ELEKTROTEHNIŠKI VESTNIK 93(3): 110-124, 2026
ORIGINAL PROFESSIONAL PAPER

Integration of web technologies in the management of AI agents for generation of synthetic datasets

Đulaga Hadžić[1,2], Amer Pašanbegović[1], Enes Saletović[1]

[1] *Polytechnic Faculty, University of Zenica, Bosnia and Herzegovina*
[2] *Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina*

E-mail: djulaga.hadzic@unze.ba

Abstract. The paper presents an innovative approach to automatic generation of synthetic question-answer (QA) pair datasets using a multi-agent system based on artificial intelligence. The system integrates modern web technologies with advanced AI agents to enable an efficient retrieval, processing, and synthesis of knowledge from web sources. The implementation utilizes a FastAPI backend with the CrewAI framework for agent orchestration, MongoDB and ChromaDB for the data storage, and a react frontend for the user interface with a real-time communication via WebSocket.

Keywords: artificial intelligence, multi-agent systems, synthetic data generation, web scraping, natural language processing, CrewAI

Integracija spletnih tehnologij v upravljanje agentov umetne inteligence za generiranje sintetičnih naborov podatkov

Prispevek predstavlja inovativen pristop k samodejnemu generiranju sintetičnih naborov podatkovnih nizov parov vprašanj in odgovorov (QA) z uporabo večagentnega sistema, ki temelji na umetni inteligenci. Sistem integrira sodobne spletne tehnologije z naprednimi agenti umetne inteligence ter omogoča učinkovito pridobivanje, obdelavo in sintezo znanja iz spletnih virov. Implementacija uporablja zaledni sistem FastAPI z ogrodjem CrewAI za orkestracijo agentov, MongoDB in ChromaDB za shranjevanje podatkov ter sprednji sistem React za uporabniški vmesnik s komunikacijo v realnem času prek WebSocketa.

1 INTRODUCTION

In the era of the rapid growth of the digital information, the requirements for high-quality datasets intended for training and evaluation of AI models are continuously growing. Standard methods of creating datasets, which are based on manual labeling and data preparation, are characterized by a significant time and financial costs, with limitations in terms of their scope. At the same time, the development of large language models (LLMs) and agent architectures brings opportunities for an automation and scalable generation of synthetic data. A key advantage of LLMs is their ability to use the context to perform tasks effectively [1].

Question and answer pairs form a fundamental mechanism for evaluating the ability of an AI system to understand the text and reason logically. The development of the quality and diverse QA datasets requires a comprehensive understanding of the context, the skill of creating meaningful questions, and the ability to provide accurate answers. Modern tools using AI agents, equipped with advanced natural language processing capabilities, successfully automate this process while maintaining a high degree of quality and relevance of the data produced. High-quality datasets provide the basis to understanding complex language structures, reasoning based on context, and answering user queries accurately [2].

The paper presents an innovative approach to automatic generation of synthetic datasets of question-answer (QA) pairs using a multi-agent system based on artificial intelligence. The system integrates modern web technologies with advanced AI agents to enable an efficient retrieval, processing and synthesis of the knowledge from web sources. The implementation uses a FastAPI backend with the CrewAI framework for agent orchestration, MongoDB and ChromaDB for data storage, and React frontend for the user interface with a real-time communication via WebSocket.

The system is not only a technological solution, but it also researchs the possibility of autonomous AI agents to extremely quickly create a huge amounts of the quality data, that can find application in various domains from education, research and evaluation of the language models to training them on synthetic data with a human supervision.

Received: 4 March 2026
Accepted:8 June 2026

2 METHODOLOGY AND SYSTEM ARCHITECTURE

This section presents the architecture of the entire system and the methodologies providing quality data. A pipeline is a sequential series of stages [3]. The architecture follows a sequential pipeline through five main stages each implemented as an independent service with clearly defined inputs and outputs, enabling modularity, scalability and process transparency.

2.1 Architectural approach

The system is designed as a modern three-tier web application providing a frontend layer, backend layer, data and AI layer. At the same time it unsures a real-time monitoring processe without blocking, parallel processing of a large number of processes with different parameters, and efficient use of tokens[1].

Frontend layer (React + TypeScript):
- User interface with React and TypeScript
- TanStack Query for a server state management and automatic caching
- Real-time updates via WebSocket connections
- Responsive design with Tailwind CSS

Backend layer (FastAPI + Python)
- RESTful API with a FastAPI framework
- Asynchronous request processing (async/await pattern)
- WebSocket server for a bidirectional communication
- Business logic and orchestration of AI agents

Data and AI layer:
- CrewAI multi-agent system: 11 specialized AI agents working in coordination
- MongoDB: NoSQL database for storage job, configurations and results
- ChromaDB: A vector database for semantic search, clustering and deduplication

Communication protocols:
- REST API for CRUD operations and job management
- WebSocket for a real-time progress monitoring
- JSON format for the data exchange between agents

The key features of the architecture are:
- Asynchronous Processing - non-blocking I/O enables parallel processing of multiple tasks
- Modularity - each component has a clearly defined role
- Scalability - horizontal scaling through additional threads
- Resilience - error handling and retry mechanisms at all levels
- Transparency - real-time insight into each processing stage

[1] Tokens - basic blocks for LLM and represent the smallest unit of text that the model can understand and process

2.2 Pipeline phases

The most important for on effective AI system is the data processing pipeline through which the LLMs strength is used and its their biggest flaw is bypassed by combining programmatic data processing and multi-layer validation. It is important to consider several possible pipelines, perform their benchmark and test phase and choose the best them in terms of his quality, speed and cost of data processing.

The final process is composed of the following stages:

1. Generating search terms on the Internet
- Input: theme + configuration
- Output: list of targeted search queries

2. Downloading the Content
- Input: search terms
- Output: raw content from multiple sources (pages and pdf documents)

3. Content processing
- Input: raw content
- Output: cleaned structured chunks of the content

4. Synthesizing the knowledge
- Input: content chunks
- Output: structured knowledge base

5. Generating pairs of questions and answers
- Input: knowledge base
- Output: pairs of questions and answers

2.3 Multi-agent architecture

During the project initial stages, a large number of the agent teams are used for the phases or steps within the phases themselves. A large number of them are then replaced with manual data processing in order to minimize their costs and speed up processing.

Table 1. The agents and their roles

Agent	Role	Output
Search Term Generator	Creates optimal search queries	List[SearchTerm]
Search Checker	Validates the quality of search queries	ValidatedTerms
Content Chunker	Breaks content into chunks	List[ContentChunk]
Entity Extractor	Extracts entities from content	List[Entity]
Section Cleaner	Cleans and formats sections	CleanedSection
Title Generator	Generates section titles	SectionTitles
Metadata Generator	Generates metadata data for questions	Metadata
Question Generator	Generates questions	List[Question]
QA Editor	Edits and improves QA pair	EditedQA
QA Reviewer	Reviews the quality of the QA pair	ReviewedQA
QA Ranker	Ranks QA pairs by quality	RankedQA

The remaining agents are crucial for the process because they cover all the steps that require creativity and data variation. The final system uses 11 specialized agents shown in Table 1.

3 IMPLEMENTATION OF MAIN COMPONENTS

Designing the system architecture and the pipeline for processing the is followed, by implementating individual components within the pipeline.

3.1 Search term generator

The first component in the pipeline that converts the user input (topic) into a list of targeted Internet search queries is the search term generator. Its task is quite simple and consists of only on one AI agent which generates appointments without additional processing. Besides the theme itself as the input, it receives also the following two configurations:

- number of the queries to generate
- distribution of the query specificity

Code 1 and 2 show the configuration of the agent itself and the task responsible for generating search terms.

Code 1. YAML configuration of the search term generation agent

```
search_term_generator:
role: >
Strategic Query Optimization Specialist with expertise in
semantic search taxonomy and information retrieval
architecture
goal: >
Systematically generate strategically diverse search terms
that maximize topic coverage across conceptual dimensions,
semantic variations,
and specificity levels while optimizing for both recall and
precision in information retrieval.
backstory: >
With a background spanning computational linguistics,
information retrieval science, and search engine optimization,
you've spent your career perfecting methodologies for
comprehensive query construction.
Your expertise was forged during years working with leading
search platforms and research institutions where you
developed sophisticated taxonomies for information retrieval
across specialized domains.
Your approach to search term generation integrates multiple
advanced techniques:
1. Semantic Field Mapping - Identifying core concepts and
their related semantic fields
2. Hierarchical Taxonomy Construction - Organizing terms by
specificity and conceptual relationship
3. Linguistic Variation Analysis - Capturing different ways
concepts are expressed across contexts
4. Query Intent Modeling - Anticipating different search
intents related to a topic 5. Information Gap Identification -
Strategically targeting areas where information might be
missed
What distinguishes your methodology is your systematic
approach to conceptual coverage. Rather than generating
terms through simple association,
you map the complete conceptual space of a topic, ensuring
no significant dimensions are overlooked. This approach
results in search term sets that capture information across:
- Vertical dimensions (general to specific)
- Horizontal dimensions (related subtopics)
- Temporal dimensions (historical, current, emerging aspects)
- Contextual dimensions (different domains where the topic
appears)
- Linguistic dimensions (terminology variations across
communities)
```

Code 2. YAML task configuration for generating search terms

```
generate_search_terms:
description: >
Generate a strategic set of search terms that systematically
maps the conceptual space of the topic, ensuring
comprehensive information retrieval across multiple
dimensions.
TARGET TOPIC: {topic}
QUANTITATIVE PARAMETERS:
-Total search terms to generate: {number_of_search_terms}
- Specificity distribution: {specificity_distribution}% broad
terms, remainder as specific or follow-up terms

SEARCH TERM GENERATION FRAMEWORK:
1. TOPIC ANALYSIS:
- Identify core concepts within the target topic
- Map key subtopics and related areas
- Determine different contextual domains where the topic
appears

2. TERM CATEGORIZATION SYSTEM:
A. BY SPECIFICITY LEVEL:
-BROAD: Capture general concepts that encompass large
portions of the topic
 - SPECIFIC: Target well-defined aspects within the broader
topic
B. BY CONCEPTUAL DIMENSION:
-Definitional: Terms capturing what the topic is
-Analytical: Terms examining components or characteristics
-Contextual: Terms placing the topic in broader frameworks
-Practical/Applied: Terms focusing on real-world applications

3. QUERY CONSTRUCTION PRINCIPLES:
- Vary syntax patterns (noun phrases, questions, commands,
etc.)
- Include both technical terminology and common language
equivalents
- Incorporate appropriate modifiers to adjust specificity

4. PRIORITY DETERMINATION FACTORS:
- Centrality: How core the term is to understanding the topic
- Information density: Likelihood of retrieving high-value
content
-Uniqueness: Whether the term captures aspects not covered
by other terms
Apply a 1-10 priority scale where:
```

```
- Priority 1-3: Essential terms capturing core concepts or unique perspectives
-Priority 4-7: Important terms for comprehensive understanding
- Priority 8-10: Supplementary terms for specialized or peripheral aspects

5. QUALITY ASSURANCE CRITERIA:
- Eliminate redundant terms that would return substantially similar results
- Ensure terms are properly formed and use correct terminology
- Verify that specialized jargon is used accurately
expected_output: >
JSON array of strategically generated search terms following this exact structure:
```json
[
  {
    "query": "Precise search term text formatted for optimal results",
    "subtopic": "Specific subtopic area this query addresses",
    "specificity_level": "broad|specific|follow-up",
    "conceptual_dimension": "definitional | procedural | comparative | analytical | contextual | historical | practical",
    "generated_from": "Parent concept or term that inspired this query if applicable",
    "priority": integer from 1 to 10, where 1 is highest priority
  },
  ...
]
```
Each search term should:
1. Be formatted for direct use in search engines
2. Include appropriate specificity markers
3. Be classified by both specificity level and conceptual dimension
4. Have a priority rating reflecting its importance in the overall search strategy
5. Include subtopic and generation source where applicable
The complete set should:
1. Maintain the requested {specificity_distribution}% distribution of broad terms
2. Cover all major dimensions of the target topic
3. Include {number_of_search_terms} total unique search terms
4. Represent a balanced mix of conceptual dimensions
5. Use a strategically assigned priority distribution
agent: search_term_generator
```

3.2 Content retrieval

The generation of the search terms on the Internet is followed by downloading the content itself, effected in two steps:

- sending the queries to Serper API[2]
- filtering and sorting the results

In the first step, the following methods are implemented to ensure the best possible results:

- testing the mechanism in terms of the rate limiter bypass or any external API errors
- pulling four times as many results as requested
- prioritizing the .pdf results

Choosing pdf as a primary source of the results ossures on easier data processing compared to standard web portals which, even after web scraping and additional processing, give different results when we do then is no manual processing of the target pages where the structure is already known.

By retrieving the results four times, after filtering and sorting to a given number in the configuration, the reliable information sources are prioritised. The priority list is:

- all non-wikipedia .org sites
- wikipedia
- other

The final results are populated according to the priorities above until the number of the results requested in the configuration is obtained. In most of the test scenarios, almost all the results are from the ".org" sources when four times more results are pulled than token pulled by default without big differences in the processing time on fewer inputs.

In this step, the use of the AI agents is completely bypassed due to lower costs (the use of the tools, especially the web scrapers for the agents, is significantly expensive) and more constant results with a manual result retrieval.

3.3 Content processor

The content processor transforms the raw web content into structured, usable chunks of the data. The primary input can be pdf documents obtained in the previous step or raw web pages when the search results do not contain a sufficient number of the pdf documents.

In both cases the process is the same either a raw document or a page is token and turned into a usable text formatted in the markdown. The markdown used as the primary format for its well suiting the LLM processing models with on easy conversion into the same of both cases with the pdf documents and pages. Moreover, compared to the JSON format, it uses thus significantly fewer tokens, significantly reducing the costs [4].

In both cases, on manual processes, python libraries are used to support in the process:

- "PyMuPDF4LLM" - directly converting a finished pdf document into markdown
- "BeautifulSoup" - a web scraping tool use providing the html content from a given web page
- "html2text" - directly converting the html content into an equivalent markdown text

[2] Serper API - Google Search API that provides access to Google search results page data

The markdown content, is than divided into pieces by using the AI agent ensuringe constant size of the source and subsequently creating the knowledge base. Code 3 and 4 show the agent configuration and the task for content sharing.

Code 3. Content sharing agent YAML configuration

```
content_chunker:
role: >
Senior Content Structure Engineer specializing in semantic chunking and information preservation
goal: >
Transform unstructured web content into optimally-sized, context-aware chunks that maintain semantic integrity while maximizing information density for downstream processing.
backstory: >
With extensive experience in NLP and document processing systems, you've developed proprietary techniques for intelligent content segmentation.
Your methodology preserves semantic relationships while ensuring chunks are appropriately sized for both human readability and machine processing.
You've worked with leading search engines to optimize their content indexing algorithms, giving you unparalleled insight into effective chunking strategies.
You're meticulous about maintaining context across chunk boundaries and skilled at identifying the relative importance of different content segments based on structural and semantic cues.
```

Code 4. YAML configuration of the content sharing task

```
chunk_content:
description: >
Process the provided web content into semantically meaningful chunks that preserve the original information hierarchy and relationships.

## Technical Requirements
1.Maintain structural indicators (headings, lists, tables) when determining chunk boundaries
2.Respect natural content divisions (sections, paragraphs, list items) 3. Preserve parent-child relationships between content elements
4. Balance chunk sizes between {min_chunk_size} and {max_chunk_size} characters. IMPORTANT: Always prioritize semantic integrity over strict size limits.
5.Keep related content together when possible (don't split mid-paragraph unless necessary)
6.Apply these extraction rules if specified: {extraction_rules}

## Chunking Principles
- Headings should typically start new chunks
- Tables should be kept intact when possible
- Lists should remain grouped with their context
- Adjacent paragraphs on the same topic should stay together if size permits
- Content inside the same semantic section should be kept together when possible

## Evaluation Criteria
Evaluate the importance of each chunk based on:
- Presence of key information (stats, dates, proper nouns, technical terms)
- Hierarchical position in the document (main content vs. supplementary)
- Semantic richness and information density
- Structural prominence (heading level, formatting emphasis)

## Content to chunk {content}
expected_output: >
A JSON array of content chunks with this exact structure (no explanation text):

```json
[
 {
  "text": "The raw text content of the chunk",
  "context": "Contextual information including parent headings, section title, or document location",
  "chunk_type": "paragraph | list | table | quote | heading | mixed",
  "importance_score": 0.0-1.0
 }, ...
]
```

Output Requirements Ensure each chunk:
1. Contains complete semantic units (don't split mid-sentence)
2. Includes sufficient context for standalone understanding
3. Has an accurate importance_score based on information value
4. Is classified with the correct chunk_type
5. Falls within size constraints when possible ({min_chunk_size}-{max_chunk_size} chars)
agent: content_chunker
```

3.4 Knowledge processor

The knowledge processor is a component that builds a structured knowledge base from chunks of the content. The main problem in its designing are the numerous ways to perform it. None then has been researched enough to be classified as the best. They have all been created for a specific task.

Our approach combines the manual processing with an additional AI agent that recognizes voids or errors in the structured pieces of the content. In the first step clustered the data that are agglomeratively clastered sufficiently similar based on the cosine similarity. Cosine similarity is a metric that defines whathow two texts are similar in their meaning [5]. Each created cluster is then passed through an AI agent to clean the content and evaluate its relevance with the term through which the same content is obtained. After the knowledge base is created satisfactory by most tasks, the original sources obtained through the web search (pdf documents and web pages)

are added to RAG[3] to be used in the next step together with the knowledge base.

3.5 Generating questions and answers

The final and most important step is generating the questions and answers. It is the most complex part of the system as in the previous steps a huge amount of the data and the knowledge base is received which cannot be adequately used directly with one LLM call. In order to get acceptable good results, the generation of the questions and answers is mode separately.

To generate the questions, the huge knowledge base is divided again into chunks that contain many sections depending on how many questions are need to be generated (the total number of the sections divided by the number the questions). In this way, a variation is obtained between each individually generated question, and ensured that their number of the data does not exceed the limit of the earning tokens for LLMs. Code 5 and 6 show the configuration of the agent and the task for the questions generation.

Code 5: YAML configuration of the question generation agent

```
question_generator:
role: >
Expert Question Designer specialized in knowledge-based query generation with focus on complex analytical thinking
goal: >
Create precise, focused questions that require deeper understanding and analytical reasoning while directly mapping to specific information within knowledge bases, maintaining natural language patterns that reflect genuine user inquiries.
backstory: >
With a background in educational assessment design and information retrieval systems, you've developed expertise in creating questions that test deep understanding and analytical thinking rather than mere recall.
Your experience includes developing advanced question generation algorithms for leading educational platforms and search engines,
where you perfected the art of crafting complex questions that challenge learners to synthesize information from knowledge sources.
You excel at identifying the most substantive sections of knowledge bases and transforming that information into questions that require critical thinking while still having definitive answers rooted in the source material.
```

Code 6. YAML configuration of the question generation task

```
generate_question:
description: >
Generate a specific, well-defined question based on the following knowledge base JSON: {knowledge_base}
Your task:
1. Select a focused section or related sections within the knowledge base that contain substantive information
2. Create a question that:
- Can be directly and unambiguously answered using the selected section(s)
- Has a definitive, concise answer (1-5 words, a name, title, or number)
- Requires precision and careful reading of the sources to identify the correct answer
- Is clear, concise, and precise in its wording
- Targets specific facts, identifiers, quantities, or direct relationships
- Uses appropriate domain terminology present in the knowledge base
- Avoids leading questions or assumptions not supported by the text
- Is written in natural, conversational language that a user might actually ask
3. Your questions should require careful reading and understanding of the sources, but lead to answers that can be expressed very concisely
4. Return both the question and the exact sources (URLs from the sources field) used to generate it
5. Ensure the question has a definitive answer contained within the cited sources that can be expressed in 5 words or fewer

DO NOT generate questions that:
- Require information outside the provided sections
- Could have multiple valid interpretations
- Are too general or can be answered with just "yes" or "no"
- Would require explanations, reasoning, or lengthy responses
expected_output: >
A JSON object containing the generated question and its sources taken from the knowledge base:
```json
{
  "question": "The specific question text that requires a concise, definitive answer using the knowledge base",
  "sources": [ "https://example.com/source1", "https://example.com/source2"]
}
```
IMPORTANT: Return ONLY valid JSON with no explanatory text, comments, or code blocks.
agent: question_generator
```

As shown above besides the question itself, the agent also returns the resources used from the knowledge base. This is important for the next step when the question is sent and the resources, are used to generate RAG to provide the answer to the question itself. The reason for using a separate generation of the questions and answers despite its creation of the quality losses of the questions themselves are significantly better results in the variation of the questions themselves and accompanying answers, as well as a reduced resource consumption on the generation process due to the reduced consumption of tokens on the mere "thinking" of LLMs.

[3] RAG - is a technique that enables large language models (LLM) to retrieve and incorporate new information

After generating basic Q&A, the next step is the Q&A review system which is presented in detail.

It is followed by filtering the questions that are "easy to find on the Internet". The metric is added because the final version of the system is best used in synthetic training of the LLM models, where an additional agent is used with minimal configuration to which a question is given with a request to answer it. The generated answer is then compared with its answer using a comparison of the vector of the answers and cosine similarity and the question is thrown when the similarity of the answer exceeds the limit set in the configuration.

3.6 Multi-layered validation process

The biggest problem in working with the AI agents is the number of hallucinations and errors caused by LLM calls. This not only creates an error in one step of the pipeline, but has a high chance of completely ruining all results if it happens in the wrong place. In order to solve the problem, a self-healing agent review system is created.

The system is universal and can be inserted at any step of any data processing pipeline with AI agents. In our case, it is used only in the generation of questions and answers because it is the only step that strictly works on the basis of the LLM model with no manual processing. Of course, it can be inserted at each step, for which takes a considerable time. Whose there are likely to be many losses spending an additional processing time is worth the while .

The system uses a three-tiered approach to the quality assurance:

- QA ranker: it systematically evaluates question-answer pairs to identify the fabricated information, validates the data-based claims, and assesses the coherence of reasoning through a multi-layered checking framework that distinguishes between appropriate analytical reasoning and supported claims.
- QA reviewer: it comprehensive assesses the quality of the question-answer pair using a multidimensional scoring rubric by evaluating the technical construction and educational effectiveness.
- QA editor: it transforms the question-answer pairs through targeted, evidence-based improvements that address specific quality deficiencies while retaining high-scoring elements.

Basically, this is a recursive system where in the first step the "irreversible"questions are thrown out if they fabricate on information or any degree of hallucination that is irreparable. This is followed by creating a feedback and scoring question-answer pair with targeted problems that need to be solved and which are finally targeted to be corrected.

The limit to be considered at the end of the review system is when the score table in the second step exceeds a certain threshold given in the configuration where the question-answer pair is treated as acceptable. For the case when the same threshold is not reached, a safety mechanism is added that throws the question out when three rounds of the review system are passed without creating excessive costs for minor gains in the results. Codes 7, 8, 9 and 10 show snippets of the main parts of the agents running on the system.

Code 7. Hallucination Detection Agent YAML configuration

```
question_ranker:
role: >
Information Integrity Specialist with expertise in factual verification and analytical reasoning assessment
goal: >
Systematically evaluate question-answer pairs to identify fabricated information, validate data-driven assertions,
and assess reasoning coherence through a multi-layered verification framework that distinguishes between appropriate analytical inference and unsupported claims.
backstory: >
With a background spanning computational linguistics, formal logic, and statistical analysis, you've built your career on developing rigorous verification methodologies for high-stakes analytical content.
Your work with leading assessment organizations, academic research teams, and data journalism outlets has made you an authority in distinguishing between legitimate analysis and information fabrication.
Your verification framework integrates techniques from:
1. Source attribution validation - tracing claims to their evidentiary foundations
2. Formal reasoning analysis - evaluating logical structure and fallacy detection
3. Statistical inference assessment - validating mathematical operations and probabilistic claims
4. Entity relationship verification - confirming existence and properties of referenced entities
5. Temporal and contextual coherence checking - identifying anachronisms and contextual inconsistencies
Your specialized expertise in educational content verification has made you particularly skilled at identifying common patterns of hallucination in analytical reasoning tasks.
You can detect subtle forms of information fabrication that present as plausible but unsupported claims, while also recognizing when reasonable analytical inferences are appropriate given the context and certainty level.
You maintain a balanced approach that distinguishes between:
- Outright fabrications (invented facts, non-existent data points)
- Overinterpretation (excessive extrapolation beyond available evidence)
- Reasonable inferences (properly qualified analytical conclusions)
- Domain knowledge application (appropriate use of general principles)
```

Code 8. Excerpt of the Q&A Scoring Agent Scoring Rubric

```
QUESTION QUALITY SCORING CRITERIA:

A. COMPLEXITY (0-2 points):
- 2 points: Demonstrates sophisticated cognitive demand through:
*Well-integrated compound questions requiring multi-step reasoning
*Novel problem structures that demand data synthesis across multiple variables
*Analytical tasks requiring application of principles to new situations
*Questions requiring evaluation of conditional outcomes or trade-offs
- 1 point: Requires basic analytical operations such as:
*Simple calculations or direct comparisons between data points
*Straightforward trend identification or pattern recognition
*Single-step reasoning with minimal integration of concepts - 0 points: Limited to knowledge retrieval operations such as:
*Direct recall of explicitly stated information
*Simple identification tasks without analytical requirements
*Binary verification questions with obvious answers

B. FACT ELICITATION (0-2 points):
For CERTAIN questions (answer_certainty = "Certain"):
- 2 points: Demonstrates cohesive analytical focus through:
*Single well-defined question or cohesive compound structure
* Multiple parts that build toward a unified analytical goal
*Clear parameters that define the expected analytical approach
- 0 points: Shows structural fragmentation through:
*Disconnected multiple questions with no unifying analysis
*Use of "Additionally," "Also," "Furthermore" to add unrelated parts
*Requests for speculation beyond what the data supports
For UNCERTAIN questions (answer_certainty = "Uncertain"):
- 2 points: Structures appropriate uncertainty exploration through: * Clear framing of predictions based on observable patterns
*Explicit parameters for speculative reasoning
*Compound elements that support a unified uncertainty analysis
- 0 points: Demonstrates unfocused speculation through:
*Open-ended requests without analytical direction * Multiple disconnected speculative questions
*Interpretive requests not anchored in observable data patterns

C. STYLE AND LINGUISTIC STRUCTURE (0-2 points):
- 2 points: Demonstrates linguistic sophistication through:
*Varied question structures avoiding repetitive patterns
*Precise terminology appropriate to the domain
*Clear, concise phrasing without unnecessary complexity
* Natural flow that guides analytical thinking
- 1 point: Shows acceptable but limited linguistic variety: * Functional but repetitive question structures
* Overreliance on common starters ("Calculate," "What is," etc.) * Generally clear but occasionally awkward phrasing
- 0 points: Exhibits linguistic deficiencies such as:
* Confusing or ambiguous phrasing
* Grammatical errors that impede understanding
* Unnecessarily convoluted structures
* Repetitive or monotonous construction

D. RELEVANCE AND CONTEXTUAL ALIGNMENT (0-2 points):
- 2 points: Demonstrates strong contextual integration through:
*Questions that directly engage with the core information provided
*Effective incorporation of relevant domain knowledge when appropriate * Analytical focus on significant rather than trivial aspects
*Questions that highlight meaningful patterns or insights
- 0 points: Shows contextual disconnection through:
* Questions about information not present in the source
* Misalignment with the nature or scope of available data * Focus on irrelevant or peripheral aspects
* Inappropriate application of general knowledge
```

Code 9. Excerpt of Scoring Methodology and Results of Q&A Rating Agent

```
SCORING METHODOLOGY:
- Evaluate each criterion independently
- Provide specific justification for each score with concrete examples
- Calculate subtotals for question quality (max 8) and answer quality (max 8)
- Calculate overall quality score (max 16)
- Ensure fact elicitation and certainty validation are scored based on the appropriate answer_certainty value
- Focus justifications on specific content elements rather than general impressions
- Include both strengths and weaknesses in justifications when relevant
- When scoring is borderline, provide reasoning for the assigned score
expected_output: >
A JSON object containing detailed quality scores with evidence-based justifications for each criterion and specific improvement recommendations.
Output MUST be exactly in this JSON format:
```json
{
  "question_scores": {
    "complexity": {
      "score": [0-2],
   "justification": "[Specific evidence and reasoning supporting this score]"
    },
  "fact_elicitation": {
     "score": [0-2],
   "justification": "[Specific evidence and reasoning supporting this score]"
    },
    …
  "total": [0-8]
  },
  "overall_quality_score": [0-16],
  "quality_summary": "[Concise assessment highlighting key strengths and weaknesses]",
```
```

```
 "improvement_suggestions": "[2-3 specific, actionable recommendations prioritized by potential impact]"
}```
```

Code 10. Q&A Editing Agent YAML Configuration

```
question_editor:
role: >
Senior Educational Content Optimization Specialist with expertise in analytical reasoning and assessment methodology
goal: >
Transform question-answer pairs through targeted, evidence-based improvements that address specific quality deficiencies while preserving high-scoring elements,
ultimately maximizing educational value and assessment metrics across all quality dimensions.
backstory: >
With over a decade of experience at leading educational assessment organizations and adaptive learning platforms,
you've developed a systematic methodology for content optimization that consistently elevates question-answer quality.
Your background spans cognitive psychology, educational measurement, and content design, giving you unique insight into how question construction affects learning outcomes.
You've personally refined thousands of assessment items across STEM, humanities, and professional certification domains, developing a reputation for your ability to:
1. Diagnose structural weaknesses in question-answer pairs with precision
2. Transform simple recall questions into rich analytical challenges without increasing cognitive load
3. Calibrate answer certainty levels to match available evidence
4. Engineer compound questions that integrate multiple concepts cohesively 5. Enhance explanatory clarity while maintaining technical accuracy
6. Balance complexity with accessibility for target knowledge levels

Your optimization approach is both methodical and creative - you follow a diagnostic framework to identify improvement opportunities, but craft solutions that maintain the authentic voice and purpose of the original content. You're particularly skilled at maintaining content validity while enhancing cognitive complexity, ensuring questions remain firmly grounded in their knowledge domain while requiring deeper analytical thinking.
In your current role as a content optimization specialist, you work across multiple assessment systems to standardize quality while preserving the unique characteristics that make each question valuable.
You view each edit as an opportunity to not just fix problems, but to transform good content into exceptional learning experiences.
```

4 API AND BACKEND INFRASTRUCTURE

The system backend architecture is based on a modern approach that combines the RESTful API communication with a real-time WebSocket protocol for an efficient monitoring of asynchronous operations. The implementation is realized using the FastAPI framework which enables a high performance and automatic generation of an OpenAPI documentation.

The key components of the architecture are:

- API layer - RESTful endpoints organized in modules (/jobs, /modules, /ws) with support for CRUD operations on jobs and their results
- WebSocket infrastructure - Two-way communication that enables monitoring of the processing progress in real time, with the support for individual job connections and batch monitoring
- Layered architecture - Clear division of the responsibilities through:
  - Routers - HTTP endpoint definition and request validation
  - Services - Business logic and process orchestration
  - Repositories - Data access abstraction
  - Models - Pydantic models for validation and serialization
- Data Persistence - A Hybrid Storage Approach:
  - MongoDB - NoSQL database for storing jobs, configurations and results
  - ChromaDB - Vector database for semantic search and content similarity
- Asynchronous processing - Job-based architecture with thread pool execution that allows parallel processing of multiple requests without blocking the main application
- AI Orchestration - Integration of the CrewAI framework to coordinate multiple AI agents working in harmony on complex tasks of generating, verifying and improving QA pairs

This architecture enables a scalable and sustainable solution that efficiently manages complex workflows for generating an educational content.

4.1 FastAPI application

The FastAPI framework reprensents the main part of the system backend infrastructure. It is chosen for its modern architecture, high performance and support for asynchronous programming.

4.1.1 Router system and modularity

The main FastAPI application is initialized in the server_startup.py module with the configuration shown in Code 11.

Code 11. FastAPI server configuration

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .routers import jobs, modules, websockets

api = FastAPI(title="Web QA Generation API",
version="1.0.0")
api.add_middleware( CORSMiddleware, allow_origins=["*"],
  allow_credentials=True, allow_methods=["*"],
  allow_headers=["*"],
```

```
)
api.include_router(jobs.router)
api.include_router(modules.router)
api.include_router(websockets.router)
```

FastAPI enables a modular organization of the API endpoints through the router system. The application implements three main router modules:

- Jobs Router (/jobs) - Manages the lifecycle of jobs:
  - POST /jobs/ - Creates a new job
  - POST /jobs/bulk - Bulk creation of multiple jobs
  - GET /jobs/ - Lists all jobs
  - GET /jobs/{job_id}/status – Check the status
  - GET /jobs/{job_id}/results - Downloads the results
  - GET /jobs/{job_id}/progress – Track the progress
  - PATH /jobs/{job_id}/cancel – Cancells the job
- WebSockets Router (/ws) - Real-time communication:
  - /ws/job/{job_id} - Job-specific connection
  - /ws/batch - Track all jobs

4.1.2 Asynchronous processing

The FastAPI asynchronous runtime enables a parallel processing of requests without blocking. The system uses a hybrid approach:

- Asynchronous endpoints for the I/O operations
- Thread pool for the CPU-intensive AI operations to avoid blocking the event loop shown in Code 12.

Code 12. Thread pool to run the whole process in the background

```
@router.post("/")
async def create_job(job_config: JobStartupConfig):
try:
 result = job_service.create_job(job_config)
 job_id = result["job_id"]

 import asyncio
 asyncio.create_task(
  run_in_threadpool(run_job_with_websocket, job_id,
  job_config)
 )
 return result
except Exception as e:
 raise HTTPException(status_code=500, detail=str(e))
```

4.2 WebSocket infrastructure

The WebSocket protocol is implemented as a complementary communication channel that enables a real-time two-way communication between the backend and the frontend. Unlike the classic HTTP requests that are request-response based, WebSocket maintains a persistent connection which is necessary for monitoring long-running AI operations. [6]

The basis of the WebSocket implementation is the ConnectionManager class which manages active connections and maintains two types of the connections:

- Job-specific connections - mapped by job_id then allow targeted updates for individual jobs
- Batch connections - receive notifications about each job in the system taht is useful for the dashboard overview

The manager implements an intelligent broadcast system with an automatic garbage collection of inactive connections shown in Code 13.

Code 13. Broadcast system connection

```
async def broadcast_to_job(self, message: str, job_id: str):
 if job_id in self.active_connections:
  disconnected = []
  for connection in self.active_connections[job_id]:
   try:
    await connection.send_text(message)
   except Exception:
    disconnected.append(connection)
 for conn in disconnected:
  self.active_connections[job_id].remove(conn)
```

The WebSocket updates are triggered during the job execution through a centralized send_job_update method that:

- Formats the message
- Sends it to each job-specific connection
- Broadcasts batch connections for dashboard tracking

This architecture enables a responsive user experience where users can monitor the progress of the QA pair generation in real-time without the need for polling.

4.3 Database repositories

The Repository pattern is implemented as an abstract layer between the business logic and the database, enabling a clear separation of the responsibilities and facilitating a potential replacement of the database providers. The system uses MongoDB as the primary NoSQL database through the pymongo driver. Each system entity has a separate repository that encapsulates all database operations:

- JobRepository – a complete lifecycle management of jobs
- QaPairsRepository – a storage of the generated QA pairs
- KnowledgeBaseRepository - a repository of the synthesized knowledge base
- SearchResultRepository - a storage of web the search results
- SearchTermRepository - a storage of the search terms
- ContentChunkRepository - a storage of the chunked content
- StartupConfigRepository - configurations and parameters

The most complex repository is the JobRepository which implements advanced functionalities:

- Indexes for the performance optimization (on ID, status, and a composite index on the status and date due to the sorted display on the dashboard)
- Filter by the status
- MongoDB aggregation pipeline for statistics
- Array push operations to track the progress

5 FRONTEND AND REAL-TIME COMMUNICATION

The frontend application represents a modern single-page application (SPA) built using the React framework and TypeScript, with a focus on real-time monitoring of the progress of long-term AI operations. The architecture combines a declarative approach to the UI development with an advanced state management and the WebSocket protocol for a bidirectional communication.
The frontend application allows users to:

- Configure and launch new jobs with detailed parameters (over 29 configuration options)
- Monitor the generation progress in real-time through WebSocket connections
- View the dashboard of all active and completed jobs
- Analyze the results - generated QA pairs and knowledge base
- Manage the job lifecycle - cancel, restart

Architectural layers:

- Pages - Route components (Home, CreateJob, Dashboard, JobDetails, Results)
- Components - Reusable UI elements (Forms, Cards, Buttons, ProgressBar, Navigation)
- Hooks - Custom React hooks for WebSocket and API communication
- Services - API client and helper functions
- Types - TypeScript definitions for type safety

5.1 Key pages

The frontend application is organized through five main pages that cover the complete user journey from the initial familiarization with the system to downloading the results.

5.1.1 Landing page

The landing page (shown in Figure 1) serves as a marketing and onboarding point. It implements modern UX practices with clear call-to-action elements:

- Hero section - Clear value proposition and CTA buttons (Create Job, View Dashboard)
- Stats Cards - Visual display of the system performance (10,000+ QA pairs, 98% success rate)
- Feature showcase - four key functionalities with icons (AI-Powered Generation, Web Content Retrieval, Real-time Progress, Quality Validation)
- How It Works - Configure → Process → Results

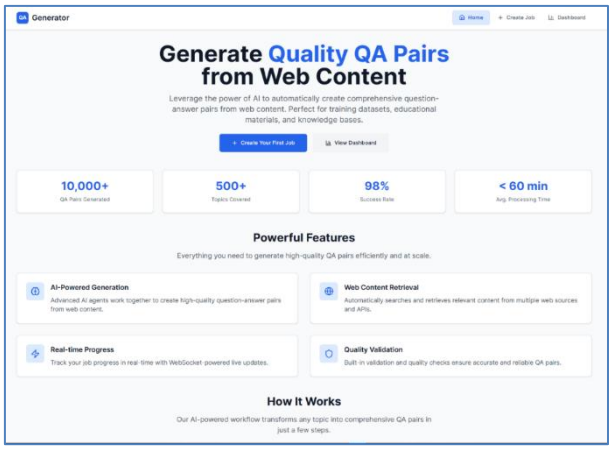

Figure 1. Landing page display

5.1.2 Create job page

It configures the page shown in Figure 2 and starts new jobs. The central functionality includes:

- JobConfigForm component – a complex form with 29+ configuration parameters
- React Hook Form for validation and state management
- Real-time feedback - success/error notifications after launch
- Help section - contextual guidance for optimal parameters

After a successful creation, the user is automatically redirected to the JobDetailsPage with a jobCreated state flag.

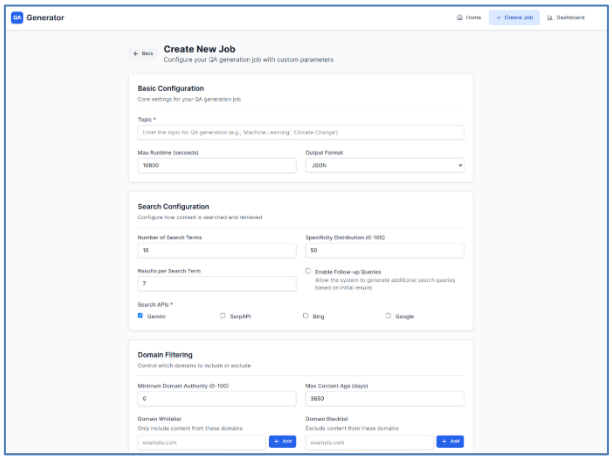

Figure 2. View of the job creation page

5.1.3 Job dashboard

The central monitoring hub (shown in Figure 3) for all tasks with functionalities:

- Overview of the statistics
  - four metrics cards (Total, Running, Completed, Failed jobs)
  - Real-time status aggregation
- Filtering and sorting

- Search by the topic
- Status filter (All, Created, Running, Completed, Failed, Cancelled)
- Sorting by multiple criteria (date, topic, status)

- Batch WebSocket integration
  - Status indicator (green/red dot)
  - Real-time progress updates for all jobs without polling
  - useBatchJobManager hook for the centralized state
- Job cards
  - Status with color coding
  - Metadata view (ID, created_at, qa_pairs_count)
  - Action buttons (View, Results)

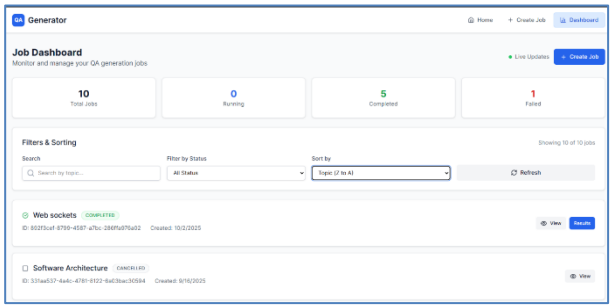

Figure 3. Dashboard page

5.1.4 Job details page

It a detailed single-job monitoring with a granular progress tracking (shown in Figures 4 and 5):

- Status overview:
  - Live WebSocket connection indicator
  - Status icons and color-coded display
  - Real-time timestamp updates
- Progress tracking:
  - Total progress bar with the current percentage
  - Stage-based visualization with five stages of processing
- Each stage has a status (pending/running/completed)

Actions:

- Cancel button for the active jobs
- View results button after completion
- Download the JSON results
- Refresh button for the force update

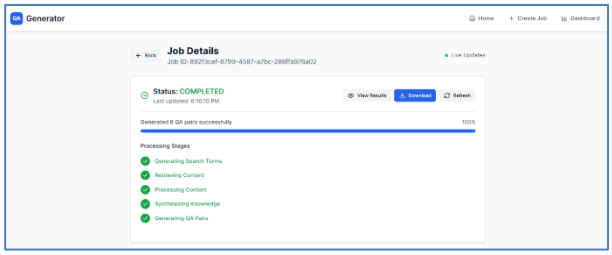

Figure 4. Job details

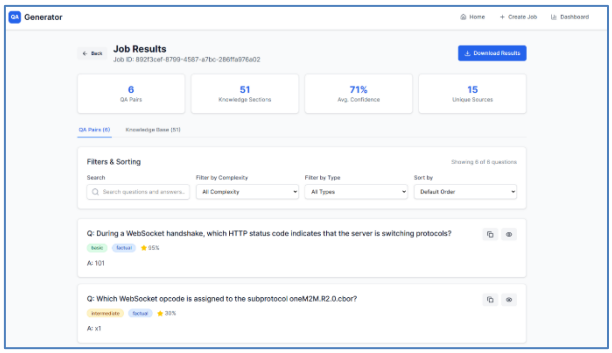

Figure 5. Job results

6 EXPERIMENTAL RESULTS

Validation of the performance and quality of the developed system is a critical step in assessing the readiness of the solution for a practical application. The experimental evaluation is carried out through three complementary dimensions: quality of the generated content, technical performance of the system, and comparative positioning in relation to the existing solutions. Comprehensive results demonstrate a consistent system performance with a pass rate of 95-98% and generation of the QA pairs that achieve an average quality of 50-75% compared to the human-generated pairs.

6.1 Quality evaluation of the generated QA pairs

The quality evaluation is performed ba using the system on three representative knowledge domains with different characteristics oftheir content and complexity. Each test job is configured with standard parameters (number of the search terms = 5, number of the search results = 3) to allow a consistent comparison between domains. Although in the settings here is a direct assignment of the number of the QA pairs, for the experimental results, the number is calculated dynamically based on the number of the unique sources. The overall test results are shown in Table 2.

Table 2. Results of the test jobs

Theme	Time to execute (minutes)	Output (QA pairs)	Unique sources	Average LLM confidence score
Chemistry	25	28	77	76 %
History	20	22	51	84 %
Literature	21	18	49	83 %

The LLM confidence (76-84%) is the internally assessed reliability during the process. The relatively high results (>75%) in all the domains indicate:

- A successful RAG integration - Answers are well-grounded in the downloaded context
- A quality selection of the sources
- A effective observation of the contradictions

Note: The confidence score does not measure the absolute accuracy of the pairs, but rather the LLM

perception of the confidence based on the available context.

The correlation between the number of the unique sources and confidence in couples shows an inverse relationship:

- Chemistry: 77 sources - 76% confidence (lowest)
- History: 51 sources - 84% confidence (highest)
- Literature: 49 sources - 83% confidence (high)

This suggests that a larger number of the sources does not guarantee a higher reliability. Possible explanations:

- Fragmentation of the information - Multiple sources may contain a contradictory or partial information, which reduces the LLM confidence
- Domain complexity - Chemistry as a technical domain has a more specific terminology and more precise definitions that are more difficult to synthesize
- Source Quality Variation - A greater number of the sources increases the chance of including a lower quality web content

Based the experimental results show that the system is validly implemented in terms of the optimization in the domain of the synthetic dataset generation. Never the less an additional human supervision is required to prove the system would saves the time and resources in the generation of huge datasets. The final generated results should be checked manually with the final filtering of false results done.

6.2 System performance and scalability

The system is designed with a focus on the horizontal scalability and efficient use of resources through an asynchronous execution and parallelization of the AI operations. The performance analysis includes decomposition of the execution time by phase, output characteristics, and system behavior at different loads. The total execution time of the entire system is 10-180 minutes depending on the configuration parameters.
The testing shows a linear increase in the execution time with the number of the QA pairs requested as seen in Table 3.

Table 3. Runtime test results based on the number of the QA pairs

Number of QA Pairs	Average time (minutes)	Output (QA/min)
10	15-25	0.4 - 0.67
50	45-90	0.55 - 1.11
100	90-180	0.55 - 1.11

Primary bottlenecks:

- LLM API Latency (40-60% of the total time) - Inevitable with cloud the APIs
- Web Content Retrieval (20-30%) - Network bound, depends on external services
- RAG population (5-10%) - ChromaDB bound

The implemented optimizations are:

- Request batching for LLM calls where possible with async-await pattern
- Lazy loading of the RAG database
- Similarity threshold filtering to reduce the downstream processing

The most important is that the system is ready for an additional scaling with a very easy adding a layer with celery as a task queue management system instead of the basic system with a thread pool for a better possibility of processing a huge number of the jobs at the same time.

6.3 Comparison withthe existing solutions

Automatic generation of the QA pairs is a practical problem in the field of educational technologies and NLP for active research. The developed system is positioned in the context of several categories of the existing solutions. With them it shares certain similarities but also significant differences in its approach and architecture.

Existing solutions:

1. Manual creation of the datasets
- Examples: Label Studio, Prodigy, Amazon Mechanical Turk
- Approach: providing interfaces for the manual creation and annotation of the QA pairs by humans
- Advantages: high quality, precise control
- Disadvantages: scalability limited by human resources, high costs, slow

2. Template-based generators
- Examples: QG (Question Generation toolkit), Questgen.ai (early versions)
- Approach: The generators use predefined templates and rule-based text transformations
- Advantages: fast execution, predictable output
- Disadvantages: The limited variety of the questions does not thorougly understand the context, rigid structure

3. Generation by calling one model
- Examples: T5-based QG, BART-based systems, GPT-4 prompting
- Approach: using one LLM model, they generate questions from a given context
- Advantages: a relatively simple implementation
- Disadvantages: the tendency to hallucinations, lack of validation, limited to the existing text

4. Academic research
- Examples: Neural Question Generation [7], Harvesting Paragraph-level QA [8]
- Approach: exploratory prototypes focused on specific aspects of the QA generation
- Advantages: state-of-the-art techniques, peer-reviewed methodologies
- Disadvantages: usually not production-ready, require the existing datasets and lack the end-to-end pipeline

5. Commercial AI platforms
- Examples: OpenAI Playground, Claude Projects, Anthropic Console
- Access: general-purpose LLM interfaces with the QA generation capability
- Advantages: powerful models easy to use
- Disadvantages: no automated workflow, no specialized validation, no web retrieval

The presented system is positioned as a hybrid solution between fully automatic calls with a single model and expensive manually created data offering:
- 50-75% quality of the humancreated pairs
- 90-95% faster than the manual process
- $0.05-0.15 per QA pair vs. $2-10 for the manual creation
- Scalability up to hundreds of the QA pairs per day
- End-to-end automation with a minimal human intervention

6.4 Application and limitations

The presented system for the automatic generation of QA pairs offers a universal solution for a wide application across different domains and industries. The flexibility of the configuration parameters and the scalability of the architecture enables adaptation to specific requirements of different use cases, from individual educational projects to enterprise-level pipelines for content generation. The system proves to be especially valuable in the contexts where high quality educational content is needed with limited human resources, as well as in scenarios where traditional approaches to creating QA pairs become a bottleneck for content scaling. By combining the automatic web search, multi-agent validation and RAG-based generation, the system creates consistently high-quality educational materials meeting the standards of professional educational platforms.

The system can be used in:

1. Education:
- Creation of tests and quizzes
- Generation of learning materials
- Automatic systems tutoring

2. Training the LLM models (with on additional human intervention):
- Question - answering models
- Information retrieval sistems
- Conversational AI

3. Content creation:
- FAQ sections
- Knowledge - based papers
- Interactive learning contents

4. Research:
- Dataset generation for the NLP research
- Evaluation criteria
- Comparative studies

Some of the main limitations of the system along with the possible solutions are:

1. Errors, losses and long execution due to the cloud LLM services
- Consequence: a smaller number of the final sources, generated questions and output per minute
- Solution: locally hosted LLM (ollama[4])

2. Hallucinations in the generated questions and answers as a result of the LLM errors
- Consequence: reduced quality of the dataset and uncertainty in the final results
- Solution: inevitable final human intervention in cases insiteling accurate data

3. Impossible dataset generation in specific domains
- Consequence: limited use of the system
- Solution: a separate processing branch with additional parameters and a different source selection method based on a specific domain

4. Challenge in evaluating the results
- Consequence: difficult improvements and progress with no measuring quality
- Solution: Human-in-the-loop evaluation of each individual processing step using the feedback to improve the system and an eventuall expan in with fine-tuned LLM models[5]

7 CONCLUSION

The paper implements a complex system for an automatic generation of datasets for the QA pairs using advanced AI technologies and modern web frameworks. The system can be used for various functionalities provided by the current technologies a they are explained in detail to simplifay future implementations the development of similar AI-driven applications.

For a FastAPI framework, the paper shows how to build a scalable backend API with a minimal boilerplate code. Moveover, the Pydantic integration enabling on automatiely data validating and type safety throughout the entire application stack also performs well. The WebSocket implementation extremely powerful for real-time monitoring of the progress of long-running AI operations, allowing users to monitor each processing stage with no polling mechanism.

Particularly well-performing are the CrewAI framework and multi-agent architecture. CrewAI excels in multi-agent orchestration, offering robust support for task management, inter-agent communication and error handling [9]. The RAG approach using the ChromaDB vector significantly reduces the hallucination problem

[4] Ollama - open-source command line tool that simplifies local launch and management of LLMs

[5] Fine-tuned models - pre-trained LLMs that have undergone additional training on a smaller, domain-specific dataset to adapt to a specific task or use case

characteristic of a direct LLM generation, while the multi-stage validation ensures that only the high-quality QA pairs pass through the entire pipeline.

Future improvements will include the use of fine-tuned models, Redis caching, Celery architecture for horizontal scaling and an improved error recovery the process.

The developed system enables the pesser to perform an automatic generation of educational QA pairs from the web content, to configure custom tasks with a precise control over the complexity, type of question, number of sources, and validation. The system autonomously searches the Internet, processes the content, synthesizes the knowledge base, and generates validated QA pairs that achieve 70-85% of the quality of the human-created pairs at a drastically lower cost and time. The real-time progress monitoring provides a transparent insight into each processing phase, while the multi-agent architecture ensures a consistently high quality by on automatic validation, ranking and editing.

REFERENCES

[1] Ruishen Liu, Shaorong Xie, Xinzhi Wang, Xiangfeng Luo, Hang Yu „FKQG: Few-shot question generation from knowledge graph via large language model in-context learning“, Data & Knowledge Engineering Volume 161, January 2026

[2] P. Wang, T. Shi, C. K. Reddy, Text-to-sql generation for question answering on electronic medical records, in: Proceedings of The Web Conference 2020, 2020, pp. 350–361.

[3] Michael McCool, Arch D. Robison, James Reinders „Chapter 9 – Pipeline“, Structured Parallel Programming 2012, Pages 253-262

[4] Wetrocloud, Why Markdown is the best format for LLMs, 2025. Dostupno na: https://medium.com/@wetrocloud/why-markdown-is-the-best-format-for-llms-aa0514a409a7

[5] Abdullah Mudzakir, Jip Tyrone Ativiirya Janaprasetya, Emanuel Kayowuan, Ventje Jeremias Lewi Engel, Bagaskoro Saputro „Evaluating Three Large Language Models for Security Incident Detection and Analysis“, The 10th International Conference on Computer Science and Computational Intelligence 2025

[6] Alex Diaconu, WebSockets explained: What they are and how they work, 2025 Dostupno na: https://ably.com/topic/websockets

[7] Shasha Guo, Lizi Liao, Cuiping Li, Tat-Seng Chua, „A Survey on Neural Question Generation: Methods, Applications, and Prospects“, 2024, https://arxiv.org/pdf/2402.18267

[8] Xinya Du and Claire Cardie, „Harvesting Paragraph-Level Question-Answer Pairs from Wikipedia“, Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1907–1917 Melbourne, Australia, July 15 - 20, 2018

[9] Kiumarse Zamanian, „A Comparative Study of AI Agent Orchestration Frameworks“, November 2024, https://www.researchgate.net/publication/386083531_A_Comparative_Study_of_AI_Agent_Orchestration_Frameworks ,

Đulaga Hadžić received his PhD from the Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina (B&H). He is employed as a Professor with the Department of Software Engineering, Polytechnic Faculty, University of Zenica (B&H), and the Department of Computer Science and Informatics, Faculty of Electrical Engineering, University of Tuzla. His areas of interest are: web programming, user interfaces, data quality, artificial intelligence and cloud computing.

Amer Pašanbegović graduated Software Engineering from the Polytechnic Faculty of the University of Zenica, Bosnia and Herzegovina (B&H), in 2025. He is currently employed as an AI engineer with Galeyo d.o.o, Sarajevo, (B&H). His fields of interests include: artificial intelligence, software architecture, machine learning, data science, and web programming.

Enes Saletović is an associate professor at the University of Zenica, Bosnia and Herzegovina (B&H), and expert advisor in the Ministry of Internal Affairs of FB&H. He received his B.S. and Ph.D. degrees from the Faculty of Electrical Engineering University of Sarajevo, B&H in 2014. His research interests are mostly in the field of unmanned vehicle remote control, control theory, networks and embedded systems. He has published several scientific papers and a book in the field of computer science, networks, control theory and optimization. He is a member of several expert commissions in B&H.