to the internal processes, tools, or terminology of a given organization.

2.2.3 Microlearning platforms

Microlearning platforms, like *Axonify*, *LearnUpon,* and *Spekit* [22], just to name a few, deliver short focused learning modules designed to be completed quickly. These tools are better suited to modern attention spans and mobile-first use cases. Integrating microlearning content into traditional e-learning platforms allows for more flexible and accessible learning experiences, meeting time constraints and specific needs of employees [23]. However, they still rely largely on manual content creation and often lack AI-driven automation or advanced personalization.

2.2.3 Learning Management Systems

Learning Management Systems (LMS) offer dynamic platforms that support both synchronous and asynchronous learning environments, thus becoming an integral part of the structure and delivery of online education. These platforms facilitate the distribution of course materials, monitoring of student progress, and promotion of collaborative learning through discussion forums and group activities. The flexibility of LMS platforms allows for tailoring their use to

specific instructional goals and technological capacities, contributing to improving learner outcomes and satisfaction [24].

LMS platforms typically are designed to support SCORM [25] and xAPI [25] standards, allowing for seamless content integration and tracking of learner progress across multiple systems. Despite these technological capabilities, LMS may not be well-suited for developing skills that require direct interaction, such as oral communication, and may contribute to feelings of isolation due to limited face-to-face interaction and social presence in fully virtual settings [26]. Nonetheless, platforms such as *Moodle*, *SAP Litmos*, *Cornerstone*, *Blackboard*, and *TalentLMS* continue to represent essential tools for corporate training in many organizations.

2.2.4 Authoring tools

The use of e-learning authoring tools has become a fundamental component of digital learning strategies. These platforms allow businesses to develop interactive training content tailored to contemporary educational demands. Authoring tools vary in complexity and purpose, from simple, drag-and-drop interfaces to advanced platforms requiring programming skills, allowing companies to choose among them based on their technical capabilities and instructional goals. Solutions like *Articulate*, *Adobe Captivate*, and *GLO Maker* are designed to produce SCORM-compliant modules, ensuring integration across different Learning Management Systems [27]. By simplifying the content creation process as well as supporting reusability and adaptive learning paths, these tools reduce training costs and help make knowledge transfer more effective.

2.2.5 AI-powered tools and chatbots

Emerging tools are beginning to incorporate Artificial Intelligence to automate parts of the content creation and personalization process [28]. For example, some platforms use Natural Language Processing (NLP) to generate quizzes from text, while others integrate AI chatbots for learner support. Tools like *Docebo* and *Tovuti LMS* have started incorporating machine learning for learner analytics and content suggestions.

2.2.6 Key limitations of existing technologies

While the technologies discussed above offer numerous benefits to businesses, they often suffer from:

- Significant costs associated with installation and maintenance;
- High complexity in design tools, making them difficult for content developers to master;
- Limited capacity to tailor learning experiences to individual needs;
- Lack of integration across content creation, delivery, and performance tracking;
- Inadequate support for real-time assistance or contextual learning.

These limitations underscore the importance of adopting an integrated, intelligent, and user-centric approach, which the proposed AI-enhanced microlearning platform aims to address by combining automated content creation, interactive chatbot support, and modular learning units in a cohesive digital environment.

2.3 Application of Artificial Intelligence in e-learning

AI is transforming the e-learning landscape by introducing intelligent systems that personalize learning experiences, streamline administrative processes, and support adaptive learning. This section examines how AI is being integrated into digital learning environments, highlighting its benefits, challenges, and future prospects.

2.3.1 Personalized and adaptive learning

AI facilitates the development of tailored learning experiences by analyzing individual learner data to customize content and learning pathways. Methods such as content recommendation, curriculum sequencing, and automated feedback are leveraged to align instruction with each learner's unique needs.

For instance, content recommender algorithms use collaborative filtering to suggest learning materials aligned with a learner's preferences, knowledge level, and learning goals. Combining these approaches in hybrid recommender systems results in increased engagement, improved outcomes, and more effective adaptive learning experiences [29]. Additionally, AI-powered platforms dynamically adjust content presentation based on learner responses, fostering deeper understanding and retention.

2.3.2 Intelligent Tutoring Systems and Virtual Assistants

AI-based Intelligent Tutoring Systems (ITS) and Virtual Teaching Assistants (VTAs) are commonly used to provide real-time support and feedback to learners. These systems use Natural Language Processing to understand and respond to student requests, generate quizzes, flashcards, and offer personalized learning paths, tailored to individual learning needs [30]. They are also used to automatically translate content, adapting it to learners from different linguistic regions and supporting the global deployment of e-learning programs.

Intelligent Tutoring Systems incorporate interactive elements that simulate human-like social interactions, fostering greater learner engagement and improving the effectiveness of the educational process.

2.3.3 AI in learning analytics and assessment

Artificial Intelligence is rapidly reshaping e-learning, particularly in the domains of Personalized Learning (PL) and Adaptive Assessment (AA), by leveraging insights from cognitive neuropsychology. AI-driven systems enable the customization of educational experiences by analyzing learners' cognitive profiles, emotional states, and performance data to adapt content, feedback, and assessments in real time. This integration fosters increased engagement, motivation, and learning efficiency, as AI tailors instruction to individual needs, learning styles, and cognitive abilities.

In addition, AI facilitates learning analytics by monitoring student progress, analyzing learning behaviors, and predicting academic challenges. These insights support timely interventions to assist at-risk students and enhance overall educational outcomes.

AI also plays a key role in assessments by automating grading and delivering instant feedback, which allows educators to focus more on interactive teaching and student engagement. Adaptive systems further enhance this process by dynamically adjusting task difficulty and optimizing cognitive load and memory retention through the use of neurophysiological data.

Despite these advancements, challenges such as algorithmic bias, data privacy, and equitable access remain critical concerns. The literature underscores the importance of empirical validation and ethical frameworks to ensure that AI-enhanced learning environments are not only effective but also inclusive and fair.

As AI continues to evolve, its potential to revolutionize education through cognitively informed, personalized, and adaptive learning systems remains both promising and profound [31].

2.3.4 Enhancing accessibility and inclusion

The integration of Artificial Intelligence with inclusive design principles offers powerful solutions to enhance accessibility, ensuring that all learners have equitable access to educational resources, also those with disabilities.

Adaptive learning systems customize content based on individual learning styles and needs, increasing engagement and comprehension. Natural Language Processing (NLP) and speech recognition tools are able to convert speech to text and vice versa, helping students with hearing or speech impairments, while other AI tools can generate descriptive audio for visually impaired students. AI-powered chatbots and Virtual Assistants can support learners with learning difficulties, helping them keep pace with their peers, providing personalized guidance, answering questions, and helping navigate complex concepts throughout the learning process. Therefore, AI facilitates compliance with accessibility standards, promoting inclusivity across educational platforms [32].

Despite these benefits, challenges such as data privacy concerns, technical limitations, and implementation costs must be addressed to fully realize AI's potential in inclusive education.

2.3.5 Recent trends and innovations

Recent literature highlights the rise of *ChatGPT*-style conversational agents in e-learning for corporate and higher education. For instance, *Bettayeb et al.* conducted a systematic review of *ChatGPT* in education, highlighting numerous benefits such as personalized assistance, instant feedback, and improved accessibility, all contributing to greater learner engagement and improved educational outcomes [33]. Similarly, *Choudhary et al.* surveyed the use of *ChatGPT* in corporate training, concluding that AI-driven chatbots significantly enhance learner satisfaction by providing on-demand, personalized guidance and automating routine support tasks [34]. A large meta-analysis conducted by *Wang et al.* found that the use of *ChatGPT* was associated with a large improvement in student performance and a moderate increase in their attitudes and higher-order thinking [35]. Collectively, these studies suggest that conversational AI has the potential to simulate human-like tutoring and engage learners dynamically.

Generative models like *DALL·E* are increasingly used to produce educational images and infographics on demand. Empirical studies suggest that tailored visuals can aid learning by leveraging visual memory. For instance, *Ichimura et al.* created thousands of synthetic medical images for training ophthalmology students, finding that an image-based learning quiz significantly outperformed a traditional video lecture for identifying eye tumors [36]. Their analysis showed that trainees exposed to diverse AI-generated visuals achieved higher diagnostic accuracy and faster decision times than those who only watched a narrated slide presentation. This suggests that carefully designed AI visuals can engage pattern recognition and reinforce learning better than text alone. Educators are therefore advised to use AI-

generated images in supportive roles (e.g., charts, diagrams, illustrative pictures) that are directly in line with learning goals, while avoiding random or decorative images that may distract rather than help. In short, generative imagery tools offer scalable ways to produce tailored educational graphics, and preliminary evidence indicates they can enhance comprehension and recall in technical subjects.

Meanwhile, AI-driven Text-To-Speech (TTS) and Automatic Speech Recognition (ASR) technologies are transforming both accessibility and assessment in e-learning. High-quality neural TTS can generate realistic narrations for any text, enabling audio-based microlearning (podcasts, voice-over slides) without the need for human voice actors. Such dual-mode presentation (read-along audio plus on-screen text) supports Universal Design for Learning (UDL) by meeting diverse learner preferences and needs. For example, TTS narration allows learners with visual impairments or reading difficulties to access content more easily and can boost retention by combining auditory and visual input. On the other side, ASR technologies are increasingly used in language training and assessment. *Liu et al.* reported positive feedback from English learners using ASR tools in speaking practice exercises; both students and instructors appreciated the automatic scoring and found it helpful for assessing speaking proficiency [37]. In another controlled experiment, *Wilschut et al.* compared a traditional typing-based vocabulary app with a speech-based version using ASR. They found that speaking practice using ASR achieved equivalent learning gains to typing practice, and that an intelligent scheduling algorithm (based on ACT-R memory models) improved vocabulary recall in both modes [38]. These findings suggest that ASR can effectively support scalable conversational exercises, such as pronunciation or oral exams, while maintaining learning outcomes comparable to traditional methods. In summary, voice technologies improve accessibility and flexibility. TTS systems provide automated narration and audio course units (benefiting all learners, especially those with reading disabilities), while ASR can support novel interaction modes (e.g., voice quizzes and automated oral assessments). Early studies demonstrate that these tools are well-received by learners and can maintain, or even enhance, learning performance.

2.3.6 Ethical considerations and challenges

The deployment of AI-driven tools (adaptive tutors, intelligent assessments, language models, etc.) in e-learning promises personalization and efficiency, but also raises several ethical concerns that must be carefully addressed to ensure fairness, transparency, and respect for learners' rights [39], [40], [41]. Key ethical considerations include:

- Data privacy and security: AI systems rely heavily on collecting and analyzing vast amounts of learner data, including sensitive personal information. Ensuring robust data protection mechanisms and compliance with privacy regulations (such as GDPR) is critical to safeguard learners' confidentiality and prevent misuse of data [42];
- Bias mitigation and fairness: AI systems must be designed to avoid reinforcing existing biases in educational content and assessment [43]. Systems trained on biased models should incorporate bias detection and regular auditing to ensure equitable recommendations and assessments [31];
- Transparency and explainability: Learners and educators often lack a clear understanding of how AI systems make decisions. Students should know what data is collected and how AI-driven feedback is generated. Ethical AI in education should

prioritize explainability to build trust and allow users to understand and challenge AI-driven outcomes [44], [45];

- Accountability and governance: Clear policies and oversight mechanisms are needed. Educational organizations are encouraged to establish governance to monitor AI use and assign responsibility [31];
- Equity and accessibility: AI tools should ensure equal opportunities. In practice, this means addressing technology gaps and designing non-discriminatory algorithms so that all students or employees benefit equally [31];
- Human oversight: AI should complement and not replace human judgment [39]. Experts recommend keeping instructors or managers in the loop (for example, faculty reviewing AI-graded work or trainers supervising AI-driven learning paths) to maintain ethical standards.

2.4 Standards and formats for digital training (SCORM, xAPI)

SCORM (Sharable Content Object Reference Model) is the older, widely-adopted e-learning standard originally developed by the *ADL Initiative* for the *U.S. Department of Defense*. It defines how course content is packaged (as "SCOs") and how learning modules communicate with a Learning Management System via a *JavaScript API* [46]. SCORM was created to promote interoperability and reusability, allowing courseware developed in the SCORM format to be efficiently shared across different LMS platforms in a standardized way [46]. In practice, SCORM packages are launched within an LMS, which tracks course status, completions, and simple scores via the SCORM runtime data model [46].

However, SCORM is inherently LMS-centric. *Panagiotakis et al.* note that SCORM is deeply integrated with the LMS and cannot operate independently, meaning content must be delivered within a compliant LMS in order to record any data [47]. Its tracking model is also limited: SCORM can record basic metrics such as course completion, test results, or time spent, but it soon became clear that these capabilities were insufficient to meet evolving learning and data tracking needs [48]. In short, SCORM's architecture binds content to an LMS and only captures formal course-based interactions.

By contrast, xAPI (the Experience API or Tin Can API) was introduced in 2013 to overcome these limitations. It is a platform-neutral learning data specification designed to record any learning experience [48], [49]. Technically, xAPI uses a RESTful web service model: learning activities send JSON-formatted statements (actor-verb-object triples) over HTTP to a dedicated Learning Record Store (LRS) [49], [50]. The LRS is a repository that stores all xAPI statements; content and devices (from mobile apps to simulations and games) act as Learning Record Providers (LRP) that issue statements. *Panagiotakis et al.* describe the LRS as more than just a data store; it also serves as a source for data aggregation and analytics, capable of ingesting statements from any source in a standardized format [47]. In short, xAPI decouples activity tracking from any specific LMS and allows learning data to be collected from diverse tools and contexts.

2.4.1 SCORM use cases, advantages, and limitations

SCORM has historically been used for formal online courses in both corporate and higher-education LMSs. A typical use case is compliance or certification training delivered as packaged course modules in a company LMS or university learning platform. Due to its status

as the de facto standard for interoperability, SCORM is supported by most commercial and open-source LMSs [46], [51].

One of SCORM's biggest strengths is its maturity and broad support. It established a common model that enables content creators to distribute learning materials across a wide range of LMS platforms efficiently [46]. Its tracking and sequencing rules (especially *SCORM 2004*) allow defining simple learning paths and completion rules. Many organizations still rely on SCORM because their existing training content and systems were built around it.

However, SCORM also has constraints. Since it requires content to run within an LMS, it cannot track learning that occurs outside that environment [47]. It only records basic data (completion, quiz scores, session time) and cannot capture rich behavioral data or informal learning activity. For this reason, SCORM quickly proved to be too limited for modern learning needs [48]. Its dependence on packaged course files (typically ZIP archives) can be cumbersome for content authors. Furthermore, once content is launched, the SCORM data remains locked in the LMS, with minimal support for integration into modern analytics tools or cross-system data sharing. Ultimately, SCORM's LMS-centric design makes it inflexible and unable to exploit new learning modalities (mobile apps, games, social learning, etc.) [47].

2.4.2 xAPI use cases, advantages, and limitations

xAPI has been applied in corporate and academic contexts where richer data is needed. For example, xAPI can record on-the-job training activities, simulations, discussions, or offline learning. According to an ADL report, xAPI enables the tracking of learners as they carry out work-related tasks, generate outputs, interact with others, collaborate, and participate in any other online activities [48]. In corporate learning, xAPI is used to measure informal and social learning, compliance exercises in realistic simulations, or mobile app usage. In education, xAPI supports learning analytics systems by aggregating diverse student activities into a common record [47], [49].

The core advantage of xAPI is its flexibility. Unlike SCORM, xAPI is not limited by content format or tied to a specific LMS: any learning action (watching a video, reading a PDF, conducting a lab, participating in a forum) can be captured as an xAPI statement [47], [49]. As *Panagiotakis et al.* demonstrate, xAPI enables the tracking of diverse activities across multiple platforms, with statements originating from sources such as websites, mobile apps, simulators, or virtual games [47]. Another major advantage is xAPI's support for offline learning: learners can complete activities without an active internet connection, and once reconnected, the system transmits the stored statements to the Learning Record Store [47]. Thanks to its structured statement format, xAPI integrates seamlessly with Business Intelligence (BI) tools, enabling the generation of detailed reports, an aspect increasingly emphasized by researchers for its potential in advanced analytics [47]. Another benefit is connectivity: multiple LRSs (or an LMS with embedded LRS) can share xAPI data, enabling federated or lifelong learning records [47]. In summary, xAPI provides much richer, granular data and frees learning data from the confines of a single LMS [47], [49].

Despite its power, xAPI presents some challenges. Its flexible data model, while powerful, requires careful management of semantics. As a technical report warns, xAPI does not enforce consistent use of terminology across systems. Without standardized vocabularies, statements generated by one system may be difficult for another to interpret [49]. To ensure meaningful data exchange, organizations must adopt shared xAPI profiles or schemas. There are also

practical considerations: every xAPI deployment requires an LRS infrastructure (though many LMS platforms now include an LRS). Adoption has been relatively slow, with many organizations still relying on SCORM-based systems. As a more recent standard (introduced in 2013 and approved by IEEE in 2020 [50]), xAPI may not be fully supported by older tools and content, which often need updates to ensure compatibility.

2.4.3 Comparison of technical architectures

SCORM packages content (HTML/*Java*/video/etc) with a manifest that describes SCOs. When a learner launches a SCO in an LMS, the LMS provides a *JavaScript* API endpoint ("API adapter") that the SCO calls to initialize, and to get/set runtime data (*CMI -Computer Managed Instruction*- data model) during the session [46]. At session end or on calls, the SCO reports data (e.g., lesson status, score) back to the LMS API, which the LMS stores. The learner's interactions are thus tracked only while the content is hosted in that LMS. The architecture is tightly coupled: without an LMS (or a SCORM player), the content cannot send or store any data [47].

In xAPI, there is no single "master" environment. Instead, any learning application (web app, mobile app, simulator, etc.) can act as an xAPI Learning Record Provider (LRP) and send statements over the network to one or more LRSs [47], [49]. xAPI uses a RESTful API with JSON: each statement is a small JSON document (who did what, when, and optional context). For example, an LMS might send "*Alice completed Quiz 1*" when a quiz is done, while a mobile app might send "*Alice watched Video 3*" separately. The LRS exposes REST endpoints (via HTTP GET/POST) to receive and store these statements [50]. Any system or tool can retrieve learning data from the LRS via HTTP as well. This distributed architecture allows xAPI to decouple content generation from storage: learning content does not need to know where data is going ahead of time [47]. *Panagiotakis et al.* illustrate that an LRS can store learning activities from a wide range of sources; whether the statement comes from a game, a mobile application, or a webpage, it can all be captured and stored using the same format within the LRS [47]. Crucially, xAPI supports multiple LRSs, cross-system sharing, and offline batch updating (statements can be sent whenever a connection is available [47]).

The table below highlights the main distinctions between SCORM and xAPI in terms of content delivery, data storage, data scope, and interoperability, illustrating how xAPI offers greater flexibility and extensibility for modern learning environments:

Feature	SCORM	xAPI
Content delivery	Packaged courses (ZIP) launched via LMS	Content and activities can run anywhere; statements sent via REST
State storage	LMS internal database (proprietary to each platform)	Learning Record Store, either standalone or embedded in an LMS
Data scope	Limited to course/session-level data (completion, score, time)	Any learning event (activities, assessments, clicks, etc.)
Interoperability	Based on legacy specifications with a fixed data model	JSON-formatted statements; use shared vocabularies and xAPI profiles for semantic interoperability

Table 2.1: Key architectural differences between SCORM and xAPI

2.4.4 Applications in corporate and academic settings

In corporate training, SCORM has long been the norm for standard e-learning modules (compliance, product training, certifications). xAPI is now gaining traction for more dynamic scenarios: for example, recording simulations or informal social learning (forums, peer coaching) that occur outside the LMS. Industry reports note that xAPI enables flexible tracking of a virtually limitless range of learning activities [48], so learning managers can connect training to business metrics. Some corporations implement xAPI in custom apps or immersive training, then feed all data to a corporate LRS for analytics.

In higher education, LMS platforms historically relied on SCORM for interactive content. New research on learning analytics encourages institutions to capture data that extends beyond the traditional confines of the university [31]. In this sense, xAPI is interesting because it allows the integration of data from multiple sources, making it useful for university-scale analysis projects. A recent study has identified SCORM and xAPI as key standards for interoperability in LMSs [51]. Some universities are experimenting with using xAPI to track student engagement across different platforms, but full institutional adoption is still limited: most formal courses continue to rely on LMS and SCORM, while xAPI is mostly used in research projects or blended learning initiatives.

Recent studies emphasize that SCORM and xAPI fulfill different needs. SCORM remains a stable standard for conventional, LMS-hosted e-courses [46], yet it falls short in capturing the diverse cross-platform interactions of modern learning. Conversely, xAPI was specifically designed to overcome SCORM's limitations by focusing on learners and their activities [49]. *Panagiotakis et al.* conclude that xAPI represents a much broader technology compared to SCORM, operating independently of LMSs and supporting the vision of "lifelong learning" [47]. The drawback is that xAPI requires a new Learning Record Store infrastructure and careful semantic governance, whereas SCORM leverages existing LMS databases and workflows. In practice, many organizations blend both standards: using SCORM for legacy courses and adopting xAPI for advanced learning experiences. Several authors note that the emerging CMI5 standard (which combines xAPI with launch rules) aims to facilitate migration [46]. Ultimately, the literature suggests choosing the standard based on use case: SCORM for traditional course content delivery [46], and xAPI for broad flexible tracking of distributed learning activities [47], [48].

Chapter 3

3. PLATFORM DESIGN AND ARCHITECTURE

This chapter outlines the key requirements and design considerations guiding the development of the AI-based microlearning platform for enterprise knowledge transfer. The goal is to define a solution that can efficiently support Subject Matter Experts (SMEs) and employees during the creation, customization, and fruition of microlearning content, without requiring advanced technical skills. For this purpose, the system uses cutting-edge AI models, offering a high degree of usability and ensuring compatibility with real-world enterprise contexts.

The first paragraph details the functional and non-functional requirements of the platform. These requirements describe what the system must do (such as enabling AI-assisted content generation, user interaction through a chatbot, and multilingual support) as well as qualities it must possess, including usability, performance, scalability, reliability, availability, maintainability, and security.

The second paragraph presents user personas and use cases, which help contextualize the platform's requirements through realistic user behaviors and scenarios. This section defines typical users (such as Training Managers and Domain Experts) and explores how each interacts with the platform to accomplish specific goals.

The third paragraph surveys the core AI techniques used for automated content generation. These include methods for content summarization, document parsing and preprocessing, slide and quiz generation, and multilingual support. This section provides a technical foundation for understanding how AI capabilities will be integrated into the platform to enhance user experience and reduce manual workload.

The fourth paragraph presents an in-depth overview of the technology stack chosen for the development and deployment of the system. Each subsection is dedicated to a key component of the architecture, detailing the reasons behind the selection of specific tools, frameworks, and services.

3.1 Functional and non-functional requirements

In the development of modern educational platforms, it is essential to clearly define the system requirements to ensure alignment with user needs and technological capabilities. This paragraph presents both the functional and non-functional requirements that guided the design and implementation of the proposed solution. Functional requirements focus on the core features that the platform must provide to support content creation, management, and personalization, while also enabling intelligent assistance through AI-driven tools. By outlining

these requirements, we establish a foundation for building an intuitive, flexible, and intelligent learning environment that can adapt to diverse user scenarios and promote efficient knowledge delivery.

3.1.1 Functional requirements

The platform has been designed to support users throughout the entire process of learning content creation, from initial input to final delivery, combining automation, flexibility, and intelligent assistance. One of the core functionalities is content upload and management. Users can upload multiple document formats, including URLs, PDFs, *Word* documents (DOCX), *Excel* spreadsheets (XLSX), and plain text files (TXT). Once uploaded, the system automatically extracts the textual content from these sources. Users can manage the sources and, in particular, they can add or remove them, edit the name, title, and description.

In cases where users do not upload any input or wish to supplement their material with external information, the platform also supports online resource retrieval. Leveraging the *Tavily* search engine, the system can automatically perform searches on the web to find relevant and credible content based on user queries or contextual needs. This ensures that learning resources are enriched with updated and reliable information, broadening the scope of the educational material generated.

A central feature of the platform is its AI-driven content generation capability. Using Natural Language Processing and Large Language Models (LLMs), the system is able to analyze raw input and transform it into structured learning modules. These modules include both slide-based presentations and interactive quizzes for learner self-assessment. Users can choose between different levels of summarization (ranging from detailed explanations to concise overviews) depending on their learning goals or audience. The quiz generation component produces questions in both multiple-choice and single-choice formats, encouraging engagement and reinforcing understanding through interactive learning.

Beyond automatic generation, the platform emphasizes user control and personalization. All generated content can be edited to suit individual preferences or institutional standards. Users can modify slide titles, revise descriptions, add and remove slides, as well as split or merge them. The structure of the content can also be refined (for example, by converting paragraphs into bullet points, emphasizing keywords, or reorganizing sections for improved clarity). Likewise, quizzes can be customized: users can add new questions, remove or edit existing ones, and edit answer options, allowing for highly tailored experiences.

To make the platform even more accessible and user-friendly, an integrated AI-powered chatbot provides ongoing assistance. This virtual assistant is capable of understanding natural language queries and can guide users through various tasks, such as managing uploaded resources, editing content, generating new modules, or navigating the platform's features. This conversational support system enhances usability, particularly for users who may not be familiar with more technical tools or interfaces.

Lastly, recognizing the global nature of modern education and collaboration, the platform includes multilingual support. All content generated can be automatically translated into multiple languages, making it easier to share learning resources across different regions, teams, or linguistic backgrounds. This feature not only broadens the platform's reach but also promotes accessibility and knowledge sharing on an international scale.

In summary, the system brings together robust functionality, intelligent automation, and user-centered design to create a versatile learning content generation platform. Each feature (whether related to document handling, AI-powered synthesis, personalization, or multilingual support) contributes to an integrated workflow aimed at simplifying the creation of rich, engaging, and adaptable educational materials.

3.1.2 Non-functional requirements

In addition to the core functional capabilities, the platform must also meet a series of non-functional requirements that are crucial to ensuring a seamless, secure, and sustainable user experience. These requirements address aspects such as usability, performance, reliability, maintainability, and security (each contributing to the platform's overall effectiveness and long-term viability).

Usability stands as a cornerstone in the platform's design philosophy. The interface must be intuitive and user-friendly, enabling individuals of varying technical backgrounds to navigate and utilize the system. Whether the user is a corporate trainer or a learner, the interface should feel natural and require minimal instruction.

Equally important is the system's performance and scalability. Given the computational demands of AI-driven content generation and real-time online search integration, it is essential that the platform delivers fast response times. Users should experience minimal delay when generating slides or quizzes, retrieving external resources, or interacting with the AI assistant. Moreover, the platform must be designed to support multiple users simultaneously. As organizations grow and more users interact with the system at once, the underlying infrastructure should be capable of scaling accordingly, maintaining high performance even under increased load.

Reliability and availability are also critical aspects to consider, especially given the platform's goal of supporting continuous learning and uninterrupted content development. Technical downtimes or unexpected failures can significantly hinder user productivity and compromise trust in the system. At this stage, no specific mechanisms such as fault tolerance, server redundancy, or real-time monitoring have been implemented. However, introducing such measures in the future could prove highly beneficial. Ensuring high system availability would allow the platform to remain functional even in the event of partial outages, thus improving user experience and making the system more resilient to unforeseen issues. As the platform evolves and scales to accommodate a growing number of users and more complex use cases, integrating these kinds of reliability strategies could play a key role in maintaining performance and user confidence over time.

Maintainability is another key requirement, especially in a rapidly evolving technological landscape. The platform should be constructed using a modular architecture that supports easy updates and extensions. This modularity allows new AI models or features to be integrated without requiring major changes to the existing codebase, facilitating ongoing innovation while preserving system stability. Developers should be able to isolate and update specific components without affecting the rest of the system.

Security considerations are fundamental, particularly when dealing with sensitive user-generated content and external data sources. At a basic level, the platform must implement an authentication system to ensure that only authorized users can access its features. This protects

both the users and the integrity of the system itself. In addition, a dedicated middleware layer should act as a gatekeeper between the frontend and backend, filtering incoming requests and blocking any unauthorized access attempts from external sources. This not only safeguards the application from malicious attacks but also reinforces data privacy. File handling must be conducted with care, using secure protocols such as temporary storage and *base64* encoding for data transfer. These measures reduce the risk of data leaks and ensure that files are not permanently stored unless explicitly required by the user.

3.2 User personas and use cases

To better contextualize the platform's functional requirements and validate its design choices, it is essential to identify the typical user personas and analyze the core use cases they engage in. The platform is intended to support a broad range of enterprise users with varying levels of technical expertise, knowledge ownership, and learning objectives. This section defines two primary user personas: the Training Manager and the Domain Expert. Each persona represents a different perspective in the knowledge transfer lifecycle and contributes to a distinct phase of the microlearning workflow.

3.2.1 Training Manager

The Training Manager plays a key role in supervising the creation, delivery, and tracking of corporate training materials. Their main objective is to ensure that learning modules are aligned with the organization's objectives and are easily accessible to employees across different departments and locations. They interact with the platform to review AI-generated content before publication, ensuring that all microlearning materials meet quality and consistency standards. By streamlining the content approval process and maintaining alignment with internal guidelines, the Training Manager helps accelerate the launch of training programs while improving overall compliance and efficiency.

3.2.2 Domain Expert

The Domain Expert is a Subject Matter Specialist (e.g., engineer, HR lead, or sales trainer) who possesses deep knowledge in a specific business area but may not have formal skills in instructional design. Their main objective is to share their expertise, ensuring that their specialized knowledge is accurately represented. On the platform, the Domain Expert interacts by uploading documents (such as technical reports, procedures, or internal guides) and using the AI-powered assistant to automatically generate learning modules, including slides and quizzes. They then edit and refine the generated content through the chatbot. This process empowers experts to contribute directly to training materials, eliminates technical barriers, and accelerates the availability of content.

3.2.3 User profiles summary

Persona	Primary role	Main interactions with the platform	Expected benefit
Training Manager	Supervises learning delivery	Reviews, customizes, and approves content; monitors training progress	Ensures quality and alignment of learning programs
Domain Expert	Contributes domain knowledge	Uploads content; generates and edits modules using an AI assistant	Shares expertise efficiently without technical bottlenecks

Table 3.1: Use case interactions

By mapping these user personas and their respective behaviors, the platform's user experience and feature set can be tailored to better meet the diverse needs of stakeholders involved in corporate knowledge transfer.

3.3 Survey of AI techniques for automated content generation

AI-driven automation plays a foundational role in the development of modern microlearning platforms. Recent advancements in Natural Language Processing (NLP) and Large Language Models have enabled a variety of intelligent operations that streamline the transformation of raw enterprise content into interactive learning materials. This section provides an overview of the key AI techniques leveraged in the system for automating content generation.

3.3.1 AI for content summarization

In the proposed microlearning platform, automated content summarization plays a fundamental role because it enables the transformation of complex corporate documents into concise and easily consumable learning content. This capability is supported by advanced Natural Language Processing algorithms and is typically implemented using two primary approaches [52]:

- Extractive summarization: This technique focuses on selecting the most relevant sentences directly from the original text. The resulting summary preserves the exact wording and sentence structure. While this method is efficient and grammatically reliable, it may lack coherence and smooth transitions between the extracted segments;
- Abstractive summarization: Abstractive methods interpret the meaning of the text and generate entirely new sentences to convey the main ideas. Although this approach is more sophisticated and computationally demanding, it produces summaries that are more natural, fluid, and similar in quality to those written by humans.

The most advanced summarization systems are built on *Transformer*-based architectures (which excel in understanding context and relationships within long sequences of text) such as:

- *GPT-4.1* (*OpenAI*): An internal advancement over *GPT-4*, *GPT-4.1* delivers improved summarization quality, especially in instruction-following, long-context retention (up to 128k tokens), and robustness across diverse domains. Frequently used via *OpenAI*'s API and integrated into advanced copilots [53];
- *GPT-4.5* / *GPT-4o* (*OpenAI*): These are the latest iterations of the *GPT* family. *GPT-4o* (*Omni*) introduces multimodal capabilities (text, images, and audio) and significantly improved efficiency, while maintaining high summarization fluency, coherence, and contextual retention [54].
- *BART* (*Meta* AI): A hybrid model combining *BERT* (for encoding) and *GPT* (for decoding), capable of both extractive and abstractive summarization. It delivers fluent outputs with strong contextual coherence [55];
- *T5* / *FLAN-T5* (*Google*): T5 (Text-To-Text Transfer Transformer) frames all NLP tasks (including summarization) as a text-to-text problem, offering flexibility and high-quality outputs even for domain-specific texts [56]. *FLAN-T5* is a fine-tuned variant that excels in instruction-based tasks, including summarization;
- *PEGASUS* / *PEGASUS-X* (*Google*): Optimized specifically for summarization, *PEGASUS* is trained using a gap-sentence generation objective, making it particularly effective for long documents [57]. *PEGASUS-X* extends these capabilities to longer input sequences with improved scalability;

- *DistilBART*: A lighter version of *BART*, designed for faster execution and lower computational load, while retaining a significant portion of *BART*'s summarization quality [58];
- *Longformer* / *LongformerEncoderDecoder* (*Allen AI*): Designed for handling very long documents, *Longformer* uses sparse attention mechanisms to reduce computational complexity and memory usage [59].

These models can be integrated through APIs or hosted frameworks such as *Hugging Face*, depending on the desired balance between cost, performance, and scalability. Overall, content summarization through AI enables the platform to rapidly generate accurate and engaging learning material while reducing the manual effort required from Subject Matter Experts.

The following table summarizes the performance characteristics of key models used for summarization, based on criteria such as output quality, execution time, resource needs, and integration complexity:

Model	**Type**	**Cost**	**Quality output**	**Execution time**	**Integration complexity**	**Scala-bility**
GPT-4.1	Extractive/ Abstractive	Per token (input: $2/1M tokens, cached input: $0.5/1M tokens, output: $8/1M tokens) [60]	Excellent coherence and accuracy	Medium-slow (long texts)	Medium (via *OpenAI* API)	High (cloud-native)
GPT-4.5	Extractive/ Abstractive	Per token (input: $75/1M tokens, cached input: $37.5/1M tokens, output: $150/1M tokens) [60]	Very high, human-like summaries	Medium-slow (long texts)	Medium (via *OpenAI* API)	High (cloud-native)
GPT-4o	Extractive/ Abstractive	Per token (input: $2.5/1M tokens, cached input: $1.25/1M tokens, output: $10/1M tokens) [60]	Very high, human-like summaries	Medium-slow (long texts)	Medium (via *OpenAI* API)	High (cloud-native)
BART	Extractive/ Abstractive	Open source	Good contextual summaries	Medium	Medium (via Hugging Face)	Medium
T5	Abstractive	Open source	Flexible and	Medium-slow	Medium	Medium

			accurate summaries			
PEGASUS	Abstractive	Open source	High accuracy for long texts	Slow	Medium	Medium -High
DistilBART	Extractive Abstractive	Open source	Fast and lightweight	Fast	Low (easy *Hugging Face* setup)	Medium
Longformer	Extractive	Open source	Optimized for long input handling	Medium	High	Medium

Table 3.2: Comparison of AI models for content summarization

In the proposed microlearning platform, summarization models are integrated as part of a larger content pipeline. The selected models must provide a balance between high-quality output and operational efficiency to support real-time, scalable learning content generation. For this reason, *GPT-4.1* was chosen as the primary model for content summarization. *GPT-4.1* combines advanced natural language understanding with strong abstractive capabilities, producing highly fluent, coherent, and contextually accurate summaries across a wide range of business and technical domains. Compared to open-source alternatives, *GPT-4.1* offers superior handling of long and complex documents, requires minimal fine-tuning, and integrates seamlessly via API. Despite its higher token-based cost, its reliability, multilingual support, and reduced need for pre- and post-processing make it ideal for production environments where consistency and quality are essential.

3.3.2 Document parsing and preprocessing

One of the fundamental steps in the automated content generation pipeline is document parsing and preprocessing, because it allows unstructured or semi-structured data to be transformed into structured formats, thus making them suitable for processing by AI models.

To handle the variety of existing document formats (such as PDF, *Word*, *Excel*, *PowerPoint*, audio files, scanned images, or real-time speech), there are numerous tools and libraries available, both open-source and commercial.

In the specific case of text documents, real-world applications use *Python* libraries for content extraction and preprocessing, depending on the source file type. For PDF files, for example, *PyPDF2* is commonly used for simple parsing tasks, such as continuous text extraction or basic metadata retrieval [61]. When dealing with more complex PDFs (such as those containing multi-column layouts, tables, or embedded images), tools like *pdfplumber* and *pdfminer.six* are more effective, as they can handle complex document structures more accurately. *pdfplumber* allows for granular extraction of elements like bounding boxes, table structures, and line coordinates, which are essential when aiming to preserve the spatial integrity of educational materials (e.g., diagrams or tabular data) [62]. *pdfminer*, on the other hand, provides access to typographic features such as font weight, style (bold, italics), and character spacing, which can be useful for detecting emphasized text (e.g., definitions, titles, key terms) [63]. *PyMuPDF* (*fitz*) offers a superior handling of layout structure compared to simpler libraries like *PyPDF2*

and includes support for text and image extraction, layouts, annotations, and fonts, proving very useful in complex document pipelines [64].

Instead, *python-docx* provides access to text contained in *Word* documents (.docx), as well as applied styles (headings, bold, italics), tables, embedded images, and other structural elements [65]. It is suitable for analyzing administrative, academic documents, or formatted reports.

In the case of *Excel* files, two libraries are mainly used: *openpyxl* and *pandas*. *openpyxl* allows reading, writing, and editing *Excel* (.xlsx) files [66]. It allows the extraction of values from cells, formatting, formulas, styles, tables, and charts. It is useful in contexts where data are already structured in tabular form. p*andas*, although it is a data analysis library, offers a high-level interface for reading *Excel* and CSV files [67]. It uses *openpyxl* (for .xlsx) or *xlrd* and *xlsxwriter* internally, simplifying the import and manipulation of numeric and textual datasets.

Meanwhile, *python-pptx* provides tools for reading and manipulating *PowerPoint* presentations (.pptx) [68]. It allows the extraction of slide text, titles, text box content, images, and speaker's notes. It is very useful for generating or analyzing multimedia and educational content.

Python's native functions (*open, read, readlines*) can be used to easily read .txt files [69]. This method is suitable for linear content without complex formatting, such as logs, scripts, and simple text documents.

File format	Library/tool	Key features
PDF	*pdfplumber, pdfminer.six, pyPDF2, PyMuPDF*	Extraction of text, tables, layout info, and metadata
Word (.docx)	*python-docx*	Access to paragraphs, headings, and table content
Excel (.xlsx)	*pandas, openpyxl*	Tabular data extraction and manipulation
PowerPoint (.pptx)	python-pptx	Slide content extraction (titles, bullet points)
Plain text (.txt)	Native Python I/O	Direct text file reading

Table 3.3: Common *Python* libraries for document parsing by file type

In cases where documents are scanned or presented as images, Optical Character Recognition (OCR) becomes essential. *Tesseract*, an open-source solution, is used for printed text and offers multilingual support [70], though it is less accurate with cursive handwriting. *Google Cloud Vision API* and *Microsoft Azure Computer Vision* provide more robust support for handwritten text, layout detection, and multilingual content [71], [72]. *Amazon Textract* is specialized in reading structured documents, such as forms or invoices, and can extract tables and key-value pairs [73]. The following table provides an overview of key OCR technologies used in production environments for digitizing printed or handwritten text.

Tool/service	Type	Key features
Tesseract OCR	On-device	Open-source, suitable for printed text
Google Cloud Vision, Azure OCR, Amazon Textract	Cloud-based	High accuracy, support for complex layouts and handwriting recognition

Table 3.4: OCR tools and services

When audio input needs to be transcribed into a textual format (e.g., for processing webinars, meetings, or podcasts), Speech-to-Text services are often integrated. These services differ by supported languages, real-time capability, and specialization. The table below outlines some of the most relevant options:

Service	Key features
Google Cloud Speech-to-Text [74]	High accuracy, keyword and punctuation support
Microsoft Azure STT [75]	Customizable model
Amazon Transcribe [76]	Optimized for call transcription
IBM Watson STT [77]	Domain-adaptive transcription

Table 3.5: Speech-to-Text services

For privacy-conscious scenarios or offline use, open-source alternatives such as *Vosk* and *Mozilla DeepSpeech* offer on-device speech recognition. *Vosk* is lightweight, compatible with platforms like *Raspberry Pi*, and supports multiple languages, making it suitable for real-time transcription in low-resource environments [78]. *DeepSpeech*, although more resource-intensive, is based on neural networks and supports fine-tuning, making it flexible for custom applications [79].

After text extraction, the preprocessing pipeline often includes language-specific NLP analysis, such as tokenization, sentence segmentation, named entity recognition, and keyword extraction. For this process, the platform may use *Google Cloud Natural Language API* or *Azure Text Analytics*, both of which provide real-time entity extraction, sentiment analysis, and classification capabilities in multiple languages [80], [81]. These tools help enrich the extracted data before it is passed to summarization models, allowing for more context-aware learning content.

In summary, the document parsing and preprocessing phase is based on a broad set of libraries and APIs, designed to handle a wide range of heterogeneous input formats. It not only extracts the raw text but also preserves the structure, context, and semantic information of the content. This process builds a solid, high-quality foundation, essential for transforming corporate knowledge into interactive microlearning experiences powered by artificial intelligence.

The technologies adopted, considered among the most advanced in the field of document parsing, are widely used in business contexts such as automatic transcription, compliance management, customer support, and the digitization of data from legacy archives. However, the current version of the proposed platform is specifically focused on structured, text-based file formats and does not include support for audio or image-based content at this stage. It employed *PyMuPDF* for accurate and layout-aware text extraction, especially effective with complex PDFs such as reports and corporate documents. *python-docx* was chosen to process structured *Word* documents, where preserving styles and headings was important. For *Excel* files, *pandas* was preferred due to its concise syntax and efficient handling of tabular data. Simple .txt files were processed using *Python*'s native I/O functions, given their linear structure.

3.3.3 Online resource retrieval

A key step in automatic content generation is efficient online research. Among the most innovative tools in this field is *Tavily*, a search engine designed for research purposes, which uses Artificial Intelligence to provide accurate, high-quality, and contextually relevant results

[82]. Unlike traditional search engines, *Tavily* is able to filter out unhelpful or noisy content, making it particularly suitable for academic or technical contexts. Among the other AI-based alternatives are *Perplexity*, which provides conversational answers with cited sources [83], or *Exa.ai*, which is geared towards intelligent semantic search.

In the proposed project, *Tavily* was chosen for its ability to quickly retrieve high-quality information, a crucial aspect for ensuring the accuracy and relevance of the generated microlearning content. Moreover, the availability of a free plan for up to 1,000 requests made this tool perfectly compatible with the project's operational needs.

Tool	Key features	Pricing
Tavily	Filters noise, delivers high-quality, contextually relevant results	Free under 1,000 requests + $30/month (4,000 req/month) [84]
Perplexity	Provides answers with cited sources, conversational interface	$6/month (1,000 req/month) [85]
Exa.ai	Semantic search, real-time crawling	$5/month (1,000 req/month) [86]

Table 3.6: Search engine services

3.3.4 Slide generation techniques and quiz generation

In the proposed microlearning platform, automatic slide and quiz generation is based on the use of advanced linguistic models. To create slides, models such as *GPT* and *BART* are used to identify central concepts, suggest effective titles, and structure content into bullet points. This process is guided by heuristic rules or thematic segmentation techniques, while the visual aspect is managed through the use of predefined templates or multimodal models capable of integrating textual and visual elements to optimize layout and design.

Quiz generation, on the other hand, is based on Question Generation (QG) models, often specialized through fine-tuning on educational datasets. These models allow for the production of a variety of questions, including multiple-choice, true/false, single-answer, or short-answer questions. Specific techniques are employed, such as constructing cloze-style questions, using named entity recognition to generate distractors, and applying principles derived from *Bloom*'s Taxonomy to ensure an adequate level of cognitive depth. Integration of these techniques within e-learning platforms facilitates scalable personalized content creation and assessment.

In the proposed microlearning application, *OpenAI*'s *GPT4.1* was chosen to generate both slides and quizzes due to its versatility, fluency, and strong contextual understanding.

3.3.5 Image generation

Image generation is a rapidly advancing field within AI-based content creation, playing a crucial role in enhancing the visual dimension of automatically generated materials. Recent techniques leverage deep generative models such as Generative Adversarial Networks (GANs) [87], Variational Autoencoders (VAEs) [88], and, most notably, diffusion models [89]. These approaches can synthesize high-quality images from textual prompts or structured data. Among the latest breakthroughs, diffusion-based systems like *OpenAI*'s *DALL·E 2* [90] and *DALL·E 3* [91] have demonstrated remarkable capabilities in generating detailed, semantically accurate images directly from natural language input. *DALL·E 3*, in particular, offers significant improvements in prompt understanding and image fidelity compared to earlier versions, making it especially suitable for academic and educational content generation [91]. In this thesis project,

DALL·E 3 was adopted for image generation tasks due to its ability to consistently produce visually coherent and contextually relevant images aligned with the instructional goals of the generated materials. Its integration ensured that the visual elements complemented the text content both aesthetically and semantically, contributing to a more engaging and informative learning experience.

3.3.6 Multilingual support

Multilingual support represents a fundamental component of modern AI-based content generation systems, especially in educational and corporate learning contexts. As organizations increasingly operate globally, it is essential that learning materials are accessible to users with different languages and cultures. This means not only making content understandable to speakers of other languages, regardless of their technical skill level, but also adapting it to be culturally appropriate and familiar to the target audience. Content must be adapted to idiomatic expressions, formatting conventions, and regional language nuances (ensuring that the material feels natural and relevant to the target audience rather than appearing as a word-for-word translation).

To address these challenges. The most advanced solutions leverage Neural Machine Translation (NMT) systems, such as *MarianMT* [92], *mBART* [93], and multilingual versions of language models developed by *OpenAI* [94]. These models are designed to preserve the meaning, style, and narrative coherence of the original text, offering more natural and contextually appropriate translations.

In the proposed project, *OpenAI*'s *GPT-4.1* was adopted as the primary model for both content summarization and multilingual support. Its ability to interpret context and generate coherent texts makes it particularly suitable for translating training materials, where maintaining semantic clarity and consistency across language versions is crucial. By integrating *GPT-4.1* into the AI pipeline, the system was able to automatically create microlearning modules in multiple languages, eliminating the need for human intervention while simultaneously ensuring high quality and consistency in content delivery internationally.

3.4 Technology stack selection

To support the objectives of the AI-enhanced microlearning platform, a carefully designed technology stack was chosen, combining frontend agility, backend robustness, and cutting-edge AI capabilities. The overall architecture is designed to be modular, scalable, and easily maintainable.

The internal data flow follows an event-driven linear model:

1. The process begins when the user uploads a document or asks the chatbot to retrieve some sources about a specific topic;
2. The backend handles file parsing and extracts the content;
3. The processed data is then forwarded to the AI layer, where it is summarized, converted into slides, and enriched with automatically generated quizzes;
4. The final content is returned to the frontend, where the user can view, edit, and customize it.

A chatbot-based virtual assistant guides the interaction in real time, making this process seamless and accessible. Through structured prompts or conversational input, it guides the user

in the entire process, making managing the generated content simple and intuitive, even for those without specific technical skills.

3.4.1 Backend architecture and technologies

The backend architecture is built to support a wide set of advanced capabilities, ranging from AI-driven research and content analysis to secure, scalable file processing. By integrating *FastAPI* and *CopilotKit* with state-of-the-art *LLM*s and custom agents, the system provides a versatile and high-performance backend suitable for intelligent real-time applications.

The modular and layered design ensures extensibility, while security, monitoring, and validation layers maintain system integrity under load. Built in *Python 3.12* to leverage its mature ecosystem for document processing and AI integration, this architecture offers a powerful, flexible, and secure foundation for modern AI applications, whether processing documents, answering queries, or managing agent workflows.

FastAPI **framework**

The backend is built around *FastAPI*, a modern, high-performance web framework optimized for creating APIs with *Python 3.7* and later versions. Its native support for asynchronous programming makes it especially well-suited for handling multiple concurrent I/O-bound tasks (such as managing API requests) efficiently and reliably. *FastAPI* also takes advantage of *Python*'s type hints to perform automatic request validation and data conversion, ensuring consistent and robust input handling throughout the application. This is complemented by seamless integration with *Pydantic*, which provides powerful models for data serialization, parsing, and validation, resulting in robust and clearly defined data structures. These features not only boost developer productivity by reducing boilerplate code but also enhance code quality and maintainability. As the primary entry point for client requests, *FastAPI* orchestrates *interactions* among the various backend modules.

CopilotKit **integration and configuration**

The project builds upon *CopilotKit*, an open-source framework available on *GitHub*, which served as a reliable starting point for integrating AI agents into the web application. *CopilotKit* offers high-level abstractions for managing conversational agents and connecting them with AI models. Among its features is the *LangGraphAgent*, which enables the design of modular, stateful agents capable of multi-step reasoning. The framework supports various Large Language Models, including *OpenAI*'s *GPT-4* and *Google GenAI*; in this microlearning application, *OpenAI*'s *GPT-4.1* is specifically used. Additionally, *CopilotKit* provides custom action handlers (flexible tools that agents can invoke to perform operations such as file analysis, resource management, search, and summarization), greatly enhancing the system's interactivity and adaptability.

The decision to use the *CopilotKit* project rather than starting from scratch was driven by both technical and strategic considerations. It provides a good foundation for managing conversational agents and orchestrating AI workflows. It allows to reduce development time and focus on implementing application-specific microlearning features. By adopting an existing, robust infrastructure, the project benefited from proven patterns and tools, avoiding the need to reinvent core functionalities.

Core endpoints

The backend exposes a well-structured, RESTful API that acts as the main interface for client interactions. At its core is the */copilotkit* endpoint, which processes requests directed to the AI agent and returns structured responses. To support system monitoring and reliability, a */health* endpoint is provided, offering real-time information about backend availability and operational status. Furthermore, the *CopilotKit* framework includes custom action endpoints, enabling the triggering of specific functions (such as file uploads, data analysis, or other auxiliary processes) directly through the API. The custom actions are registered with the *CopilotRuntime* and can be called by the AI when needed, allowing it to perform specific tasks and interact with other services.

Request Handling

The system implements asynchronous processing through *FastAPI* and *aiohttp*, enabling non-blocking operations for resource-intensive tasks. Concurrent execution is achieved using *Python*'s *asyncio*, particularly for parallel downloads and searches. The application uses event streams for real-time updates and implements caching mechanisms for downloaded resources.

Authentication

The backend secures API access using a simple token-based authentication system. A valid token is retrieved from environment variables, with a fallback to a default hardcoded token if none is specified. Authentication is enforced through middleware that intercepts all incoming HTTP requests, except those targeting the root endpoint (/), which remains publicly accessible to serve the application.

For all other requests, the middleware requires the token to be provided as a query parameter named '*token*'. It verifies that the token is present and matches the configured valid token. If the token is missing or incorrect, the middleware immediately returns a '*403 Forbidden error*' response, effectively blocking unauthorized access.

After successful validation, the middleware rewrites the request path by removing the token from the query parameters while preserving any other parameters. This ensures that downstream request handlers receive a clean URL without the authentication token exposed.

On the frontend, the token is appended as a query parameter to all API calls directed to protected endpoints. This simple yet effective approach provides a basic security layer for the API, ensuring that only requests with a valid token are processed, while keeping the root endpoint open for public access.

State management

State management plays a fundamental role in ensuring consistency, traceability, and continuity in the execution of intelligent agents. To support these needs, the system incorporates a dedicated structure for representing the agent's internal state, which serves as a structured memory. This mechanism enables the agent to:

- Retain relevant information throughout its execution lifecycle;
- Ensure data consistency across operations and interactions;

- Facilitate monitoring and debugging by maintaining a clear record of actions and decisions.

The state model is designed to be flexible, operating in either ephemeral or persistent modes depending on the nature of the task. It also supports synchronized updates across asynchronous workflows, ensuring coherence even in complex execution scenarios.

Specific implementation details (such as the structure of the state, the types of data it manages, and the mechanisms for updating and persisting information) are discussed in the dedicated implementation chapter.

LangChain and *LangGraph* integration

The backend integrates *LangChain* and *LangGraph* to deliver advanced AI-based capabilities. *LangChain* acts as a powerful abstraction layer for developing applications that leverage Large Language Models, offering essential tools for connecting with diverse model providers and toolchains. It also simplifies prompt management and enables the seamless chaining of context-aware interactions.

Building upon this foundation, *LangGraph* introduces a stateful, graph-based architecture designed to manage complex multi-agent workflows. In this architecture, each node represents a distinct task or tool, allowing for modular and composable agent behavior. *LangGraph* further supports agent memory and dynamic decision-making, enabling workflows to adapt flexibly based on intermediate outcomes.

Together, *LangChain* and *LangGraph* form a cohesive framework capable of modeling and executing sophisticated AI applications in areas such as conversational agents, information retrieval, and content generation.

Agent architecture

The agent architecture is designed to be modular and extensible, which promotes maintainability and clarity throughout the system. At the heart of the system lies the research agent, an AI component tailored for tasks like document analysis, summarization, and multi-tool reasoning. Custom actions are designed modularly, each featuring thorough input validation, error handling, and execution logic to ensure reliable operation. To maintain consistency, the agent's responses are formatted in a standardized way, facilitating their use in frontend interfaces or chaining with other workflows. Furthermore, all tools and actions within the agent adhere to strict parameter validation schemas and incorporate fallback mechanisms to handle unexpected inputs.

Graph architecture

The system's workflows are orchestrated using *LangGraph*'s *StateGraph*, a declarative framework in which nodes represent discrete functional units. Each node encapsulates a specific operation within the broader task execution pipeline. For instance, the *Download Node* is responsible for retrieving and preprocessing external resources, while the *Chat Node* interprets user inputs and coordinates Large Language Model responses. The *Search Node* interfaces with *Tavily* to conduct real-time web searches, injecting dynamic context into the agent's reasoning.

Additionally, *Add* and *Delete Nodes* enable agents to modify their internal resource list, facilitating the dynamic management of their working context.

Transitions between these nodes are governed by conditional logic and updates to the shared state, allowing workflows to adapt based on intermediate outcomes. This architecture supports modularity, extensibility, and precise control over agent behavior, making it well-suited for building complex and multi-stage AI applications.

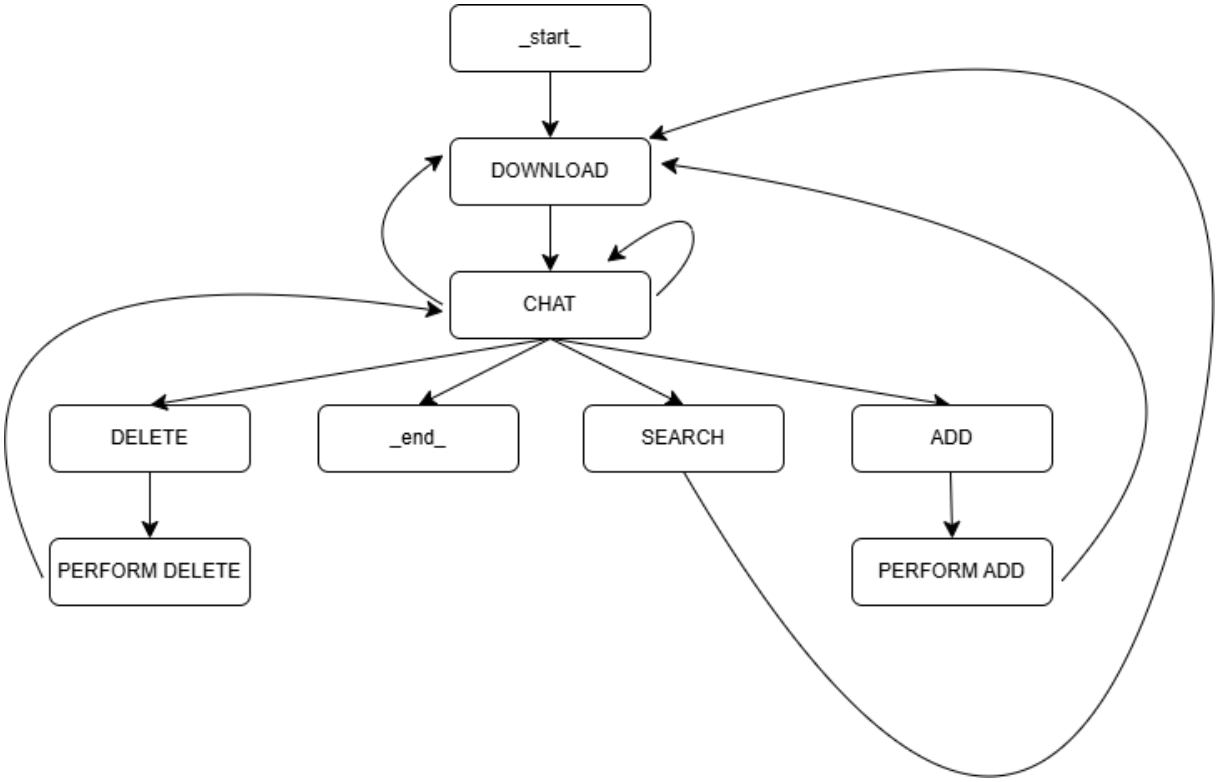


Figure 3.1: Graph architecture including the nodes and their interactions

File conversion system

The system provides robust support for handling various formats by using b*ase64* encoding and decoding to allow file transmission over the API. Supported file types include PDFs, *Word* documents, *Excel* spreadsheets, and plain text files. Processing occurs either in-memory or through secure temporary storage, with regular cleanup tasks to ensure efficient resource management. To enhance robustness, fallback mechanisms are implemented to recover from corrupted or unsupported files. The following *Python* libraries have been selected and integrated into the backend system:

File format	Library used	Reason for selection
PDF	*pyMuPDF* (*fitz*)	Offers fast and reliable text extraction
Word (.docx)	*python-docx*	Handles structured *Word* documents effectively
Excel (.xlsx)	*pandas*	Robust handling of tabular data and spreadsheet structures
Plain text (.txt)	Native *Python*	Direct reading

Table 3.7: Tools adopted in the thesis project for file parsing

These choices reflect a balance between flexibility and practicality. They allow the system to process a wide variety of document types efficiently, while keeping the architecture simple and maintainable. Additionally, the selected libraries benefit from active community support and thorough documentation, ensuring long-term viability and ease of future enhancement.

This multi-format document processing pipeline ensures that enterprise content can be ingested regardless of its original source structure, enabling seamless transition into AI-driven microlearning workflows. Future enhancements may include support for OCR and audio-based input, broadening the platform's functionality and enabling it to process informal or legacy content more effectively.

File download and processing

The system includes robust capabilities for retrieving and preparing external content to support downstream tasks handled by the language model. When a resource is specified via URL, asynchronous downloading is performed using *aiohttp*, enabling efficient non-blocking retrieval (a key requirement for maintaining responsiveness in concurrent workflows).

After download, raw HTML is converted to Markdown using the *html2text* library, improving readability and ensuring prompt compatibility with the language model by eliminating unnecessary visual styling. To further clean and structure the input, the system leverages *BeautifulSoup* for HTML parsing and content extraction, removing scripts, ads, and irrelevant formatting to produce structured text optimized for analysis.

To improve performance and reduce redundant network calls, a caching layer is employed. This mechanism stores previously fetched content, minimizing latency and resource usage in repeated or batch operations.

Performance optimizations

Given the computational intensity and I/O demands of the application, optimizing performance is essential to ensure responsiveness and scalability. For this purpose, the system employs two key strategies.

Primarily, asynchronous workflows are utilized extensively, enabling non-blocking execution of API calls, file I/O, and resource downloads. This approach significantly enhances throughput by allowing multiple operations to proceed concurrently without waiting for individual tasks to complete.

Caching mechanisms play a crucial role in reducing load and improving response times. Downloaded files and resources are cached locally or in memory to avoid redundant retrievals. *LangGraph* agents benefit from graph state caching, reusing intermediate computational states to accelerate processing.

Error management

Error management is centralized, providing standardized behavior across all API endpoints. Errors are returned in structured *JSON* responses with clear status codes and descriptive messages. Domain-specific custom exceptions improve clarity and ease debugging, while

comprehensive logging captures full stack traces and relevant metadata. Additionally, fallback strategies are implemented to allow recovery from non-critical service failures.

System monitoring

To ensure high availability and system reliability, the system incorporates comprehensive health check mechanisms implemented via dedicated HTTP *GET* endpoints that serve as liveness and readiness probes, allowing orchestration tools and deployment platforms to assess service status in real time. The liveness probe verifies that the application is still running and not stuck in an unrecoverable state, enabling automated restarts when necessary to restore functionality. The readiness probe, on the other hand, checks whether the application is fully initialized and ready to serve traffic, preventing premature routing of requests during startup.

By decoupling these checks, the system provides fine-grained control over lifecycle management, reducing downtime and ensuring that traffic is only directed to healthy instances.

Debugging and profiling

Comprehensive observability tools support efficient debugging and performance optimization. Each request and response is logged in detail, capturing information such as HTTP method, request path, user token, status code, and execution duration. A centralized logging infrastructure ensures that errors are recorded in a structured and searchable format, greatly simplifying the debugging process. Additionally, developers can inspect the internal state and transitions of *LangGraph* agents in real time using *LangSmith*, providing valuable insight into the system's reasoning processes and helping in the development and tuning of AI workflows.

3.4.2 AI models

At the core of the system's intelligence layer are integrations with *OpenAI*'s *GPT* models, which drive natural language understanding and generation, along with *DALL·E 3*, which enables image-based content creation. To further enrich the AI's contextual awareness, the system also leverages *Tavily* for advanced web search capabilities. These models are accessed through the *CopilotKit* framework and play a central role in transforming static learning materials into dynamic conversational experiences.

***OpenAI* integration**

The backend is tightly integrated with *OpenAI*'s suite of models to enable advanced language and image capabilities. At the core of this integration is *GPT-4.1*, which is employed for a wide range of natural language understanding and generation tasks. *GPT-4.1* supports key features such as conversational interfaces and document analysis, forming a foundational component of the system's intelligence layer.

Beyond text processing, the system integrates *DALL·E 3* to generate high-quality images from textual prompts, enriching the content creation process through seamless multimodal interactions.

The backend also leverages *GPT-4.1*'s advanced tool-calling capabilities, including parallel function execution. This allows the model to invoke multiple tools, such as web search or image generation, within a single conversational turn, enhancing both contextual depth and responsiveness.

To provide greater control over model behavior, configurable parameters, such as temperature and token limits, are exposed through the system's settings, enabling fine-tuned customization of model outputs based on application requirements.

Tavily search integration

Tavily is integrated into the system as a high-performance web search API, enhancing the Large Language Model's ability to access and incorporate real-time information. This enables dynamic context injection during conversations and task execution, ensuring that responses remain up-to-date and relevant.

Unlike traditional keyword-based search engines, *Tavily* uses semantic relevance to return results that closely match the user's intent, allowing the system to extract high-quality content snippets tailored to the context.

Within the *LangChain* and *LangGraph* architecture, *Tavily* is implemented as a tool node, making it selectively accessible to agents as needed during workflow execution. This modular integration ensures that agents can retrieve external information when necessary, improving both accuracy and adaptability in complex tasks.

3.4.3 Frontend architecture and technologies

The user interface is primarily developed with *React*, enhanced by *CopilotKit* components that enable AI-powered features such as contextual chat and intelligent suggestions. A highlight of the frontend implementation is the integration of a *Svelte* web component sourced from another internal company application.

Overall, the frontend architecture of the application is designed to offer a modern, scalable, and maintainable solution, balancing performance, usability, and seamless AI integration. It leverages cutting-edge frameworks and libraries to ensure both development efficiency and a rich user experience. This chapter details the core technologies, design systems, state management strategies, and optimization techniques used to construct the user-facing side of the application.

Core framework and technologies

The frontend is built on *Next.js 14*, a *React* framework that supports both Server-Side Rendering (SSR) and Static Site Generation (SSG), offering flexible content delivery and improved load performance. The application leverages *App Router*, a modern *Next.js* routing architecture, which introduces advanced features such as nested layouts, server components, and co-located routing logic.

The codebase is written in *TypeScript*, enabling static typing and type safety, which helps prevent runtime errors. For client-side interactivity, the "*use client*" directive is employed to clearly define which components should be rendered on the client, allowing for fine-grained control over hydration and performance optimization.

Underlying the framework is *React 18*, which serves as the core UI library. The application uses *React Hooks* for managing state and effects, ensuring a clean and functional code structure. *React Context* is used for handling global or shared states, simplifying data flow and avoiding prop drilling.

UI components and design system

The application's interface is built using custom-built UI components based on *Radix UI* primitives, ensuring accessibility and composability at the core.

Core elements, such as *Dialogs, Cards, Buttons, Inputs,* and *Textareas,* have been tailored to meet the unique functional and aesthetic requirements of the platform, with a focus on usability, visual clarity, and themability.

Styling is managed with *Tailwind CSS*, a utility-first *CSS* framework that accelerates development by allowing components to be styled directly via composable class names. To maintain a cohesive visual identity, the application defines a custom color palette and a set of design tokens, supporting consistent theming across all components and layouts.

Slide visualization

In the project, a custom web component named <*challenges-app*> is used to encapsulate a self-contained reusable part of the user interface dedicated to slide visualization. This cross-project integration demonstrates the system's capability to reuse components across different projects within the same enterprise ecosystem. Leveraging an existing production-tested module helped accelerate development, ensure visual and functional consistency, and minimize maintenance overhead by centralizing component logic. This approach reflects a strategic emphasis on scalability, efficient resource utilization, and interoperability between internal tools and applications.

The web component is built in *Svelte* and integrated into the *React* page, but it operates independently from the *React* application itself. When the *React* page is loaded, an external *JavaScript* file is executed. This file defines and registers the <*challenges-app*> custom element, making it available in the *DOM*. After a brief initialization period (to ensure the component is registered and ready), the component is dynamically inserted into the page using its custom HTML tag. The *React* app communicates with the web component by setting properties directly on the *DOM* element. The slides object is passed to the component in this way, allowing it to render the appropriate content. Once initialized, the component takes full responsibility for rendering and managing the slide presentation. It handles its own internal state and user interactions, without relying on *React* for rendering or lifecycle management.

The primary function of the <*challenges-app*> component in this project is to present a sequence of slides to the user. By offloading this functionality to a dedicated web component, the slide viewer benefits from focused, encapsulated logic and a clean interface for integration.

State management

The application adopts a context-based state management approach, using *React*'s built-in *Context* API in combination with custom hooks to create a shared state container. This design enhances clarity and modularity by minimizing prop drilling and ensuring that all components can access and update the state they need.

At the core of this system is *Copilot Context*, which manages application-level data in a single unified store. Components interact with the *Context* through a custom hook that abstracts internal state logic, providing a clean separation between state logic and UI rendering.

The design organizes the state into distinct domains, each serving a specific function:

- Actions management: Handles user actions and updates related to frontend logic;
- Context and document handling: Manages contextual information relevant to the user or session;
- Chat and *Co-Agent state*: Supports conversational features, including instructions, suggestions, and dynamic agent behavior;
- API configuration: Centralizes endpoint and authentication settings for external communication.

This architecture enforces boundaries between concerns, making the application easier to understand, scale, and maintain.

The state management system is guided by several core principles that align with *React*'s component-driven architecture. First, type safety is ensured through the use of *TypeScript*, allowing the entire application state to be strongly typed (this reduces runtime errors and improves developer tooling support). Second, encapsulation is achieved by restricting state access to well-defined interfaces, minimizing coupling between components, and promoting modularity. Lastly, the provider pattern is used to expose the context through a <*CopilotKit*> component, ensuring that any part of the application accessing shared state is properly scoped within the component tree.

This context-based approach offers strong benefits in terms of scalability, as new state slices or features can be integrated with minimal disruption to existing code. It also enhances maintainability by enforcing a clear separation of logic and structure, while strong typing simplifies debugging and code navigation. Finally, the use of custom hooks and clearly defined interfaces improves the overall developer experience by streamlining interaction with shared state and encouraging code reuse.

AI integration

A defining feature of the frontend is its deep integration with AI capabilities, achieved through the comprehensive use of the *CopilotKit* library suite. The system uses *@copilotkit/react-core* to manage AI interaction logic, *@copilotkit/react-ui* to provide interface components that expose AI functionalities to the user, and *@copilotkit/runtime* to handle communication with backend services and AI models.

At the center of the user experience is a highly interactive chat interface that allows users to engage in real-time conversations with the AI agent. This interface supports sending and receiving messages, presenting contextual suggestions, and dynamically generating follow-up prompts (creating a fluid, responsive dialogue that adapts to user input).

Advanced features such as file processing, image generation, and context-aware suggestions are embedded directly into the UI, enhancing the user's ability to interact with and benefit from AI assistance in practical, task-oriented workflows.

File handling and resources

To support diverse user workflows, the frontend incorporates comprehensive file management capabilities. Users can upload and edit a variety of file types (including PDF, DOCX, XLSX,

and TXT formats) directly within the application. Files are encoded and decoded using b*ase64*, allowing transmission in the browser environment.

The user interface consists of intuitive components (such as resource cards, edit dialogs, and upload panels) that make file interaction simple and accessible. Progress indicators provide real-time feedback during lengthy operations, while rigorous file type validation ensures only supported formats are processed. The system is designed so that users can manage and edit their resources without interrupting the main application workflow.

Performance optimizations

Optimizing performance is a core concern throughout the frontend codebase. *React*'s built-in features, like *useCallback*, are widely adopted to memorize functions and computed values, reducing redundant calculations and re-renders.

UI performance is further enhanced by leveraging *React 18*'s concurrent rendering, which improves responsiveness during intensive operations. Responsive design principles and feedback mechanisms, including loading states and progress indicators, contribute to a smooth and intuitive user experience.

Development tools and configuration

To ensure high code quality and support efficient development, the project adopts a robust set of configuration and tooling practices. The entire codebase is written in *TypeScript*, providing strong type safety and enhancing developer productivity through improved editor support and static analysis. *ESLint* is integrated to enforce a consistent code style and proactively catch potential issues during development.

Styling is managed using a customized *Tailwind CSS* configuration, which includes project-specific themes and plugins to maintain visual consistency and flexibility. Additionally, environment variables are securely handled across multiple deployment stages, enabling dynamic configuration of critical features such as API keys.

The build and deployment pipeline takes full advantage of *Next.js* build optimizations, balancing SSR and SSG for performance. API routes are defined within the *Next.js* project structure, allowing seamless frontend-backend communication.

User experience features

User experience is a core focus of the frontend design. Interactivity is elevated through a real-time chat interface, intuitive file management tools, and visual feedback mechanisms such as error messages and loading spinners. The latter not only informs users about the current system status but also serves a psychological purpose: by providing clear feedback during asynchronous operations, they reduce the perceived waiting time and make interactions feel faster and smoother.

Accessibility features are seamlessly embedded within the component structure. Use of *ARIA* roles and labels, keyboard navigation, and screen reader compatibility ensures that the application is usable by a diverse range of users, including those relying on assistive technologies.

This architecture reflects a modern frontend implementation that harmoniously blends performance, accessibility, and AI functionality. By leveraging the capabilities of *Next.js, React 18, Tailwind CSS*, and *CopilotKit*, it achieves a high level of interactivity and responsiveness while maintaining clean code organization and developer ergonomics. The result is a frontend that is not only technologically advanced but also user-centric.

3.4.4 Cloud infrastructure and security layer

The solution is built using a modern serverless approach on *Amazon Web Services* (*AWS*), leveraging the *AWS Cloud Development Kit* (*CDK*) to provision and manage infrastructure. The design prioritizes scalability, maintainability, and cost-effectiveness while following best practices in cloud-native development.

Architectural overview

The application follows a serverless-first architecture, built around two key computational components:

- UI function: A frontend application implemented in *Next.js*, deployed as a *Lambda* function.
- Agent function: A *Python*-based backend responsible for handling core logic and external API communication, deployed as a *Lambda* function.

These components are independently deployed as *AWS Lambda* functions using container images, enabling flexible dependency management and consistent runtime environments.

Region and identity management

The application stack is deployed in the *eu-west-1* (Ireland) *AWS region* and secured using *AWS Single Sign-On* (SSO) for administrator access.

The complete infrastructure is defined and provisioned using *AWS CDK*. The main infrastructure stack is named *CoAgentsDemoStack* and is version-controlled within the source code repository. This approach ensures repeatability, traceability, and facilitates continuous integration and delivery (CI/CD).

Key *AWS* services used

The architecture is built upon a tightly integrated set of *AWS* services, each playing a critical role in supporting the application's functionality, scalability, and security. At its core, the system relies on *AWS Lambda* as the primary compute layer for both the frontend and backend components. This serverless execution model eliminates the need for managing infrastructure and enables automatic scaling based on demand, ensuring responsiveness and cost-efficiency.

Both *Lambda function*s are deployed using container images hosted on *Amazon Elastic Container Registry* (*ECR*), which provides a secure, scalable, and highly available registry for *Docker* images. By packaging the application logic in containers, the development team gains full control over runtime dependencies, simplifies the build and deployment process, and ensures consistency across executions. This approach also enables the use of custom runtimes and native libraries, which are essential for the AI-powered backend services. In our case, the entire system is currently configured to operate within a single development environment,

eliminating the complexity of managing multiple deployment stages. However, the architecture is designed to be easily extendable to additional environments (such as test or production) if needed in the future.

For secure handling of sensitive credentials (such as third-party API keys), *AWS Secrets Manager* is used. Secrets are stored centrally and accessed securely at runtime, avoiding hardcoding sensitive data into the codebase. In this case, both the *OpenAI* and the *Tavily* API keys are stored using the *Secrets Manager*.

At the network edge, *AWS CloudFront* is used not only to accelerate content delivery globally but also to enhance security at the infrastructure level. A layer of *Basic Authentication* is configured at the *CloudFront* distribution, acting as a gatekeeper by requiring valid credentials before users can even reach the application entry point. This provides an additional barrier against unauthorized access and protects the system from casual probing and misuse.

Amazon CloudWatch is used to monitor the application's behavior and operational health. Dedicated log groups are created for each *Lambda function*, enabling real-time monitoring and log retention (limited to one week). This observability layer allows developers to track performance metrics, detect anomalies, and debug issues efficiently.

Infrastructure provisioning (that is, automatic creation and configuration of infrastructure resources) and lifecycle management are handled through *AWS CloudFormation*, using the *AWS Cloud Development Kit* (*CDK*). This *Infrastructure as Code* (*IaC*) approach ensures reproducibility and version control. With *CDK*, the infrastructure is defined in a programmatic and modular way, making it easy to update, audit, and replicate environments consistently across stages such as development, testing, and production.

The architecture intentionally omits any storage system or database because the application is designed to operate in a stateless real-time manner. Each interaction is handled independently, with no need to retain data across sessions or requests. This approach simplifies the system considerably and aligns well with the serverless model provided by *AWS Lambda*, allowing it to scale automatically and remain highly cost-effective without the additional complexity of managing persistent state.

Moreover, by not storing user data or logs long-term, the application inherently strengthens its security and privacy approach. It reduces the risk associated with data breaches and makes it easier to comply with data protection regulations, since there's simply less data to secure or audit. When data is needed during execution (such as for generating responses or processing input), it can be fetched on demand from external APIs, removing the necessity for a local database.

This design also helps minimize operational overhead. There's no need to manage database configurations, backups, or schema migrations, which makes the system easier to maintain and deploy. The focus remains on delivering responsive, AI-assisted interactions in real time, without the burden of persisting or managing state. If future requirements change, a storage layer can be added modularly, but for now, the lean and stateless design serves the application's goals perfectly.

The following diagram provides a visual overview of the system architecture described above. It illustrates how the various *AWS* services are interconnected to support the application's core

functionality, security, scalability, and observability in a serverless and containerized environment.

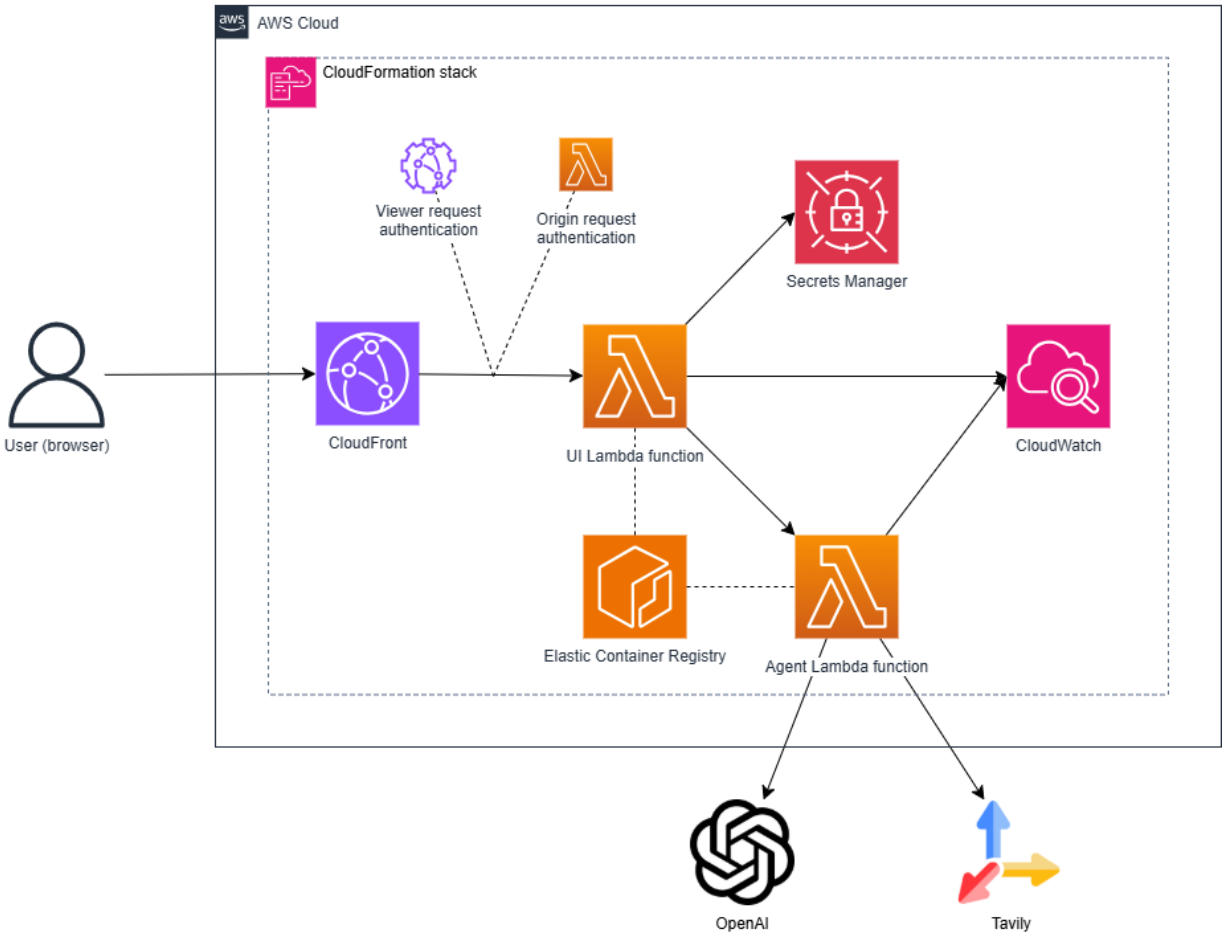


Figure 3.2: High-level architecture of the AI-assisted serverless application deployed on *AWS*

Although not shown in the accompanying figure, the application leverages *AWS Identity and Access Management* (*IAM*) to enforce secure and granular control over resource access. *IAM* policies and roles are carefully defined to ensure that each component has only the permissions it requires, adhering to the principle of least privilege and reducing the risk of misconfiguration. As it is a foundational component of any *AWS* architecture, its presence is considered implicit in the overall architecture.

Agent Function

The *Agent Function,* implemented in *Python 3.12*, operates as the backend logic layer and is deployed as a containerized *AWS Lambda Function* with the following characteristics:

Attribute	Details
Memory	1024 MB
Timeout	600 s
Runtime	*Python* 3.12
Environment variables	*OPENAI_API_KEY* *TAVILY_API_KEY* *LANGCHAIN_API_KEY*

	LANGCHAIN_ENDPOINT *LANGCHAIN_PROJECT* *LANGCHAIN_TRACING_V2* *AWS_LWA_INVOKE_MODE* PORT: 8000

Table 3.8: Agent Function characteristics

The function exposes a *Lambda Function URL* with *CORS* enabled and without *IAM* authentication. While the *Lambda URL* itself does not require *IAM* authentication, the backend is protected by a token-based middleware that validates each request. This architecture allows direct communication between the UI component and the backend while maintaining security through token validation, and still minimizing latency in inter-service communication.

Advantages of using *AWS Lambda* over *Amazon EC2*

The choice to adopt *AWS Lambda* instead of traditional *Amazon EC2* instances was guided by a clear set of architectural, operational, and strategic benefits that align closely with the goals of this project. *Lambda*'s serverless model offers a level of agility, scalability, and efficiency that is particularly well-suited for modern cloud-native applications, including AI-driven workflows and interactive learning platforms.

One of *AWS Lambda*'s most interesting advantages is its ability to scale automatically and instantly with incoming traffic. Each invocation runs independently, enabling the system to scale from zero to thousands of concurrent executions without manual intervention. In contrast, *EC2* requires predefined scaling policies and infrastructure setup, which introduces latency and operational complexity during traffic spikes.

Lambda's pay-per-use billing model charges only for the exact execution time and number of invocations, making it highly cost-effective for workloads with variable or unpredictable usage (such as demo environments, sporadic user activity, or bursty AI queries). *EC2* may be more economical for consistently high-throughput workloads, but it typically incurs continuous costs regardless of actual usage.

By abstracting away server management, *Lambda* eliminates the need to provision, patch, monitor infrastructure, or scale virtual machines, allowing developers to focus purely on application logic. In contrast, *EC2* requires full lifecycle management, including OS maintenance and performance monitoring, increasing the operational overhead.

Lambda functions run in isolated execution environments managed by *AWS*, with automatic handling of security updates and operating system patching. This built-in security model reduces the attack surface and simplifies compliance. *EC2*, in comparison, places more responsibility on the user to secure the operating system, configure firewalls, and manage updates.

Using *Lambda* with containerized deployment significantly shortens release cycles. Functions can be deployed quickly with minimal setup, while infrastructure changes are easily manageable through version-controlled configurations using tools like *AWS CDK*. This approach supports faster iterations and agile delivery.

Lambda integrates seamlessly with a wide range of *AWS* services, including *CloudWatch* for monitoring and logging, and *Secrets Manager* for secure configuration management. These integrations reduce boilerplate code and enhance the reliability and observability of the application.

In summary, *AWS Lambda* offers a robust, scalable, and cost-effective foundation for the system's architecture. Its serverless nature supports rapid development cycles, simplifies infrastructure management, and enhances security strategy (all essential elements for building and maintaining an efficient AI-assisted learning application).

Requirement	***Lambda* + containers**	***EC2* instances**
Compute granularity	Per request	Per VM instance
Scaling	Instant and automatic	Manual or slower Auto Scaling
Cost model	Millisecond billing	Hourly or longer-term
Operational overhead	Minimal	High (OS, patching, infrastructure)
Custom environment control	Container-level	OS-level full control
Task suitability	Event-driven, short	Long-running, stateful tasks

Table 3.9: Comparison between *AWS Lambda* and *EC2*

UI Function

The frontend is built with *Next.js* and deployed as a *Lambda Function* using a *Node.js 20* container image. It is configured with *Lambda* streaming features to support real-time interactivity.

Attribute	**Details**
Memory	4096 MB
Timeout	600 s
Runtime	*Node.js* 20
Environment variables	*REMOTE_ACTION_URL (*points to the *Agent Function URL)* *SECRET_NAME (Secrets Manager* reference*)* *AWS_LWA_INVOKE_MODE* PORT: 3000

Table 3.10: UI Function characteristics

Similar to the *Agent Function*, it exposes a *Function URL* with *CORS* enabled and without *IAM* authentication, optimized for seamless direct access from browsers.

Advantages of using *AWS Lambda* over *AWS Amplify*

While *AWS Amplify* is a popular choice for deploying frontend applications, we opted to use *AWS Lambda* for the following reasons:

- Custom runtime and streaming support: The frontend requires features like *Lambda* streaming for real-time interactivity, which are not natively supported by *Amplify*

hosting. Deploying it as a *Lambda function* allows to use a custom *Node.js 20* runtime and advanced features such as *HTTP* response streaming;

- Containerized deployment: The frontend is packaged as a *Docker* container, offering full control over the runtime environment. *Amplify* supports static site hosting but does not support container-based deployments.
- Unified serverless architecture: Using *Lambda* for both the frontend and backend results in a uniform serverless infrastructure. This simplifies configuration, deployment, and scaling across the stack;
- Greater flexibility: *Lambda* provides more control over request handling, environment configuration, and integration with services like *Secrets Manager* (capabilities that are limited or require workarounds in *Amplify*);
- Minimal dependencies: Unlike *Amplify*, which introduces an opinionated framework and *CLI*, *Lambda* allows for a leaner, more controlled deployment process using standard *AWS* tools.

In summary, *Lambda* offered the flexibility, advanced features, and runtime control required by this application (capabilities that go beyond what *Amplify* was designed to offer).

Containerization and *Lambda* adapter

To bridge the gap between traditional web server patterns and *AWS Lambda*'s event-driven model, both *Lambda functions* are packaged as container images that include the *AWS Lambda Web Adapter*. In our case, the application runs on *Node.js*, allowing us to use familiar frameworks and HTTP request handling patterns.

Using the adapter, each container can listen on a defined port (e.g., 3000), handling requests as if it were a conventional web server. This avoids the need to rearchitect existing application logic for *Lambda*'s native event structure. As a result, standard web server code, middleware, and routing logic can be reused without significant changes.

Ultimately, the *Lambda Web Adapter* allows containerized functions to behave like persistent web services while benefiting from *Lambda*'s elasticity, low overhead, and simplified operations. It enables a seamless serverless deployment path without sacrificing development familiarity or application consistency.

Conclusion

In conclusion, this architecture is intentionally designed to support growth, ease of maintenance, and efficient development workflows. By centering the system on a serverless model, leveraging containerized deployments through *AWS Lambda*, and integrating with managed *AWS services*, the application achieves a balance of flexibility and security. These design decisions reduce the need for manual infrastructure management, allowing the team to focus on evolving the product quickly and reliably as requirements change.

Chapter 4

4. PLATFORM IMPLEMENTATION AND FEATURES

This chapter presents the technical foundation and practical implementation of the AI-driven platform developed during this thesis project. It focuses on the core functionalities, architectural decisions, and software components that enable the system to operate effectively as a microlearning content generator and editor. Built with a strong emphasis on modularity, scalability, and user-focused design, the platform integrates modern development frameworks and cutting-edge AI technologies.

The discussion opens with a detailed look at the system's internal state management approach. It explains how the platform preserves context and logical consistency throughout user interactions by employing custom classes, *React Context Providers*, and custom hooks.

The chapter then shifts focus to the resource management system, highlighting how the application handles content uploads across multiple file types.

Subsequently, the chapter delves into the online resource retrieval mechanisms that integrate external APIs for automated content sourcing, followed by a detailed description of the automated generation of learning modules, which are structured as slide-based presentations enhanced by AI-generated quizzes and images.

Next, it describes the editing and customization interface, which enables users to personalize slides and questions through an intuitive AI-assisted editor. This is complemented by the integration of a context-aware chatbot, designed to assist users throughout the process with intelligent suggestions and conversational support.

The chapter concludes with a discussion on multilingual capabilities, ensuring accessibility across diverse user bases, and outlines the development tools and frameworks employed to build, test, and deploy the system. Each section demonstrates how the platform harmoniously combines frontend technologies, backend processing, and advanced AI to deliver a seamless and intelligent user experience.

4.1 State management

A custom-designed *AgentState* class serves as the foundation for structured and persistent memory within the system’s AI agents. This class plays a critical role in enabling the agents to maintain coherence throughout their execution lifecycle. It is responsible for tracking all

relevant information during execution, ensuring data consistency across operations, and enabling the saving and restoration of the agent's state when needed. Additionally, it facilitates debugging and monitoring, making it easier to trace and understand the agent's behavior in complex workflows. The *AgentState* class encapsulates several key components:

- A *Report* object that contains a list of *Slides*. Each slide can include a title, an image description, a textual description, the type of question (*RADIOBUTTON* or *MULTICHECKBOX*), a label, and a list of possible answers;
- A list of *Resources* representing the resources used by the agent. Each resource can be a URL, a PDF, a *Word* document (DOCX), an *Excel* spreadsheet (XLSX), or a text file (TXT). Each resource includes metadata such as ID, name, type, title, description, and content;
- A *Log* list that records actions performed by the agent, with a message and completion status, useful for tracking and debugging the agent's activity;
- The name of the AI model used (*model*).

The system's state management architecture is designed to accommodate a wide range of task complexities by supporting both ephemeral and persistent memory modes. This dual-mode capability allows agents to flexibly handle anything from lightweight interactions to complex multi-step operations. To maintain coherence across concurrent workflows, the architecture enables synchronized updates even within asynchronous execution environments.

To ensure scalability and modularity, state is organized using a layered strategy. At the core of this approach is a combination of *React Context Providers* and custom hooks, which together facilitate the separation of global and domain-specific logic while preserving accessibility and performance throughout the application.

4.1.1 *React Context Providers*

The application relies heavily on *React*'s *Context API* to manage shared state in a modular and isolated way. Several dedicated context providers are used to encapsulate specific areas of functionality. At the core is the *CopilotContext*, which serves as the primary source of global application state. It coordinates key aspects such as interaction with the AI assistant, function invocation logic, agent state and lifecycle management, API configuration, session tracking, chat prompts and suggestions, as well as UI indicators like loading states.

Complementing this is the *CopilotMessagesContext*, which is responsible for maintaining the chat message stream. It supports frequent real-time updates as users engage with the assistant, ensuring a responsive and dynamic experience.

Although the system architecture supports dynamic model selection, in this project, only one model is actively used: *OpenAI*'s *GPT-4.1*. As such, the *ModelSelectorContext* is present for architectural completeness but operates in a fixed configuration, simplifying model management by consistently routing requests through a single model.

4.1.2 Custom hooks for stateful logic

To encapsulate complex and reusable logic, the system uses a set of custom *React* hooks:

- *useCoAgent*: Manages the lifecycle of an AI agent's state, including initialization, updates, and thread handling. It abstracts away the underlying mechanisms for interacting with *AgentState*;
- *useCopilotChat*: Encapsulates the chat logic and provides handlers for user interaction, message flow, and assistant responses;
- *useCopilotChatSuggestions*: Dynamically generates real-time suggestions and prompt completions from the AI, enhancing the conversational experience.

By abstracting logic into reusable hooks, the system minimizes code duplication, promotes consistency across components, and streamlines the development of AI-driven interfaces.

4.1.3 State transitions and side effect handling

State transitions within the application are primarily managed through dedicated functions provided by custom hooks and context objects. These include:

- *setState* for performing localized updates to the agent's internal state;
- Context methods such as *setAction*, *removeAction*, or *setCoAgentStateRender* for orchestrating more complex interactions between the agent and the UI.

To handle side effects and respond to changes in application state, the system relies on *React*'s *useEffect* hook. This ensures that updates to both agent logic and user interface components are executed only when relevant dependencies change, thereby preserving performance and avoiding unnecessary re-renders.

4.1.4 Benefits of the architecture

The chosen state management strategy delivers multiple benefits across both developer experience and application performance:

- Separation of concerns: Each part of the application state is managed in isolation, improving modularity and reducing coupling;
- Targeted updates: Context providers and custom hooks enable fine-grained control over state changes, preventing unnecessary re-renders;
- Optimized performance: Memoization and selective rendering strategies help maintain responsiveness even under heavy interaction;
- Maintainability: A modular, abstracted architecture simplifies debugging and future enhancements;
- Scalability: The system can accommodate more complex workflows and growing data requirements without compromising reliability.

Together, these design principles ensure that agent memory, contextual logic, and user-facing behavior are managed cohesively, making the platform both robust and adaptable.

4.2 Content upload and management

The content upload and management system is built to flexibly handle both local file uploads and remote resource retrieval. It offers an intuitive interface that enables users to seamlessly upload, edit, and manage a variety of content types. By enforcing consistent formatting and metadata standards, this system ensures that all incoming data is properly normalized. Its design

is critical to maintaining data integrity across the entire pipeline, enabling smooth integration with the platform's intelligent operations.

4.2.1 Supported resource types

The application categorizes content into five distinct resource types, as defined in the *ResourceType* enumeration: PDF, URL, DOCX, XLSX, and TXT. This classification enables the system to process a diverse range of formats, including web-based content, text documents, and spreadsheets, ensuring broad compatibility with various user inputs.

4.2.2 Resource structure and metadata

Each resource is represented by a well-defined structure that includes a unique identifier (*id*), the original file or URL name (*name*), the resource type (*type*), a user-defined title and description, and an optional content field. The *content* field stores the *base64*-encoded version of the file, which is used during transmission and backend processing. This consistent resource schema ensures smooth integration across the frontend and backend modules.

4.2.3 Upload workflow

Content can be added to the system through two primary methods: uploading local files or submitting URLs.

- File upload: Users can upload files in PDF, DOCX, XLSX, or TXT formats, with a strict size limit of 420 KB. Upon selection, files are immediately read and encoded to *base64* via the *FileReader* API. The application determines the file type based on the file extension and assigns the corresponding *ResourceType*. Both the original file name and the encoded content are retained for downstream processing;
- URL input: Alternatively, users may submit web-based resources by entering a valid URL, accompanied by a title and an optional description. The system fetches the webpage asynchronously using *aiohttp*, parses the HTML with *BeautifulSoup* to extract relevant text, and converts it into a clean Markdown format via *html2text*. This ensures consistent formatting and structure, aligning web-based content with the treatment of uploaded files.

4.2.4 Resource management features

The resource management interface provides comprehensive capabilities for adding, editing, and deleting resources. To prevent duplicates, the system checks resource filenames before creation. Each resource is assigned a *Universally Unique Identifier* (*UUID*) at the time of creation to ensure distinct identification. Users interact with resources through the *AddResourceDialog* and *EditResourceDialog* components, which facilitate input, modification, and removal of both metadata and content. All changes are reflected immediately in the application state.

Resource deletion is performed using the resource's unique ID, guaranteeing accuracy and minimizing the risk of unintended data loss. The resource list updates reactively, delivering real-time visual feedback to users.

4.2.5 User interface and experience

The user interface presents a resource list and provides intuitive controls for adding, editing, or deleting items. Uploaded file names are clearly displayed, allowing users to easily remove files

with a single click. Resource addition and editing are managed through modal dialogs, creating a clean and focused interaction flow. The system provides real-time feedback during file selection and validation to enhance usability and reduce common mistakes.

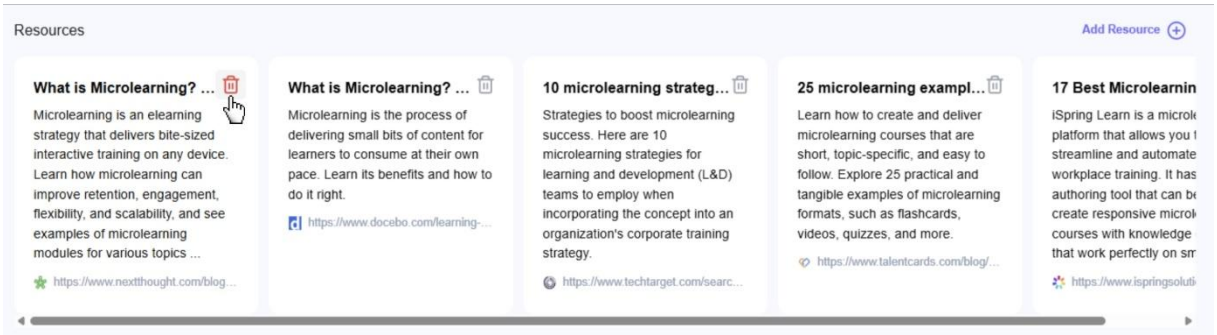


Figure 4.1: Interface for viewing, adding, editing, and deleting resources within the web application

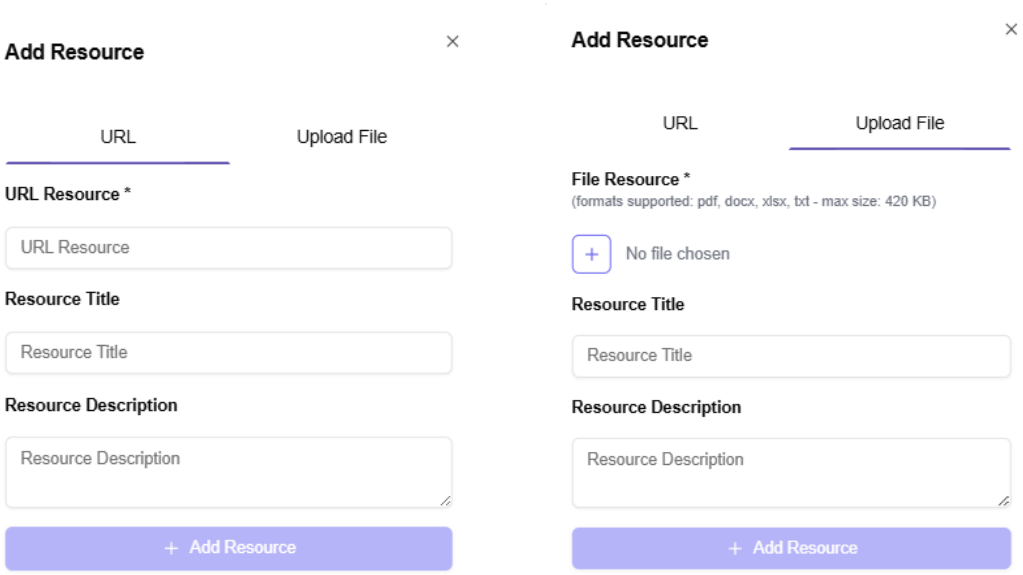


Figure 4.2: Example of a user session interacting with the application to add new resources

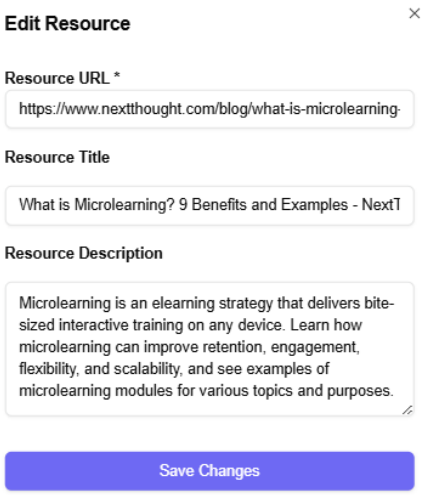


Figure 4.3: Example of a user session interacting with the application to edit an already existing resource

4.2.6 File management logic

The system enforces strict validation to accept only supported file types. Once a file is selected, it is automatically encoded into *base64* to enable the transmission between client and server, as well as backend processing. On the backend, temporary file storage is used for decoding and converting files into readable formats. Text extraction is handled using specialized libraries: *PyMuPDF* for PDFs, *python-docx* for *Word* documents, and *pandas* for *Excel* spreadsheets, while TXT files are decoded directly. These temporary files facilitate the transformation of content into a standardized textual format suitable for further processing. Additional utilities support file validation, type detection, and format conversion.

4.2.7 Security and validation mechanisms

To ensure robustness and prevent misuse, several layers of validation are in place:

- File type verification ensures that only .pdf, .docx, .xlsx, and .txt files are accepted;
- Required fields such as title and name are validated before submission;
- Errors encountered during upload or content extraction are caught and communicated to the user.

These validations are applied on both the client and server sides to ensure data integrity and enhance application security.

4.2.8 UI state management

Managing user interface state is essential for delivering a smooth and intuitive experience. The application extensively uses *React state hooks* to control dialog visibility, file handling, and form inputs. Key elements include:

- Dialog states: Modal visibility for adding and editing resources is governed by boolean flags such as *isAddResourceOpen* and *isEditResourceOpen*, enabling clear and controlled user navigation;
- File upload states: Variables like *uploadedFileName* and *isFileNameSet* monitor the file upload process, influencing UI behavior like displaying the filename, toggling save buttons, and permitting file removal;
- Resource form states: Separate state objects for *newResource* and *editResource* hold the input data during creation or modification. The *originalUrl* is preserved during edits to prevent accidental changes, allowing for intentional updates and easy rollback when canceled.

Transitions between states are managed via dedicated functions responsible for initializing dialog contents, resetting states on closure, and synchronizing the resource list after any change. This multi-layered state management approach ensures a consistent, reliable, and user-friendly interface.

4.2.9 Utility functions

Utility functions play a vital role in promoting modularity and code reuse across the application. They are grouped into several key categories:

- ID generation and normalization: Functions like *generateId()* create unique identifiers for resources, while *normalizeResources()* ensures all resources adhere to a consistent format, simplifying both UI rendering and backend processing;

- File management utilities: Helpers such as *getFileType()* and *formatFileSize()* extract metadata and present readable information, helping validation and improving the display of file details in the interface;
- Error handling: The custom *ResourceError* class categorizes different error types, and *handleResourceError()* centralizes error management. This approach enhances maintainability and guarantees uniform user feedback across the system;
- Validation functions: Methods including *isValidFileType()*, *isValidFileSize()*, and *validateFile()* enforce upload constraints, preventing unsupported files from being processed and providing immediate alerts to users when issues arise.

Together, UI state and utility functions form the backbone of the application's robustness. While UI state ensures responsiveness and clarity for the end user, utility functions abstract away common logic, reduce duplication, and enhance maintainability.

4.2.10 Backend integration

Uploaded files are sent to the backend in base64 format. The backend decodes these files, temporarily stores them, and processes them using the appropriate parser for their type. The temporary file infrastructure enables isolation and cleanup, reducing security risks and resource usage.

Caching mechanisms are in place for downloaded URLs, ensuring that repeated processing is avoided. This improves performance and ensures consistent content availability for AI pipelines.

4.3 Online resource retrieval

To support automated content generation, the system includes an advanced mechanism for searching and managing online resources. At the core of the online search capability lies the integration with the *Tavily API*, a service designed for fast and accurate web querying. The application establishes a *Tavily* client using an *API key* obtained securely from environment variables, avoiding the exposure of sensitive information in the codebase. This setup enables the application to execute external searches programmatically and retrieve structured results.

The primary logic for performing searches is encapsulated in a function called S*earch Node*. This function is responsible for orchestrating the full search process. Given a list of search queries, it initiates logging for tracking progress, sends each query to *Tavily*, and handles the response by extracting relevant data and updating the internal application state. Once complete, it logs the conclusion of the operation.

To support automated content generation, the system incorporates a sophisticated mechanism for retrieving and managing web-based information. Upon executing a search query, the system identifies and selects the top three to five most relevant results based on *Tavily*'s ranking algorithm. Each result is enriched with metadata, including the source URL, title, brief description, and content type (URL). When available, the actual content of the resource is also extracted and stored. The search process itself is designed to be asynchronous and non-blocking. Once initiated, each query is dispatched to *Tavily*, and its results are handled independently. This architecture allows the system to remain responsive and scalable, especially when dealing with multiple concurrent queries.

The application represents each result as a structured resource object that encapsulates essential metadata and full textual content. The HTML content is fetched and converted into a readable format. The system dynamically tracks the state of each resource, including its availability, download status, processing logs, and caching metadata. This dynamic management ensures consistency and performance during intensive or large-scale operations.

The user interface plays a central role in how resources are presented and managed. Retrieved items are displayed as interactive cards containing the title, description, URL (complete with favicon), and resource type. Users can remove individual resources, edit details, or upload additional documents to expand the dataset. This visual approach simplifies user interaction and provides immediate feedback, which is essential for applications involving research, synthesis, or assisted writing.

To ensure robustness, the system implements comprehensive error-handling mechanisms. It detects and manages various failure scenarios, including download interruptions, invalid URLs, unsupported file formats, content decoding errors, and response timeouts. In all such cases, the affected resources are excluded from further processing, while unaffected operations continue without interruption. This fault-tolerant design is essential for maintaining stability in real-world network and data conditions.

Performance is further enhanced by a local caching mechanism that stores previously retrieved and processed resources. This reduces the number of redundant calls to the *Tavily API* and minimizes loading times when frequently used content is accessed. Caching helps to improve responsiveness and efficiency overall, especially when the system is deployed in environments with limited connectivity or high resource turnover.

4.4 Automated generation of learning modules

The automated generation of instructional modules represents a significant advancement in the field of educational technology, particularly in the area of microlearning. This system is designed to autonomously create interactive and pedagogically effective modules, structured as slide-based presentations enriched with quizzes and visual content. These learning modules are designed to support rapid knowledge acquisition and are ideal for a variety of educational contexts, from corporate training to self-paced academic study.

At the heart of the system lies a clear and coherent structure. Each module consists of a sequence of informative slides, seamlessly integrated with interactive quizzes that assess the learner's understanding of the presented material. The content is supported visually by automatically generated images tailored to the theme of each slide. In addition, the modules are equipped with intuitive navigation tools that enhance user engagement and ensure a smooth progression through the learning path.

The generation process begins with the collection of educational resources. The system is capable of ingesting various formats provided by the user, including web links, PDF documents, *Word* files, *Excel* spreadsheets, and plain text. Furthermore, if needed, the system can autonomously search for relevant materials online, selecting from a wide range of languages to ensure accessibility for a global audience.

Once the resources are collected and processed, the content of the documents is converted into microlearning objects. These objects are then used to build modular and targeted instructional

elements. Advanced AI models are used to facilitate this transformation. In particular, AI plays a central role in generating the core learning components:

- Slide generation is conducted using Large Language Models that interpret and summarize the source material into structured slide components formatted as *JSON* objects. Each slide typically includes a title, a brief explanatory text, and an image prompt derived from the content. Images are then automatically generated using the *DALL·E* model, based on these prompts. The layout and language style of each slide are dynamically adjusted according to pedagogical heuristics to enhance clarity and retention.
- Quiz creation is also handled through language models, which are prompted to identify key concepts within the source material. These concepts are then rephrased into well-structured questions, with the generation of plausible distractors to support both single-choice and multiple-choice formats. The system allows for the configuration of difficulty levels and the adjustment of topic coverage, making the quizzes adaptable to different learning goals and audiences.

The introductory slide of each module serves a pivotal role, containing the general title of the module, a generated image, and a concise summary limited to 15 to 25 words. Subsequent slides are dynamically composed, with flexible formats that may include a combination of titles, images, and descriptions. Some slides may focus solely on visual content, while others pair imagery with explanatory text. The closing slide mirrors the structure of the opening slide, summarizing the module with a concluding title, image, and brief descriptive commentary. The system employs a variety of templates to structure the slides.

Each slide is meticulously crafted to ensure clarity and coherence. Titles are designed to be brief and impactful, guiding the learner through the core themes. Descriptions are typically composed of 60 to 120 words, written in a clear and accessible style. To aid readability and highlight key concepts, the content is formatted using HTML tags: bold text marks important keywords, while italicized words add emphasis. The images accompanying each slide are not stock visuals but are generated based on contextually relevant prompts, ensuring consistency with the narrative and educational purpose of the module.

To evaluate comprehension, the system integrates interactive quizzes. These may be single-choice questions (*radiobuttons*) or multiple-choice questions (*multicheckboxes*). Each quiz is designed with pedagogical rigor, including features such as a passing score threshold set at 80%, immediate feedback on answers, and the ability to retry the quiz. Importantly, module progression is often gated, preventing learners from advancing until they have successfully completed the required assessments.

User interactivity is further enhanced through navigational controls that allow learners to move forward or backward within the module, jump to the beginning or end, retry quizzes, or revisit previously seen content. A visual progress indicator helps maintain motivation and orientation throughout the learning experience.

The system supports high levels of personalization. It accommodates multiple languages, ensuring a broad reach. The visual identity of generated images is consistent, maintaining aesthetic and cognitive harmony.

The final output of the system is a structured package containing a unique module ID, the module's title and description, a complete list of the resources used, metadata relevant to microlearning, and a detailed breakdown of the templates applied during generation.

Through this integrated and automated process, the system empowers educators, trainers, and content creators to generate engaging and interactive learning modules rapidly. By blending rich content, dynamic quizzes, and intelligent design supported by state-of-the-art AI, it ensures a user-centered learning experience that is both effective and intuitive.

4.5 Editing and customization of content

The system described herein is a comprehensive slide editor designed to enable AI-assisted customization, providing users with an intuitive and efficient workflow to create presentations that are not only content-rich but also dynamically engaging.

A key strength of this system lies in its set of AI-powered tools that substantially simplify the content creation process. Users can use Artificial Intelligence to automatically generate complete slides from given topics or outlines, receive smart suggestions for content improvements, and obtain tailored recommendations for relevant images. These features greatly speed up the presentation building process and prove especially beneficial for users who may lack experience in visual design or narrative structuring.

The editing interface is deliberately designed to maximize usability and clarity. At its core, a central panel prominently displays the current slide, while navigation controls are positioned directly below the slide. A chatbot panel is located beside the workspace, enabling real-time interaction and assistance. This layout ensures users receive instant visual feedback during AI interactions, as any adjustments are reflected immediately on the slide panel. Furthermore, the interface grants quick access to all major functions, fostering a productive and user-friendly editing environment. Underpinning this live-update system is a robust state management architecture that consistently tracks all slide data and metadata, maintaining reliability and coherence throughout the entire editing session.

Within this environment, users have extensive control over their presentations. They can seamlessly add new slides or remove unwanted ones, as well as merge or split existing slides according to their needs. Navigating between slides is made easy with straightforward forward and backward controls. Beyond basic slide management, users can deeply customize individual slides by editing titles, descriptions, and images, adapting the structure of the text through different formats such as bullet points or highlighted keywords to better suit their narrative goals. The system also supports interactive content creation, allowing users to add, modify, or delete questions and answers, and choose among various types of quizzes (*radiobuttons* or *multicheckboxes*) to enrich the presentation.

In addition to content editing, users can modify attached files, providing further flexibility in managing supplementary materials associated with each slide. To further enhance accessibility and global reach, the system offers automatic translation of content into multiple languages, facilitating multilingual presentations without additional manual effort.

Collectively, these capabilities empower users to produce sophisticated and engaging presentations with minimal friction, leveraging intelligent automation and a thoughtfully designed interface that balances power with ease of use.

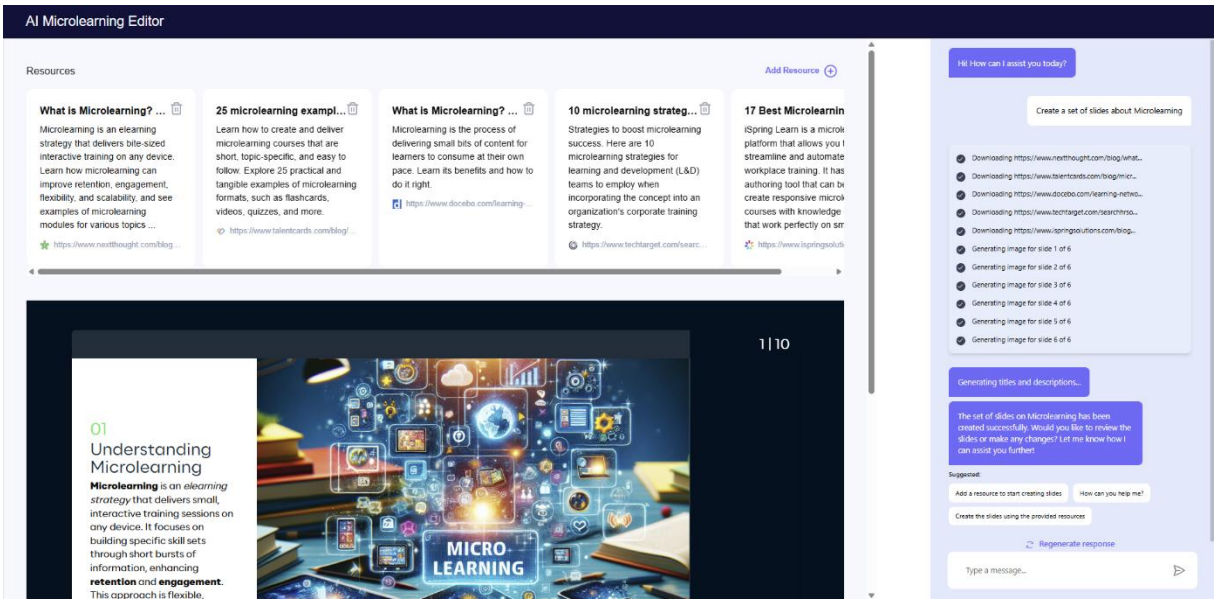


Figure 4.4: Example of a user session interacting with the application to add another *radiobutton* quiz

4.6 Integration of a chatbot for user assistance

The chatbot integration and conversation features operate through a carefully designed system that enhances user interaction with context-aware assistance. When a user first engages with the chatbot, it begins with an initial welcome message, which can be customized via the *CopilotChatLabels* interface. This message is a default greeting that sets a friendly and approachable tone. The chatbot maintains a context-aware state throughout the interaction, allowing it to deliver suggestions and responses that are relevant to the ongoing conversation.

A key component of the system is its sophisticated suggestion mechanism. Suggestions are dynamically generated based on several factors, including the current state of the conversation, the tools and actions available within the application, the user's previous interactions, and the specific context in which the chatbot operates. This ensures that the prompts offered are always pertinent and tailored to the user's needs.

The suggestion system is highly configurable. Developers can configure parameters such as the minimum and maximum number of suggestions (which default to one and three, respectively) and can also define custom instructions to guide how suggestions are generated. Visual consistency is ensured through the use of predefined *CSS* classes that align the styling of suggestions with the overall application design.

To maintain responsiveness without overloading the system, suggestions are generated using a debounced approach with a 1000ms delay. This means suggestions are not triggered immediately but are instead refreshed thoughtfully, minimizing unnecessary computations while keeping recommendations up to date.

The system recalculates suggestions whenever there are changes in the conversation state, the addition of new messages, or shifts in the loading status. If a new event occurs while a previous suggestion is still being processed, the system aborts the ongoing generation and starts over, ensuring that users always receive relevant and current suggestions.

Each suggestion consists of a title (displayed as a clickable button) and an associated message that is submitted when the button is clicked. These suggestions are presented in a dedicated section positioned just below the chat messages. The interface is designed to handle multiple suggestions at once, updating them dynamically in real time as the conversation evolves. This ensures users are continually presented with relevant and actionable prompts throughout their interaction.

Integration with the overall chat flow is seamless. Suggestions appear naturally after messages are sent or received, with the system managing loading indicators and handling error scenarios gracefully.

Finally, robust error handling mechanisms are built into the system. These safeguard against failures in suggestion generation, interruptions during processing, invalid states, and network issues, ensuring a smooth and reliable user experience.

Overall, this implementation delivers a robust and flexible framework that guides users through conversations with contextually relevant suggestions, making interactions more natural, intuitive, and productive.

4.7 Multilingual support

To address the need for multilingual accessibility within the chatbot component of the application, *GPT-4.1* was also used as the core engine for language translation and natural language understanding. This model was selected due to its robust performance across a wide range of languages and its ability to maintain semantic fidelity during translation tasks. Within the implementation pipeline, user inputs in various languages are first detected using a lightweight language identification module. Once the language is identified, *GPT-4.1* processes the input to both comprehend the user's intent and, if necessary, translate the response from a base language into the target language. This ensures that interactions remain fluid, natural, and contextually appropriate for users regardless of their language preferences.

In cases where the uploaded resources (such as textual, PDF, WORD, EXCEL content, or URLs) are written in different languages from each other or in a language other than the one used by the user to interact with the AI, the chatbot prompts the user to explicitly choose the target language to be used to generate slide content. This interactive clarification step ensures that the generated materials align with the user's expectations and intended audience. Furthermore, *GPT-4.1*'s generative capabilities were also employed to localize responses by incorporating idiomatic expressions and culturally relevant language patterns, rather than relying solely on literal translations. This strategy significantly enhances user experience and engagement, especially in educational settings where clarity and relatability of content are crucial.

4.8 Development tools

The backend is developed within a well-defined reproducible environment to allow consistency and ease of maintenance. Dependency management and project configuration are streamlined using *Poetry*, which facilitates precise package versioning and virtual environment creation. Environment variables are securely handled through *python-dotenv* to safeguard sensitive configuration details.

For logging, the backend utilizes *Python*'s built-in logging module, extended with contextual features to support centralized log aggregation and detailed analysis. During development, the backend runs on the *Uvicorn* server, configured with debugging and hot-reload enabled, allowing rapid feedback and efficient troubleshooting.

On the frontend side, as well as the wider *TypeScript* ecosystem, the project employs a *monorepo* structure managed with *PNPM* (version 9.5.0). This approach enhances dependency management across multiple packages within a single codebase and improves the development workflow. The primary technologies powering the frontend include *TypeScript* (version 5.2.3) and *React* (version 18), while *Next.js* (version 14.2.15) is used as the web framework, offering optimized routing and server-side rendering capabilities for an improved developer experience.

The development environment emphasizes maintainability and code quality, employing *ESLint* and *Prettier* with custom configurations for linting and formatting, respectively. Pre-commit hooks are enforced using *Husky*, which ensures that linting and testing scripts run before any code is committed, thereby maintaining a consistent codebase.

Styling and UI composition are managed using *Tailwind CSS* for utility-first design, while *Radix UI* and *Headless* UI provide accessible, unstyled components that support consistent design patterns across the application.

In terms of AI integration, the project includes support for various *SDK*s, with *LangChain* used to orchestrate advanced AI workflows. This setup allows seamless interaction with language models and supports both inference and toolchain integration.

Infrastructure is defined using *AWS CDK*, allowing infrastructure as code practices that align closely with the rest of the application's *TypeScript* codebase.

The *TypeScript* configuration is set up in strict mode, ensuring strong type safety throughout the codebase. The project includes multiple configurations tailored to different environments, such as a base configuration and another specific to *Next.js*.

A comprehensive set of development scripts is provided to manage builds for various environments, run the development server with hot reloading, execute tests and linters, and perform clean builds. These scripts facilitate a streamlined development experience and ensure that every part of the codebase adheres to rigorous standards for quality and consistency.

Overall, this project is built upon a modern, well-structured development environment that emphasizes type safety, scalability, code quality, and developer productivity through a cohesive and carefully selected toolchain.

Chapter 5

5. EXPERIMENTAL RESULTS AND FUTURE WORK

In the first part of the chapter, the manual testing phase that was conducted is presented, reporting an example of user interaction with the chatbot for creating and editing educational content. In this phase, multiple evaluation metrics are reported and analyzed, such as execution time, computational cost, economic cost, Time To First Token (TTFT), and completion token, in order to demonstrate the capabilities of the system, identifying performance bottlenecks and validating its applicability in the real world. Aggregate metrics on repeated experiments are then reported. These tests were conducted internally within the company, involving colleagues and department heads. Their participation provided valuable insights into the usability and core functionality of the platform.

In the second part of the chapter, a plan is defined for future automated test implementations. This will include unit, integration, system, end-to-end, and security testing in order to ensure scalability, robustness, and maintainability of the platform under broader usage scenarios.

5.1 Manual testing methodology

To ensure a comprehensive and objective evaluation of the system's performance, a set of key performance indicators was defined. These metrics were selected based on their relevance to the goals of the project and their prevalence in related work in the fields of AI-based educational technology and enterprise automation.

The main evaluation criteria include:

- Execution time: Measures the time required to complete core system operations, such as content ingestion, pre-processing, image generation, and module generation. The values reported below were recorded using *LangSmith*;
- Computational cost: Tracks the usage of the CPU during execution, providing insight into the platform's efficiency. The values reported below were extracted from print statements embedded in the *Python* code. The computational cost (% CPU) measurements were performed on a system with the following hardware and software specifications:

Attribute	Details
Processor (CPU)	13th Gen Intel® Core™ i7-1355U
Physical cores	10
Logical cores	12
Base frequency	1.70 GHz
Operating system	Microsoft Windows 11 Pro

Table 5.1: System specifications (hardware and software)

- Economic cost: Takes into account costs to run the system. It includes expenses related to the *OpenAI* APIs, *Tavily* APIs, and *Amazon Web Services*. The values reported below were calculated using *LangSmith* for *OpenAI*-related costs. *AWS* costs were estimated using the pricing on the official website. Costs for *Tavily* are currently zero, as we are using the free tier, but pricing information from their official website is also included for reference;
- Input prompt: Measures the length of input provided to the system, expressed in number of tokens. This metric helps evaluate the impact of prompt complexity on response latency and overall efficiency. The values sent were recorded via *LangSmith*;
- Time To First Token: Measures the latency between a user's request and the generation of the first output token, giving an idea of the perceived responsiveness. The values reported below were recorded using *LangSmith*;
- Completion token: Represents the number of pieces of text (tokens) generated in response to a request. In terms of length, on average, a token is about 0.75 words or 4 characters in English, but these are language dependent. The values reported below were recorded using *LangSmith*.

Experimental tests were conducted using a diverse dataset of enterprise documents, and performance was measured under controlled and repeatable conditions.

5.1.1 Example of user interaction flow: from user query to microlearning slides

As mentioned above, this section presents a sample user interaction with the platform's AI assistant through a conversational interface. The goal of the experiment is to evaluate the model's ability to generate structured content on demand.

Initially, the assistant is asked to generate a complete set of slides on the topic of microlearning and, subsequently, the user requests a series of modifications to the generated content. Below are the various interaction steps along with the corresponding performance metrics. In addition to performance metrics, there is also a description of what happens behind the scenes during the interaction.

The conducted manual test is executed locally.

Computational costs, when not specified, are negligible.

First interaction (slide generation)

Figure 5.1: Example of a user session interacting with the application to create a set of slides about microlearning

The user uploads one or more resources (in the specific example, just one PDF) and specifies that the slide generation must occur only using these resources, without external search. When the user asks to generate slides, the message is forwarded to the backend via the *CopilotKit* pipeline.

The backend system, based on a graph architecture composed of processing nodes, receives the user's input along with the uploaded resource. The uploaded file, if it is a PDF (as in the example above), is first encoded in *base64* to ensure proper handling within the *CopilotKit* data flow. This encoded file and the user's textual prompt are sent to the backend service, which starts a multi-step pipeline to process, analyze, and convert the resource into structured microlearning content.

The first component in this pipeline is the *Download Node*. In a general use case, this node retrieves external content; however, since the user explicitly asked the system to work only with the provided PDF file, no actual download occurs. Instead, the encoded PDF is handed off directly to the next processing node.

The core of the generation logic, therefore, takes place in the *Chat Node*. Here, the system decodes the file, extracts its textual content, and constructs a formatted prompt for the Large Language Model. This prompt includes several important constraints: the model must use only the uploaded resource; it must structure the content as a presentation composed of slides; and it must follow specific formatting guidelines regarding slide titles, descriptions, and the possible use of illustrative elements.

Once the language model processes the prompt and completes its internal generation phase, it triggers a specific backend tool called *WriteReport*. This tool is responsible for compiling the generated slides into a structured *JSON* report. Each slide in the report includes a title, a short description, and the prompt used later for image generation. This *JSON* report is then returned to the backend system, where it undergoes validation and further processing (using parsing functions).

After validation, the final report is serialized and stored in the backend state under a dedicated field (*state["report"]*). At this point, the frontend interface becomes aware of the new data. Specifically, the *ResearchCanvas* component detects the updated report, decodes its contents, and passes it to the slide visualization module (i.e., *challenges-app* web component). The user now sees the generated presentation in a structured and editable format. They can review the slides, change titles and descriptions, add new resources or remove provided ones, modify quizzes, or request automatic translation of the content into another language.

It is important to emphasize that in this flow, no external data retrieval is performed. The S*earch Node* of the system's graph, which typically handles external searches, is entirely bypassed. In enterprise environments, this strict adherence to the user's instruction to use only the uploaded file is a key aspect that ensures that the generated content only contains controlled information retrieved by the provided resources.

Below are the metrics recorded in this phase:

- *Download node* duration: 15.01 s
- *Chat node* duration: 610.02 s
- *Chat node* duration: 10.82 s
- Total duration: 635.85 s
- TTFT: 78,841 ms
- Tokens: 6,153
    - Prompt: 11,497/ $0.022994
    - Completion: 59/ $0.000472
- Cost (*OpenAI*): $0.023466
- Computational cost (% CPU):

Image	Computational cost (% CPU)
1	0.8
2	0.7

3	0.9
4	1.1
5	1.3
6	0.7
7	1.1
8	0.9
9	1.0
10	1.0
11	1.5
12	1.1
13	1.2
14	1.0

Table 5.2: Computational cost for image generation

Second interaction (slide modification)

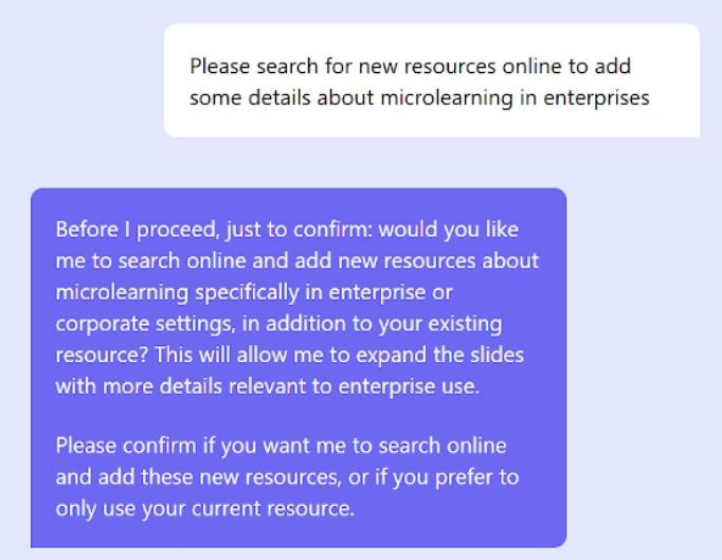


Figure 5.2: Example of a user session interacting with the application while asking for more detailed slides

After the initial slide generation is completed, users can request to enhance and refine the existing content. In this case, the user asks the system to add some details about microlearning in enterprises.

As the request enters the backend, the processing pipeline is reactivated, following a similar route as before. The input is first passed through the *Download Node*, which is the initial node of the graph. Then, the *Chat Node* serves as the central logic node for interpreting user intentions and constructing the appropriate prompts for the Large Language Model.

At this stage, the AI system asks the user whether they would like the model to search for external sources in order to enrich the content or whether the addition of details should be based strictly on the original resources already provided.

This explicit request serves an important purpose: to maintain user control. Some users may prefer to keep the learning content strictly tied to internal documentation, ensuring that all generated output remains verifiable, traceable, and compliant with corporate standards. Others may welcome the opportunity to enrich the material with broader knowledge from external

publicly available sources. By offering a clear choice, the system empowers users to guide the direction of the enrichment process.

Below are the metrics recorded in this phase:

- *Download node* duration: 0.29 s
- *Chat node* duration: 11.01 s
- Total duration: 11.30 s
- TTFT: 5,650 ms
- Tokens: 8,385
    - Prompt: 8,307/ $0.016614
    - Completion: 78/ $0.000624
- Cost (*OpenAI*): $0.017238

Third interaction (online resource retrieval)

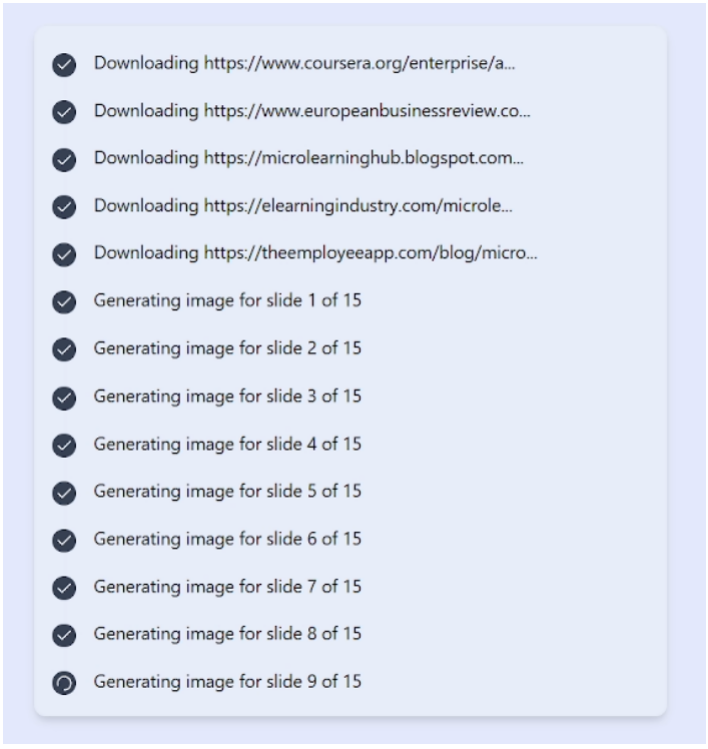


Figure 5.3: Example of a user session interacting with the application while asking for more detailed slides

The user responds to the system's request for clarification and, in this case, chooses to expand the initial presentation by inserting external information. The system does not regenerate the entire presentation from scratch, but when it receives an instruction that seems to be a continuation of the previous activity (thanks to the agent state, which preserves the session context), it promptly proceeds to make the requested change while keeping the remaining textual content unchanged. In this case, it adds a new slide, without modifying the text and structure of the others. However, even if the text of the previous slides remains unchanged, the images are still regenerated. This behavior is justified by the need to ensure visual consistency across all slides: even a minimal textual change (such as adding a detail or rewording) can imply a change in the conceptual content, tone, or focus of the slide, making it appropriate to also update the image to keep it aligned.

The process begins in the *Download Node* and then proceeds to the *Chat Node*. Here, the AI interprets the user's request (to enrich the existing content with additional web-sourced details) and creates a high-level plan for how to proceed: the model identifies the thematic areas or slide topics that require further elaboration and formulates targeted search queries accordingly. Then, these queries are passed to the next component: the *Search Node*.

The *Search Node* serves as the orchestrator of online information retrieval. It uses the *Tavily* API, which returns a ranked list of the most relevant results (usually from 3 to 5).

Each result includes a title, a URL, and a brief description. This content is retrieved, parsed, cleaned, converted from HTML into readable text, and stored in the *Resource* object. The system tracks the state of each resource and handles any failures.

Once the search is complete, the newly acquired resources are sent back through the *Download Node*. At this point, the node processes the incoming set of external documents. With all resources available (both the original internal files and the newly fetched external ones), the pipeline returns once again to the C*hat Node*. In this phase, the AI is prompted to re-analyze the slides in light of the expanded content base. It identifies opportunities to enhance existing sections, introduce clarifying examples, or add new explanatory notes. The model is instructed to preserve the structure and tone of the original slides, changing only necessary textual parts and regenerating the images.

At this point, the *AgentState* is changed: it is enriched with all the information related to the new resources. This change is propagated to the frontend, which updates the interface and displays the new content to the user.

Finally, a final pass through the *Chat Node* is triggered, whose purpose is to create the final response to inform the user that the slides have been successfully enriched.

Below are the metrics recorded in this phase:

- *Download node* duration: 0.34 s
- *Chat node* duration: 11.01 s
- *Search node* duration: 48.07 s
- *Download node* duration: 7.43 s
- *Chat node* duration: 510.79 s
- *Chat node* duration: 14.69 s
- Total duration: 592.33 s
- TTFT: 8,078 ms
- Tokens: 50,253
    - Prompt: 50,199/ $0.100398
    - Completion: 54/ $0.000432
- Cost (*OpenAI*): $0.10083
- Computational cost (CPU):
    - *Download node*: 97.7%
    - *Search node*: 64.9%
    - *Download node*:

Resource	Computational cost (% CPU)
1	30.2
2	18
3	15.9
4	22
5	29.4

Table 5.3: Computational cost for resource downloading

    - Image generation:

Image	Computational cost (% CPU)
1	1.6
2	2.3
3	2.2
4	4.6
5	2.1
6	3.1
7	1.9
8	1.9
9	1.9
10	2.3
11	2.9
12	4.1
13	3.1
14	2.3
15	1.7

Table 5.4: Computational cost for image generation

Fourth interaction (slide translation)

In this interaction, the user asks the assistant to translate the set of slides into Italian.

As with all actions initiated by the user, the workflow begins in the *Download Node*, where the *AgentState* is retrieved (it also includes the content of the resources).

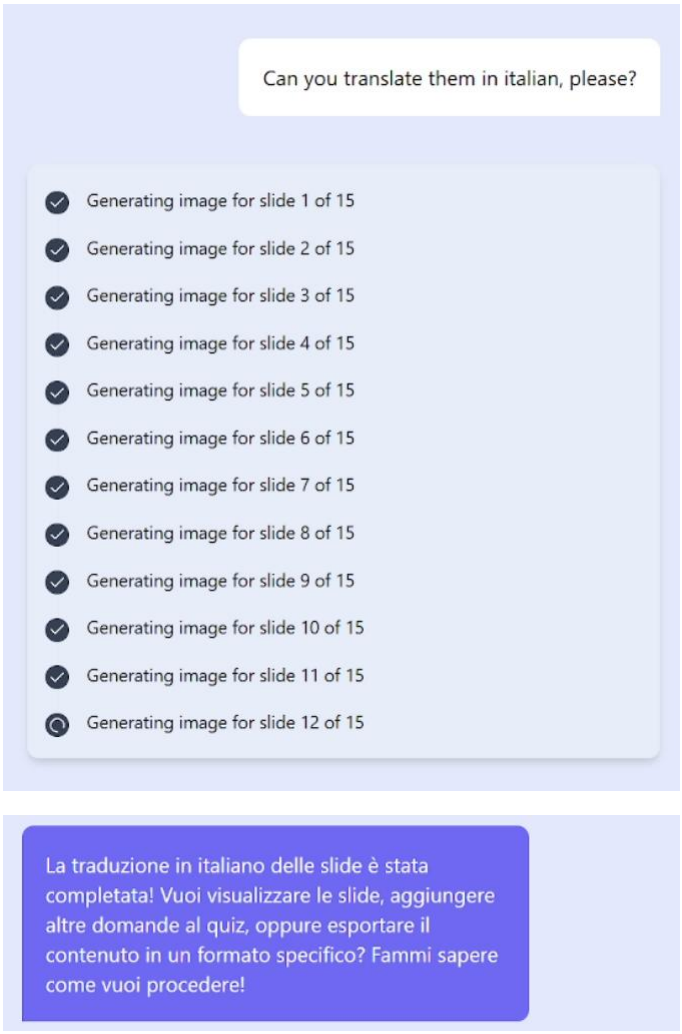


Figure 5.4: Example of a user session interacting with the application while asking for the translation of the entire content

Then, the flow continues to the *Chat Node* and here the actual translation of the content takes place. The AI receives a prompt that explicitly asks to perform a translation of each slide from English into Italian (also quizzes are modified). The model is guided not only to translate text accurately, but also to preserve the integrity of the slide format, including headings, bullet points, examples, etc.

The translation is performed in-place within the slide structure: rather than regenerating slides from scratch, the model modifies only the necessary parts (always regenerating images). Once completed, the translated slides are written back into the application state.

Finally, the workflow returns to the *Chat Node* in order to create a message for the user that confirms that the set of slides has been successfully translated. This message, along with the updated slide content, is sent back to the frontend and the user can view the new version.

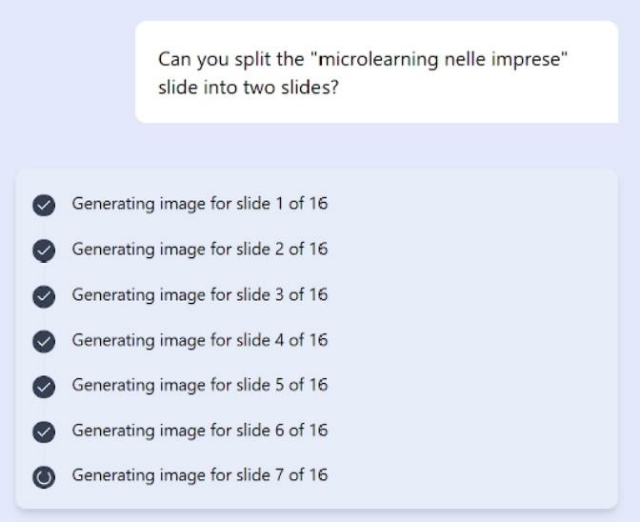
Below are the metrics recorded in this phase:

- *Download node* duration: 0.30 s
- *Chat node* duration: 492.01 s
- *Chat node* duration: 13.57 s
- Total duration: 506.2 s
- TTFT: 10,650 ms
- Tokens: 39,161
    - Prompt: 39,109/ $0.078218
    - Completion: 52/ $0.000416
- Cost (*OpenAI*): $0.078634
- Computational cost (CPU):

Image	Computational cost (% CPU)
1	2.1
2	1.5
3	1.8
4	2.5
5	3.0
6	1.8
7	1.6
8	1.5
9	1.7
10	3.3
11	2.7
12	3.0
13	2.2
14	2.4
15	2.2

Table 5.5: Computational cost for image generation

Fifth interaction (slide split)



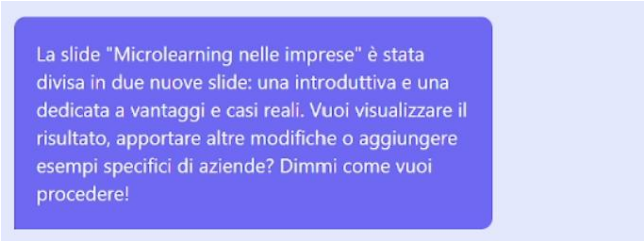


Figure 5.5: Example of a user session interacting with the application while asking to split a slide

At this point, the user asks the system to split a dense slide into two, in order to allow the learner to focus on one aspect at a time and improve the understanding of the content.

The pipeline starts, as always, with the *Download Node*, where the current state (including the most recent version of the slides) is loaded.

The process then moves to the *Chat Node*, where the model is prompted to interpret and execute the user's intent: splitting a single slide into two separate ones. The model analyzes the previously generated slide and looks for the right point to split, which allows each new slide to remain autonomous and meaningful, but still coherent with the others. The new slides preserve the original tone and formatting while redistributing content in a way that enhances cognitive load management for the learner. In this node, image regeneration is performed.

Once this process is complete, the set of slides is updated in the internal state and passed again through the *Chat Node*. This second call to the node is not about modifying content but about composing the final system response to confirm that the operation has been successfully completed.

This response, along with the modified set of slides, is sent to the frontend interface. The user can view the updated set of slides, now including the two new slides in place of the original one. The interface allows further iterative refinements, ensuring full user control over the final structure.

Below are the metrics recorded in this phase:

- *Download node* duration: 0.30 s
- *Chat node* duration: 492.01 s
- *Chat node* duration: 13.57 s
- Total duration: 506.2 s
- TTFT: 8,738 ms
- Tokens: 40,309
    - Prompt: 40,237/ $0.080474
    - Completion: 72/ $0.000416
- Cost (*OpenAI*): $0.08089
- Computational cost (% CPU):

Image	Computational cost (% CPU)
1	1.7
2	1.7
3	2.7

4	2.3
5	4.7
6	2.2
7	2.4
8	2.4
9	2.5
10	2.5
11	3.1
12	2.8
13	2.1
14	3.3
15	2.1
16	3.2

Table 5.6: Computational cost for image generation

5.1.2 Aggregated metrics from the interaction flow

After detailing the five key user interactions within the AI-powered content generation pipeline (ranging from the initial slide creation to enrichment, translation, and structural edits) and recording the various associated metrics, we can get a general idea of the overall system performance during this single run. Therefore, this section presents a summary of the most relevant metrics collected during the execution of these interactions, with the goal of offering a quantitative perspective on the behavior, efficiency, and responsiveness of the underlying architecture. Indeed, by measuring system latency, costs, and number of tokens across different stages, we can better understand how well the pipeline supports real-time microlearning content creation and further editing. Depending on the nature of the metric, values are presented either as totals or as averages.

- Total duration: 2,251.88 s
- TTFT: 22,391.4 ms
- Tokens: 144,261
    - Prompt: 149,349/$0.28698
    - Completion: 315/$0.00252
- Cost (*OpenAI*): $0.277592
- Average computational cost (CPU): 6.05%
- Number of slides generated: 38
- Number of images generated: 61

All reported data comes from direct observation of the system during execution and analysis of its logs. This approach ensures a realistic representation of the system usage, useful for evaluating performance, estimating throughput, and guiding future improvements.

From an economic point of view, the costs associated with searching for online resources should also be considered. With *Tavily*'s free plan, 1,000 credits per month are available at no cost. Each chatbot cycle, which includes a basic search (1 credit) and the basic extraction of 3-5 resources (1 credit), consumes 2 credits. So, with the free plan, up to 500 complete queries per month can be managed for free. Once this threshold is exceeded, there is the need to upgrade to a paid plan like the *Project* one (the most basic one after the free one, which guarantees 3,000

credits per month), where each credit costs $0.0075. In this case, each complete query would cost 2 × $0.0075 = $0.015. Anyway, in our case:

- *Tavily* cost: $0 (free plan)

Moreover, in the case the execution takes place on *AWS* infrastructure, the costs related to the services used must also be added to the previous costs, in particular those related to *AWS Lambda*, *AWS Secrets Manager*, and *Amazon CloudWatch*.

Actually, it is currently not possible to precisely isolate costs related to a single execution of the application, but we can estimate them using the pricing on the official *AWS* website.

AWS Lambda costs depend on how many times the function is executed and how long it remains active, on the memory allocated, and on the region in which it runs. For *AWS Secrets Manager*, you pay a monthly fee for each secret stored, such as API keys, tokens, or credentials, and additional costs when the secrets are accessed. Finally, with *Amazon CloudWatch*, costs are related to the logs generated by *Lambda*, i.e., the amount of data sent and stored there.

To provide a concrete estimate, we can consider an application used by 20 users monthly. It is assumed that each user accesses the app once a day, for a total of 30 uses per month, thus producing 600 monthly invocations (20 users × 30 accesses).

Each access involves:

- the activation of two *Lambda functions*: a *UI Lambda* (4096 MB, average duration 300 seconds) and an *Agent Lambda* (1024 MB, average duration 300 seconds);
- two requests to the *Secrets Manager* to retrieve the API keys stored;
- the generation of about 5 KB of logs, saved in *Amazon CloudWatch*.

AWS Lambda costs depend on how many times the function is executed and how long it remains active (in seconds), on the memory allocated (in GB), and on the region in which it runs (*eu-west-1*). The cost per execution can be calculated using the following formula:

$$\text{Cost}_{\text{Lamba}} = \text{Memory (GB)} \times \text{Duration (s)} \times \text{Cost (\$)} \times \text{Iterations}$$

In the case of *UI Lambda* (4096 MB = 4 GB, duration = 300 s), the reference price is $0.0000000667 per GB-second, so:

$$\text{Cost}_{\text{UI\_Lambda}} = 4\text{GB} \times 300\text{s} \times \$0.0000000667 \times 600 = \$0.048024$$

For *Agent Lambda* (1024 MB = 1 GB, duration = 300 s), with price $0.0000000167 per GB-second, the total cost is:

$$\text{Cost}_{\text{Agent\_Lambda}} = 1\text{GB} \times 300\text{s} \times \$0.0000000167 \times 600 = \$0.003006$$

The total cost for *Lambda* functions is then:

$$\text{Total}_{\text{Lambda}} = \$0.048024 + \$0.003006 = \$0.05103$$

AWS Secrets Manager charges a flat fee of $0.40/month for each secret stored. This is in addition to a variable cost for API calls, equal to $0.05 per 1,000 requests.

With a single secret stored and 600 monthly requests:

$$\text{Cost}_{\text{Secrets}} = \$0.40 \times 2 = \$0.80$$

$$\text{Cost}_{\text{Request}} = 600 \times 0.05 / 1{,}000 = \$0.03$$

$$\text{Total}_{\text{SecretManager}} = \$0.80 + \$0.03 = \$0.83$$

Amazon CloudWatch charges a log ingestion fee of $0.50 per GB. In our case, each invocation produces about 5 KB of logs. The total volume generated in a month is therefore:

$$\text{Total}_{\text{CloudWatch}} = 600 \times 5 \text{ KB} = 3{,}000 \text{ KB} = 3 \text{ MB} = 0.003 \text{ GB}$$

$$\text{Cost}_{\text{CloudWatch}} = 0.003 \times 0.50 = \$0.0015$$

However, since *AWS* offers 5 GB/month of free logs, this cost is within the free threshold. The actual cost will therefore be:

$$\text{Total}_{\text{CloudWatch}} = \$0.00$$

In conclusion:

$$\text{Total}_{20\_\text{users}} = \$0.83 + \$0.0015 = \$0.8315$$

One of the most important advantages of the serverless approach is the linear scalability of costs. If the number of users were to grow, for example, to 100 monthly users (3,000 invocations), the costs would increase proportionally. Applying the same formulas, we obtain:

- *Lambda:*

$$\text{Cost}_{\text{UI\_Lambda}} = 4\text{GB} \times 300\text{s} \times \$0.0000000667 \times 3{,}000 = \$0.24012$$
$$\text{Cost}_{\text{Agent\_Lambda}} = 1\text{GB} \times 300\text{s} \times \$0.0000000167 \times 3{,}000 = \$0.01503$$
$$\text{Total}_{\text{Lambda}} = \$0.24012 + \$0.01503 = \$0.25515$$

- *Secrets Manager*:

$$\text{Total}_{\text{SecretManager}} = 0.80 \times 1 + 3{,}000 \times 0.05 / 1{,}000 = \$0.80 + \$0.15 = \$0.95$$

- *CloudWatch*: 15 MB of logs (still under free threshold)

$$\text{Total}_{\text{CloudWatch}} = \$0.00$$

- Total:

$$\text{Total}_{100\_\text{users}} = \$0.25515 + \$0.95 = \$1.20515$$

This model shows how the system can grow with users; costs increase only based on real traffic, avoiding waste due to an infrastructure that is designed too large from the beginning (trying to predict the expansion of the user base).

5.1.3 Aggregated metrics from repeated interaction experiments

Repeated experiments were conducted to simulate real-world usage scenarios, involving the insertion of different input formats (such as URLs, PDFs, *Word*, *Excel*, and text files), according to the previously defined interaction flow model. The simulated activities included common operations such as content creation, editing, and translation, useful for analyzing system performance and stability.

By aggregating the results of these 10 sessions, a set of average metrics that reflect the typical resource consumption of the application was computed:

- Average total duration per interaction: 1,349.5 s
- Average TTFT: 13,601 ms
- Average number of tokens per interaction: 87,993

- Average *OpenAI* API cost per interaction: $0.341704
- Average *Tavily* API cost per interaction: $0 (free plan, under 500 complete queries)
- Average computational load (% CPU): 3.29
- Average number of slides generated: 31
- Average number of images generated: 40

These metrics confirm the system's ability to deliver high-quality output while maintaining acceptable latency and low resource consumption. They also serve as a benchmark for future optimizations and for estimating performance at scale.

5.1.4 Discussion and analysis

The tests conducted demonstrate that the platform guarantees good performance in terms of low costs and quality of content. Although it is not very fast in generating content, it still proves effective in using Artificial Intelligence to transform knowledge, solving critical issues such as the high cost of producing teaching materials and the lack of customization. By working on enhancing the parallelization of operations, this aspect can certainly be improved as well.

Among the main strengths that emerged:

- Automation: Automatic generation of content with minimal human intervention, reducing time and costs;
- Flexibility: Modular architecture that facilitates the integration of new features and adaptation to different business contexts;
- User Experience: Multilingual support and chatbots improve accessibility and interaction.

The addition of human feedback could also further improve the quality and evolution of the system.

5.2 Automated testing methodology

This section presents an automated testing strategy planned for future development phases to ensure greater performance, reliability, repeatability, and security of the microlearning application. Given the multifaceted nature of the platform (which integrates AI-powered content generation, multilingual processing capabilities, and interactive learning features), a rigorous and multi-layered testing approach is essential.

The planned automated testing activities include:

- Unit testing: Individual software modules (such as document parsers, text summarizers, language models, and quiz generators) will be tested in isolation to confirm they behave as expected under a variety of input conditions. Testing will be automated using frameworks such as *PyTest* for *Python* components;
- Integration testing: This will focus on the interactions among various modules in the processing pipeline. For example, it will test the successful handoff from the content extraction module to the summarization engine, and then to the learning content formatter. This type of testing is crucial to validate the system's internal cohesion and data flow correctness;

- System testing: The platform will be tested as a whole to confirm that the complete application meets functional and non-functional requirements. This includes testing the deployment architecture, APIs, databases, and frontend interface in coordination;
- End-to-End (E2E) testing: These tests will simulate real-world usage scenarios, covering the entire workflow from document upload through to user interaction with the generated learning material and the chatbot;
- Security and input validation testing: Specific attention will be given to input validation, file upload handling, user authentication, and access control. Tests will check for common vulnerabilities such as injection attacks, insecure file handling, and broken session management. Static and dynamic analysis tools like *Bandit* and *OWASP ZAP* will assist in identifying security flaws.

By transitioning towards automated testing, the development process will benefit from faster feedback loops, improved test coverage, and enhanced confidence in platform stability and security before broader deployment.

5.2.1 Planned performance evaluation

Performance testing is essential to ensure that the platform maintains good levels of responsiveness and efficiency, even under high loads. The following evaluation strategies are planned:

- Load testing: Concurrent users will be simulated to measure the scalability of the system. Using tools like *Locust, it* will be possible to apply virtual loads and monitor indicators, such as average response time, throughput, and resource use;
- Stress testing: Tests will be performed beyond normal operating limits to analyze the behavior of the system under extreme conditions. This kind of testing will help identify critical thresholds and points of failure, such as memory leaks or CPU saturation, which may not emerge during standard loads;
- Scalability and concurrency testing: These tests aim to verify how the platform reacts to increasing number of concurrent users or parallel processes, evaluating the possibility of scaling horizontally or vertically. The infrastructure, with elements such as load balancers, container orchestrators, and caching mechanisms, will be analyzed in terms of flexibility and adaptability;
- Profiling and bottleneck identification: To identify bottlenecks in the backend, especially in AI models or data processing, performance profiling tools such as *New Relic*, *Datadog*, or *Python*'s *cProfile* will be used. The goal will be to optimize execution times and improve memory usage in crucial operations, such as AI-based content generation.

In addition to quantitative metrics, logs and exception reports will be analyzed to identify anomalies and performance degradation patterns.

5.2.2 User experience and usability testing

A core pillar of platform success lies in its usability, especially since it targets users with diverse backgrounds and varying levels of digital literacy. Usability testing will be undertaken in multiple phases to gather qualitative and quantitative insights from end users.

- Participant recruitment: Users will be selected to reflect the platform's key personas, including training managers, subject matter experts, and employees participating in learning programs;
- Test design: Participants will be asked to complete representative tasks, such as uploading content, interpreting AI-generated summaries, navigating between slides, and interacting with AI. These sessions will be conducted in controlled environments, either remotely via screen sharing or in person;
- Data collection techniques:
    - Observation and think-aloud protocols: Observers will note user behavior and friction points as participants verbalize their thought processes;
    - Surveys and questionnaires: Standardized instruments like the System Usability Scale (SUS), Net Promoter Score (NPS), and custom Likert-scale surveys will be administered to assess satisfaction and usability;
    - Interviews: Follow-up interviews will allow participants to elaborate on their experiences, expectations, and any perceived limitations;
    - Interaction logs: Behavioral data from user sessions will be anonymized and analyzed to detect common navigation paths, drop-off points, and interaction errors.

These insights will be triangulated to prioritize usability issues and design refinements.

5.2.3 Conclusions

This chapter has provided a comprehensive overview of the results obtained during the development and evaluation of the AI-powered microlearning platform. The combination of computational efficiency and content accuracy highlights how AI technologies can revolutionize learning flows in enterprises.

The metric analysis confirmed the potential of the platform, especially in the automatic generation of content and its usability in different contexts. However, areas of optimization also emerged, especially regarding content generation speed.

Overall, the results obtained support the basic idea of this thesis: AI is able to increase the content quality in terms of coherence, clarity, and effectiveness in corporate training. This reflects the broader process of digital transformation of the world of work, where intelligent systems are becoming protagonists in the evolution of learning and innovation.

Chapter 6

6. CONCLUSIONS AND FUTURE DEVELOPMENTS

This final chapter presents an overall reflection on the results obtained, outlining both the main strengths of the application and the critical issues encountered, as well as suggesting possible future developments. Finally, the aspects that make the platform a promising tool for knowledge transfer in dynamic and complex contexts, such as corporate ones, are highlighted.

6.1 Limitations and challenges encountered

While the current implementation of the platform has demonstrated promising capabilities and lays a solid foundation for intelligent and automated slide generation, several limitations and challenges have been identified throughout the development and testing phases. These constraints span technical, architectural, and user experience domains, and must be acknowledged to contextualize the scope of the platform in its current state and guide future enhancements.

6.1.1 Limited input modalities

At present, the platform primarily accepts textual inputs for content generation. This restricts its ability to process and extract information from non-textual or multimedia sources such as audio recordings, video lectures, scanned documents, or handwritten notes. As a result, a significant portion of enterprise knowledge (particularly content from meetings, interviews, and legacy files) is excluded from the knowledge ingestion pipeline.

6.1.2 Inefficiencies in image generation

Image generation is performed sequentially, which can result in noticeable delays, especially when creating a set of slides with many visual elements. Moreover, the system relies on generating images based on textual slide content rather than extracting or reusing visuals from uploaded materials, which may limit the contextual relevance and consistency of visual assets.

6.1.3 Lack of a visual editing interface

Currently, users must interact with the platform exclusively through a conversational interface, even for minor edits. This can become cumbersome when users wish to make quick, targeted adjustments to individual slide elements such as titles, bullet points, or formatting. The absence of a canvas-style editor limits editing efficiency and user control over the final output.

6.1.4 Basic authentication scheme

The existing authentication mechanism is based on Basic Authentication, which lacks robustness and scalability. This approach can present security risks and does not support more

advanced user management features such as token-based session control, role-based access, or integration with enterprise authentication systems (e.g., Single Sign-On).

6.1.5 File size limitations and lack of persistent storage

The current system architecture transmits files via *base64* encoding directly between the frontend and backend. This technique introduces a hard limit of approximately 420 KB for uploaded documents, which restricts the ability to process large or complex inputs. Additionally, the absence of persistent file storage means that uploaded resources, generated slides, and chatbot conversations are not retained across sessions, limiting version control and long-term usability.

6.1.6 Absence of structured evaluation

To date, the platform has not undergone formal usability testing or performance benchmarking. Without structured user studies or adherence to established evaluation frameworks (e.g., SUS, *Nielsen* heuristics), it remains difficult to identify usability bottlenecks or validate its impact in real-world educational or enterprise environments.

6.1.7 Fixed slide template and limited customization

The platform currently employs a static slide template, without offering users the ability to customize layout, color scheme, fonts, or branding elements. This reduces flexibility for enterprise clients who may require consistency with corporate visual identities or desire tailored presentation formats for different contexts.

6.1.8 Inaccessibility of source files

Users are not currently able to download the original source that was used to generate slides. This complicates content revision and restricts the potential reuse of materials for other educational or professional workflows.

6.1.9 Dependency on external APIs and third-party services

A notable architectural limitation lies in the platform's strong dependence on third-party services and APIs (most prominently, the *OpenAI GPT* models for content generation and the *Tavily* search engine for information retrieval). While these tools provide advanced capabilities out of the box, their integration introduces several concerns:

- Data privacy risks, particularly when handling sensitive or proprietary enterprise content that must be transmitted to external servers;
- Variable response times and latency, which can impact the perceived responsiveness of the platform;
- Availability risks, where service downtimes or interruptions in third-party API services may render the system partially or entirely unusable;
- Output variability, as LLM-generated content may not always meet expected standards of factual accuracy or consistency;
- Cost-related concerns, due to reliance on usage-based pricing models, which may not scale economically with high user volumes.

These factors highlight a critical trade-off between leveraging state-of-the-art AI capabilities and maintaining control over infrastructure, privacy, and long-term operational costs.

6.1.10 Summary of key limitations

Limitation	Impact
Limited input modalities	Inability to ingest audio/video/scanned images/handwritten content
Sequential image generation	Increased waiting times during slide creation
No canvas-style editor	Low editing efficiency and flexibility
Basic Authentication	Poor security and scalability
File size restriction	Inability to process large documents
No persistent storage	Loss of files and outputs across sessions
No formal evaluation	Lack of validated UX and performance metrics
Static slide template	Limited branding and layout flexibility
No source file access	Reduced transparency and reuse potential
Dependency on external APIs	Risks related to data privacy, availability, latency, output quality, and costs

Table 6.1: Summary of key limitations in the application

6.2 Potential enhancements and future work

Building upon the current implementation, several future decisions are envisioned to enhance the platform's capabilities, improving its usability and increasing its adaptability across various enterprise and educational contexts. These potential enhancements span improvements in input handling, user experience, infrastructure, and system intelligence, and they are critical to unlocking the platform's full potential.

6.2.1 Support for expanded input modalities

One of the most significant areas for enhancement involves broadening the range of input formats that the platform can process. By incorporating multimedia inputs (such as audio recordings, video lectures, scanned images, and handwritten notes), the system would become significantly more inclusive and versatile in capturing enterprise knowledge. This would require integration with advanced speech-to-text services (e.g., *Google Cloud STT, Microsoft Azure STT, Amazon Transcribe,* or *IBM Watson STT*) and Optical Character Recognition engines to extract meaningful content from diverse data sources (such as meetings, webinars, or legacy documents).

6.2.2 Advanced visual asset handling

Future iterations of the platform could support both improved generation and intelligent extraction of images. Instead of solely relying on text-based prompts to create visuals, the system could analyze uploaded documents to extract embedded images, diagrams, or figures relevant to the slide content. This would improve contextual fidelity.

6.2.3 Parallel image generation

To address latency issues during content generation, the system could be enhanced to support parallelized image rendering. This would allow multiple visual assets to be created simultaneously, significantly reducing wait times and improving the perceived responsiveness of the slide creation process.

6.2.4 Visual slide editing interface

Introducing a visual, drag-and-drop interface for slide editing would greatly improve user control and interaction efficiency. Users would be able to directly modify elements (such as titles, bullet points, images, and formatting) without relying entirely on chatbot prompts. This would enable quick and targeted refinements and offer a hybrid workflow that balances automation with manual precision.

6.2.5 Flashcard module for spaced repetition

Integrating a flashcard system based on the platform's generated content would support knowledge retention through spaced repetition techniques. This would provide users with a lightweight structured method for reviewing and assessing their understanding of key concepts.

6.2.6 Formal evaluation and benchmarking

Conducting structured usability studies and performance benchmarks would allow the platform to be evaluated systematically. User feedback could be collected using standardized tools such as the System Usability Scale or through heuristic evaluations. Both qualitative and quantitative data would guide refinements in user interface design, model output quality, and system accessibility.

6.2.7 LLM model selection and customization

Enabling users to choose from a selection of Large Language Models would allow for customization based on performance, use-case specificity, language support, or compliance requirements. This modular approach could also facilitate experimentation with fine-tuned or domain-specific models tailored to enterprise applications.

6.2.8 Enhanced authentication and security

To improve platform security and align with enterprise standards, the current Basic Authentication mechanism could be replaced with more robust and scalable solutions such as *OAuth* 2.0 or *JSON Web Tokens* (JWT). These enhancements would facilitate secure user session management, multi-factor authentication, and role-based access control.

6.2.9 Cloud-Based persistent storage

Introducing persistent cloud storage would provide long-term access to uploaded documents, generated slides, and conversation history. This would support version control, facilitate collaboration across sessions, and remove the existing file size constraints imposed by direct *base64* encoding.

6.2.10 Integration with enterprise ecosystems

To encourage adoption within corporate environments, the platform could be extended to support integration with existing Learning Management Systems, Single Sign-On frameworks, and enterprise compliance standards such as SCORM, xAPI, and ISO 27001. This would ensure compatibility with organizational workflows.

6.2.11 Mobile and offline access

Developing mobile applications and offline access modes would expand the platform's reach, enabling usage in low-connectivity environments and enhancing flexibility for asynchronous learners. Offline synchronization could further improve the user experience on mobile devices.

6.2.12 Customizable slide templates

Allowing users to choose from or create custom slide templates (adjusting parameters such as layout, color scheme, font style, and branding) would offer greater personalization and align outputs with institutional visual identities.

6.2.13 Access to underlying source files

Finally, enabling the download of underlying content sources (e.g., transcripts, markdown files, extracted summaries) would increase transparency, support iterative revisions, and facilitate downstream reuse for different documentation or training needs.

6.2.14 Strategies to mitigate API dependency

Given the current reliance on external APIs such as *OpenAI GPT* for natural language generation and *Tavily* for web search functionalities, future versions of the platform could explore strategies to reduce risk and increase system resilience. Several directions are worth considering:

- On-premise or self-hosted LLMs: Developing proprietary enterprise language models could significantly improve data privacy, reduce reliance on external providers, and allow fine-tuning on domain-specific datasets. This approach provides greater control over cost, performance, and compliance but requires substantial investment in hardware, expertise, and ongoing maintenance;
- Hybrid inference pipelines: Implementing a system that dynamically chooses among different models based on task complexity, cost, and user policy;
- Caching mechanisms: Implement a caching mechanism to reduce repeated API calls and lower latency.

These initiatives would help balance the benefits of external APIs with improved control over privacy, performance, and long-term sustainability, making the platform more robust and enterprise-ready.

6.2.15 Summary of future enhancement directions

Future work	Goal
Multimedia input support	Broader knowledge ingestion
Visual extraction/generation	Context-aware slide visuals
Parallel image rendering	Improved performance
Visual editor	Intuitive user control
Flashcards	Better knowledge retention
Usability studies	Evidence-based system improvement
Model selection	Customization and compliance
Secure authentication	Robust and scalable security
Persistent storage	File retention and versioning
Enterprise integration	Corporate adoption readiness
Mobile/offline support	Greater accessibility
Template customization	Visual branding flexibility
Source file access	Transparency and content reuse
Mitigate API dependency	Improve privacy, reliability, and cost control
Develop proprietary enterprise model	Full control over data, customization, and compliance

Table 6.2: Summary of future enhancement directions

6.3 Strengths of the developed application

Despite the limitations outlined previously, the developed AI-enhanced microlearning platform demonstrates several strengths that make it a highly promising tool for enterprise knowledge transfer:

- Seamless content automation: The platform significantly reduces the manual effort required for content creation by leveraging state-of-the-art AI models for summarization, quiz generation, slide composition, and image generation. This automation accelerates the development cycle and empowers subject matter experts to generate structured learning content with minimal technical intervention;
- User-centered design: The interface is intuitive and designed for users with varying technical backgrounds. The inclusion of a natural language chatbot further simplifies navigation and task execution, making the platform accessible to non-technical stakeholders;
- Modularity: Built with a modular architecture, the system is easily maintainable and extensible. Its design supports integration with additional AI services and content formats in the future, ensuring long-term adaptability;
- Multilingual capabilities: The integration of multilingual support broadens the platform's applicability across diverse linguistic contexts, enabling global organizations to translate training materials effortlessly;
- Real-time assistance: The chatbot assistant enhances usability by offering interactive help and reducing the learning curve for new users. It acts as a bridge between complex AI functionalities and user-friendly experiences;
- Cloud-native infrastructure: The underlying architecture supports cloud deployment, allowing the application to scale efficiently based on organizational demand.

These strengths collectively contribute to a versatile and forward-thinking learning platform, well-suited to the dynamic and evolving needs of enterprise training environments.

6.4 Conclusion

In conclusion, the developed platform represents a solid starting point for the adoption of intelligent microlearning solutions in the context of corporate training. Among the main strengths are the effective automation of content, the user-friendly interface, the scalability of the architecture, the ability to operate in multiple languages, and the presence of a virtual assistant for real-time support. These features allow to improve the efficiency in the creation of training materials, reducing the time and costs traditionally associated with internal training.

At the same time, some limitations have emerged that open interesting ideas for the future evolution of the system, such as the absence of a visual editor, the limited management of images, and the need for formal evaluations of teaching effectiveness. The proposed future developments aim to overcome these critical issues, further enhancing accessibility, customization, and integration with real corporate environments.

Ultimately, the proposed approach proves to be consistent with the needs of modern organizations, oriented towards flexible, customizable, and scalable training solutions. The integration of Artificial Intelligence in this area not only makes content production more sustainable but also helps make the learning process more engaging, accessible, and effective.

BIBLIOGRAPHY

[1] A. Taylor e W. Hung, «The Effects of Microlearning: A Scoping Review», *Educ. Technol. Res. Dev.*, vol. 70, fasc. 2, pp. 363–395, apr. 2022, doi: 10.1007/s11423-022-10084-1.

[2] R. Sankaranarayanan, J. Leung, V. Abramenka-Lachheb, G. Seo, e A. Lachheb, «Microlearning in Diverse Contexts: A Bibliometric Analysis», *TechTrends*, vol. 67, fasc. 2, pp. 260–276, mar. 2023, doi: 10.1007/s11528-022-00794-x.

[3] J. Zhang e R. E. West, «Designing Microlearning Instruction for ProfessionalDevelopment Through a Competency Based Approach», *TechTrends*, vol. 64, fasc. 2, pp. 310–318, mar. 2020, doi: 10.1007/s11528-019-00449-4.

[4] «An assessment of a just-in-time training intervention in a manufacturing organization - ProQuest». Consultato: 8 aprile 2025. [Online]. Disponibile su: https://www.proquest.com/openview/b6826a9499190621b9bb95579b3d5d79/1?cbl=18750&pq-origsite=gscholar

[5] K. Leong, A. Sung, D. Au, e C. Blanchard, «A review of the trend of microlearning», *J. Work-Appl. Manag.*, vol. 13, fasc. 1, pp. 88–102, dic. 2020, doi: 10.1108/JWAM-10-2020-0044.

[6] B. Gross *et al.*, «Microlearning for patient safety: Crew resource management training in 15-minutes», *PLOS ONE*, vol. 14, fasc. 3, p. e0213178, mar. 2019, doi: 10.1371/journal.pone.0213178.

[7] J. B. Branzetti *et al.*, «Randomised controlled trial to assess the effect of a Just-in-Time training on procedural performance: a proof-of-concept study to address procedural skill decay», *BMJ Qual. Saf.*, vol. 26, fasc. 11, pp. 881–891, nov. 2017, doi: 10.1136/bmjqs-2017-006656.

[8] Y.-T. Cheng, D. R. Liu, e V. J. Wang, «Teaching Splinting Techniques Using a Just-in-Time Training Instructional Video», *Pediatr. Emerg. Care*, vol. 33, fasc. 3, p. 166, mar. 2017, doi: 10.1097/PEC.0000000000000390.

[9] K. E. Sawarynski e D. M. and Baxa, «Utilization of an online module bank for a research training curriculum: development, implementation, evolution, evaluation, and lessons learned», *Med. Educ. Online*, vol. 24, fasc. 1, p. 1611297, gen. 2019, doi: 10.1080/10872981.2019.1611297.

[10] P. Prasittichok e P. Smithsarakarn, «The Effects of Microlearning on EFL Students' English Speaking: A Systematic Review and Meta-Analysis», *Int. J. Learn. Teach. Educ. Res.*, vol. 23, fasc. 4, Art. fasc. 4, apr. 2024.

[11] R. Sankaranarayanan e S. Mithun, «Exploring the Effectiveness of AI-Enabled Microlearning in Database Design and Programming Course», in *2024 IEEE Frontiers in Education Conference (FIE)*, ott. 2024, pp. 1–7. doi: 10.1109/FIE61694.2024.10892916.

[12] M. A. Allela, B. O. Ogange, M. I. Junaid, e P. B. Charles, «Effectiveness of Multimodal Microlearning for In-service Teacher Training», *J. Learn. Dev.*, vol. 7, fasc. 3, Art. fasc. 3, nov. 2020, doi: 10.56059/jl4d.v7i3.387.

[13] H. Alamri, Lowell ,Victoria, Watson ,William, e S. L. and Watson, «Using personalized learning as an instructional approach to motivate learners in online higher education: Learner self-determination and intrinsic motivation», *J. Res. Technol. Educ.*, vol. 52, fasc. 3, pp. 322–352, lug. 2020, doi: 10.1080/15391523.2020.1728449.

[14] B. Sathiyaseelan, J. Mathew, e S. Nair, «Microlearning and Learning Performance in Higher Education: A Post-Test Control Group Study», *J. Learn. Dev.*, vol. 11, fasc. 1, Art. fasc. 1, mar. 2024, doi: 10.56059/jl4d.v11i1.752.

[15] L. P. A. Simons, F. Foerster, P. A. Bruck, L. Motiwalla, e C. M. Jonker, «Microlearning mApp raises health competence: hybrid service design», *Health Technol.*, vol. 5, fasc. 1, pp. 35–43, giu. 2015, doi: 10.1007/s12553-015-0095-1.

[16] A. Hesse, P. Ospina, M. Wieland, F. A. L. Yepes, B. Nguyen, e W. Heuwieser, «*Short communication:* Microlearning courses are effective at increasing the feelings of confidence and accuracy in the work of dairy personnel», *J. Dairy Sci.*, vol. 102, fasc. 10, pp. 9505–9511, ott. 2019, doi: 10.3168/jds.2018-15927.

[17] «View of Microlearning and its Effectiveness in Modern Education: A Mini Review». Consultato: 11 maggio 2025. [Online]. Disponibile su: https://tecnoscientifica.com/journal/apga/article/view/496/258
[18] T. N. Fitria, «Microlearning in Teaching and Learning Process: A Review», *CENDEKIA J. Ilmu Sos. Bhs. Dan Pendidik.*, vol. 2, fasc. 4, pp. 114–135, nov. 2022, doi: 10.55606/cendikia.v2i4.473.
[19] P. Opas, «THE IMPLEMENTATION OF TIKTOK TO PROMOTE ENGLISH LISTENING AND SPEAKING FOR EFL LEARNERS», *J. Fac. Educ. Pibulsongkram Rajabhat Univ.*, vol. 10, fasc. 1, Art. fasc. 1, giu. 2023, Consultato: 11 maggio 2025. [Online]. Disponibile su: https://so02.tci-thaijo.org/index.php/edupsru/article/view/261783
[20] I. nur Aziz e R. H. Sabella, «TikTok as Media of Learning English»:, *JEET J. Engl. Educ. Technol.*, vol. 2, fasc. 02, Art. fasc. 02, giu. 2021, doi: 10.59689/jeet.v2i02.51.
[21] M. Ouadoud, N. Rida, e T. Chafiq, «Overview of E-learning Platforms for Teaching and Learning», *Int. J. Recent Contrib. Eng. Sci. IT IJES*, vol. 9, fasc. 1, p. 50, mar. 2021, doi: 10.3991/ijes.v9i1.21111.
[22] «(PDF) A study of employees' utilization of microlearning platforms in organizations», *ResearchGate*, doi: 10.1108/TLO-07-2022-0080.
[23] R. P. Díaz-Redondo, M. Caeiro-Rodríguez, J. J. López-Escobar, e A. Fernández-Vilas, «Integrating micro-learning content in traditional e-learning platforms», 11 dicembre 2023. doi: 0.1007/s11042-020-09523-z.
[24] V. M. Bradley, «Learning Management System (LMS) Use with Online Instruction», *Int. J. Technol. Educ.*, vol. 4, fasc. 1, pp. 68–92, 2021.
[25] N. N. Mohd Kasim e F. Khalid, «Choosing the Right Learning Management System (LMS) for the Higher Education Institution Context: A Systematic Review», *Int. J. Emerg. Technol. Learn. IJET*, vol. 11, fasc. 06, p. 55, giu. 2016, doi: 10.3991/ijet.v11i06.5644.
[26] Y. N. Asrida, F. Amanda, e J. U. Fadilah, «Effectiveness and Limitations on Learning Management Systems (LMS) in Learning and Teaching: A Systematic Review», *ICOERESS*, vol. 1, fasc. 1, Art. fasc. 1, dic. 2024.
[27] M. Haghshenas, M. Khademi, e H. Kabir, «E-LEARNING AND AUTHORING TOOLS : At a Glance», 2012.
[28] «(PDF) Application of Artificial Intelligence in Employee Training and Development», ResearchGate. Consultato: 12 maggio 2025. [Online]. Disponibile su: https://www.researchgate.net/publication/378347256_Application_of_Artificial_Intelligence_in_Employee_Training_and_Development
[29] M. Ilić, V. Mikić, L. Kopanja, e B. Vesin, «Intelligent techniques in e-learning: a literature review», *Artif. Intell. Rev.*, vol. 56, fasc. 12, pp. 14907–14953, dic. 2023, doi: 10.1007/s10462-023-10508-1.
[30] R. Sajja, Y. Sermet, M. Cikmaz, D. Cwiertny, e I. Demir, «Artificial Intelligence-Enabled Intelligent Assistant for Personalized and Adaptive Learning in Higher Education», 19 settembre 2023, *arXiv*: arXiv:2309.10892. doi: 10.48550/arXiv.2309.10892.
[31] C. Halkiopoulos e E. Gkintoni, «Leveraging AI in E-Learning: Personalized Learning and Adaptive Assessment through Cognitive Neuropsychology—A Systematic Analysis», *Electronics*, vol. 13, fasc. 18, Art. fasc. 18, gen. 2024, doi: 10.3390/electronics13183762.
[32] C. R, K. S. Babu, K. S. Ranjith, A. K. Sinha, V. Neerugatti, e D. S. Reddy, «Enhancing E-learning Accessibility through AI(Artificial Intelligence) and Inclusive Design», in *2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI)*, gen. 2025, pp. 1466–1471. doi: 10.1109/ICMCSI64620.2025.10883148.
[33] A. M. Bettayeb, M. Abu Talib, A. Z. Sobhe Altayasinah, e F. Dakalbab, «Exploring the impact of ChatGPT: conversational AI in education», *Front. Educ.*, vol. 9, lug. 2024, doi: 10.3389/feduc.2024.1379796.
[34] P. Choudhary, R. Choudhary, e S. Garaga, «Enhancing Training by Incorporating ChatGPT in Learning Modules: An Exploration of Benefits, Challenges, and Best Practices», vol. 9, fasc. 11, 2024.

[35] J. Wang e W. Fan, «The effect of ChatGPT on students' learning performance, learning perception, and higher-order thinking: insights from a meta-analysis», *Humanit. Soc. Sci. Commun.*, vol. 12, fasc. 1, pp. 1–21, mag. 2025, doi: 10.1057/s41599-025-04787-y.
[36] H. Tabuchi *et al.*, «Comparative educational effectiveness of AI generated images and traditional lectures for diagnosing chalazion and sebaceous carcinoma», *Sci. Rep.*, vol. 14, fasc. 1, p. 29200, nov. 2024, doi: 10.1038/s41598-024-80732-4.
[37] J. Liu, X. Liu, e C. Yang, «A study of college students' perceptions of utilizing automatic speech recognition technology to assist English oral proficiency», *Front. Psychol.*, vol. 13, dic. 2022, doi: 10.3389/fpsyg.2022.1049139.
[38] T. Wilschut, F. Sense, e H. van Rijn, «Speaking to remember: Model-based adaptive vocabulary learning using automatic speech recognition», *Comput. Speech Lang.*, vol. 84, p. 101578, mar. 2024, doi: 10.1016/j.csl.2023.101578.
[39] M. Tanjga, «E-learning and the Use of AI: A Review of Current Practices and Future Directions», *Qeios*, mag. 2023, doi: 10.32388/AP0208.2.
[40] «AI Governance in Higher Education: Case Studies of Guidance at Big Ten Universities». Consultato: 20 maggio 2025. [Online]. Disponibile su: https://arxiv.org/html/2409.02017v1
[41] S. M. Espinoza Vidaurre, N. C. Velásquez Rodríguez, R. L. Gambetta Quelopana, A. N. Martinez Valdivia, E. A. Leo Rossi, e M. A. Nolasco-Mamani, «Perceptions of Artificial Intelligence and Its Impact on Academic Integrity Among University Students in Peru and Chile: An Approach to Sustainable Education», *Sustainability*, vol. 16, fasc. 20, Art. fasc. 20, gen. 2024, doi: 10.3390/su16209005.
[42] M. Alghodi e A. Shibani, «Privacy Enhancing Technologies for Next-Generation E-Learning Systems: A Systematic Review», vol. 01, fasc. 01, 2025.
[43] «[2407.18745] FairAIED: Navigating Fairness, Bias, and Ethics in Educational AI Applications». Consultato: 18 maggio 2025. [Online]. Disponibile su: https://arxiv.org/abs/2407.18745
[44] A. Nguyen, H. N. Ngo, Y. Hong, B. Dang, e B.-P. T. Nguyen, «Ethical principles for artificial intelligence in education», *Educ. Inf. Technol.*, vol. 28, fasc. 4, pp. 4221–4241, apr. 2023, doi: 10.1007/s10639-022-11316-w.
[45] W. Holmes, F. Iniesto, S. Anastopoulou, e J. G. Boticario, «Stakeholder Perspectives on the Ethics of AI in Distance-Based Higher Education», *Int. Rev. Res. Open Distrib. Learn.*, vol. 24, fasc. 2, pp. 96–117, 2023, doi: 10.19173/irrodl.v24i2.6089.
[46] B. Miller, T. Rutherford, A. Pack, e A. Johnson, «Bridging the SCORM and xAPI Gap: The Role of cmi5».
[47] S. Panagiotakis, K. Papadokostaki, K. Vassilakis, e A. Malamos, «Towards a novel and LMS-free Pervasive Learning System exploiting the Experience API», *EAI Endorsed Trans. Creat. Technol.*, vol. 5, fasc. 16, p. 156383, ott. 2018, doi: 10.4108/eai.13-7-2018.156383.
[48] «The Experience API—Liberating Learning Design : Research Library | The Learning Guild». Consultato: 24 maggio 2025. [Online]. Disponibile su: https://www.LearningGuild.com/insights/177/the-experience-apiliberating-learning-design/
[49] K. Kitto *et al.*, *Learning analytics beyond the LMS: enabling connected learning via open source analytics in «the wild»*. Canberra, ACT: Australian Government Department of Education, Skills and Employment, 2020.
[50] «Experience API (xAPI) Standard», ADL Initiative. Consultato: 24 maggio 2025. [Online]. Disponibile su: https://www.adlnet.gov/projects/xapi/
[51] L. Sanchez, J. Penarreta, e X. Soria Poma, «Learning management systems for higher education: a brief comparison», *Discov. Educ.*, vol. 3, fasc. 1, p. 58, mag. 2024, doi: 10.1007/s44217-024-00143-5.
[52] «[1805.06266] A Unified Model for Extractive and Abstractive Summarization using Inconsistency Loss». Consultato: 30 maggio 2025. [Online]. Disponibile su: https://arxiv.org/abs/1805.06266
[53] «Introducing GPT-4.1 in the API». Consultato: 30 maggio 2025. [Online]. Disponibile su: https://openai.com/index/gpt-4-1/

[54] «Putting GPT-4o to the Sword: A Comprehensive Evaluation of Language, Vision, Speech, and Multimodal Proficiency». Consultato: 30 maggio 2025. [Online]. Disponibile su: https://www.mdpi.com/2076-3417/14/17/7782
[55] M. Lewis *et al.*, «BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension», 29 ottobre 2019, *arXiv*: arXiv:1910.13461. doi: 10.48550/arXiv.1910.13461.
[56] C. Raffel *et al.*, «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer», *J. Mach. Learn. Res.*, vol. 21, fasc. 140, pp. 1–67, 2020.
[57] J. Zhang, Y. Zhao, M. Saleh, e P. Liu, «PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization», in *Proceedings of the 37th International Conference on Machine Learning*, PMLR, nov. 2020, pp. 11328–11339. Consultato: 30 maggio 2025. [Online]. Disponibile su: https://proceedings.mlr.press/v119/zhang20ae.html
[58] S. Alaei, *An Automated Discharge Summary System Built for Multiple Clinical English Texts by Pre-trained DistilBART Model*. 2023. Consultato: 30 maggio 2025. [Online]. Disponibile su: https://urn.kb.se/resolve?urn=urn:nbn:se:su:diva-222616
[59] I. Beltagy, M. E. Peters, e A. Cohan, «Longformer: The Long-Document Transformer», arXiv.org. Consultato: 30 maggio 2025. [Online]. Disponibile su: https://arxiv.org/abs/2004.05150v2
[60] «Pricing». Consultato: 10 luglio 2025. [Online]. Disponibile su: https://openai.com/api/pricing/
[61] «Welcome to PyPDF2 — PyPDF2 documentation». Consultato: 30 maggio 2025. [Online]. Disponibile su: https://pypdf2.readthedocs.io/en/3.x/
[62] *pdfplumber: Plumb a PDF for detailed information about each char, rectangle, and line.* Python. Consultato: 30 maggio 2025. [OS Independent]. Disponibile su: https://github.com/jsvine/pdfplumber
[63] *pdfminer: PDF parser and analyzer*. Consultato: 30 maggio 2025. [Online]. Disponibile su: http://github.com/euske/pdfminer
[64] «PyMuPDF 1.26.0 documentation». Consultato: 30 maggio 2025. [Online]. Disponibile su: https://pymupdf.readthedocs.io/en/latest/
[65] *python-docx: Create, read, and update Microsoft Word .docx files.* Python. Consultato: 30 maggio 2025. [OS Independent]. Disponibile su: https://github.com/python-openxml/python-docx
[66] «openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 3.1.3 documentation». Consultato: 30 maggio 2025. [Online]. Disponibile su: https://openpyxl.readthedocs.io/en/stable/
[67] «pandas - Python Data Analysis Library». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://pandas.pydata.org/
[68] «python-pptx — python-pptx 1.0.0 documentation». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://python-pptx.readthedocs.io/en/latest/
[69] «Python File Open». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://www.w3schools.com/python/python_file_open.asp
[70] «An_overview_of_Tesseract_OCR_Engine-libre.pdf». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://d1wqtxts1xzle7.cloudfront.net/52539350/An_overview_of_Tesseract_OCR_Engine-libre.pdf?1491626802=&response-content-disposition=inline%3B+filename%3DAn_overview_of_Tesseract_OCR_Engine.pdf&Expires=1748692112&Signature=V7ZPBR1fnrT1IJxNfF0WFHELAB~VOM24bxow3nTN5Gr2~wGneLg04aot5JuhMt1FEYobsjbrS1b3P7HDw7E4QjvWy~1EE8POkMxsKT4XWmCSjQ2L9TsEH8Dv1Y0pT60SgG-nz2Dxma9bWrGf3EDBKS-v-iNuB-tTSNHXlC7Pj8vCvwl8rRhvkLFwFB4w~de86hRm-tKMmGy1lvG48T2Kd416jFscDxEjxajjhyFrurO7LGDXg9QJgzKQFsRwP~6ls-wf~tbVnx~Iw0P0zbRh8kWr5wuFHtC0ESiZaUCUTcmAJ8rzfXsjbdh89GJmw27BpZJo7JuiuhGmtBNUyt~HYA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
[71] «Vision AI: Image and visual AI tools», Google Cloud. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://cloud.google.com/vision

[72] «Azure AI Vision with OCR and AI | Microsoft Azure». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://azure.microsoft.com/en-us/products/ai-services/ai-vision
[73] «OCR Software, Data Extraction Tool - Amazon Textract - AWS», Amazon Web Services, Inc. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://aws.amazon.com/textract/
[74] «Speech-to-Text AI: speech recognition and transcription», Google Cloud. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://cloud.google.com/speech-to-text
[75] eric-urban, «Speech to text overview - Speech service - Azure AI services». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/speech-to-text
[76] «Speech To Text - Amazon Transcribe - AWS», Amazon Web Services, Inc. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://aws.amazon.com/transcribe/
[77] «IBM Watson Speech to Text». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://www.ibm.com/products/speech-to-text
[78] «VOSK the Offline Speech Recognition», DEV Community. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://dev.to/mattsu014/vosk-offline-speech-recognition-3kbb
[79] *mozilla/DeepSpeech*. (31 maggio 2025). C++. Mozilla. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://github.com/mozilla/DeepSpeech
[80] «Natural Language API Basics», Google Cloud. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://cloud.google.com/natural-language/docs/basics#:~:text=The%20Natural%20Language%20API%20provides,analysis%2C%20use%20the%20analyzeSyntax%20method.
[81] «Tutorial: Text Analytics with Azure AI services», Microsoft Azure. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://learn.microsoft.com/en-us/azure/synapse-analytics/machine-learning/tutorial-text-analytics-use-mmlspark
[82] «Introducing our Search API», Tavily. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://www.tavily.com/
[83] «Home», Perplexity. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://docs.perplexity.ai/home
[84] «Credits & Pricing», Tavily Docs. Consultato: 10 luglio 2025. [Online]. Disponibile su: https://docs.tavily.com/documentation/api-credits
[85] «Pricing», Perplexity. Consultato: 10 luglio 2025. [Online]. Disponibile su: https://docs.perplexity.ai/guides/pricing
[86] E. Labs, «Exa», Exa. Consultato: 10 luglio 2025. [Online]. Disponibile su: https://exa.ai
[87] I. J. Goodfellow *et al.*, «Generative Adversarial Nets», in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2014. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://proceedings.neurips.cc/paper_files/paper/2014/hash/f033ed80deb0234979a61f95710dbe25-Abstract.html
[88] D. P. Kingma e M. Welling, «Auto-Encoding Variational Bayes», 10 dicembre 2022, *arXiv*: arXiv:1312.6114. doi: 10.48550/arXiv.1312.6114.
[89] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, e S. Ganguli, «Deep Unsupervised Learning using Nonequilibrium Thermodynamics», in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, giu. 2015, pp. 2256–2265. Consultato: 31 maggio 2025. [Online]. Disponibile su: https://proceedings.mlr.press/v37/sohl-dickstein15.html
[90] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, e M. Chen, «Hierarchical Text-Conditional Image Generation with CLIP Latents», 13 aprile 2022, *arXiv*: arXiv:2204.06125. doi: 10.48550/arXiv.2204.06125.
[91] «DALL·E 3». Consultato: 31 maggio 2025. [Online]. Disponibile su: https://openai.com/index/dall-e-3/
[92] M. Junczys-Dowmunt *et al.*, «Marian: Fast Neural Machine Translation in C++», 4 aprile 2018, *arXiv*: arXiv:1804.00344. doi: 10.48550/arXiv.1804.00344.
[93] Y. Liu *et al.*, «Multilingual Denoising Pre-training for Neural Machine Translation», *Trans. Assoc. Comput. Linguist.*, vol. 8, pp. 726–742, nov. 2020, doi: 10.1162/tacl_a_00343.

[94] OpenAI *et al.*, «GPT-4 Technical Report», 4 marzo 2024, *arXiv*: arXiv:2303.08774. doi: 10.48550/arXiv.2303.08774.