Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 275 (2026) 602–609

Procedia Computer Science

www.elsevier.com/locate/procedia

7th International Conference on AI in Computational Linguistics

A Modular Multi-Agentic Architecture for Automating Multiple-Choice Question Generation in Language Assessment

Peng Zhao, John Blake*, Evgeny Pyshkin

University of Aizu, Aizuwakamatsu, 965-0006, Japan

Abstract

The creation of high-quality multiple-choice questions (MCQs) for language assessment is a labour-intensive task, often requiring careful balancing of linguistic appropriacy, proficiency level, topic coverage, and distractor plausibility. We present a modular, multi-agentic system built using LangChain to generate appropriate MCQs. Each agent in the system is responsible for a distinct task in the question generation pipeline. These tasks range from topic selection and question formation to answer validation, distractor generation, and coverage checks. The system supports flexible substitution of Large Language Models (LLMs), allowing comparative benchmarking across tasks in terms of generation accuracy and latency. Human expert assessment of item quality confirmed that the best-performing configurations yielded scores exceeding 95% in grammatical correctness with satisfactory speed. Our results demonstrate that multi-agent LLM-based architectures can effectively automate complex educational content creation workflows while offering transparency, modularity, and fine-grained controllability. The proposed system offers a reusable design pattern for intelligent educational content generation in broader domains.

Keywords: Multi-agent systems; Large Language Models (LLMs); Automatic question generation; Language assessment; Educational content generation

1. Introduction

Question answering is a popular research field while its counterpart question generation is an under-represented domain of natural language processing. Researchers have worked on creating questions using string manipulation from declarative statements [1] while more sophisticated systems draw on deep learning or large language models [2]. The labour intensity in the creation stage and the evaluation stage varies by question type. Fig. 1 shows the main type of questions used in the Moodle learning management system with an evaluation of the relative intensity of each stage. Essay questions are evaluated as easy to set, but take far longer to grade. Conversly, multiple-choice questions (MCQs) can easily be graded easily, but the manual creation of MCQ is time-consuming and labour-intensive [3].

* Corresponding author
E-mail address: jblake@u-aizu.ac.jp

MCQs comprise a question and a correct answer pair (QAP) and a set of distractors. MCQs are frequently used in language assessment, particularly in large-scale, high-stakes examinations such as TOEFL, TOEIC, IELTS, and university entrance exams.

	Creation	Evaluation
Multiple choice	high	low
True/False	low	low
Matching	high	average
Short answer	low	average
Numerical	average	low
Essay	low	high

	Creation	Evaluation
Calculated	average	low
Calculated multichoice	average	average
Calculated simple	low	low
Drag and drop into text	high	low
Drag and drop markers	average	low

Fig. 1. Relative labour intensity of creation and evaluation stages by question type in Moodle.

Research has identified several advantages of MCQs, including their scalability, reliability, and compatibility with automated scoring systems. When appropriately constructed, MCQs can be used to assess both language knowledge (e.g., grammar and vocabulary) and language skills, such as reading comprehension. However, studies have documented considerable variability in MCQ quality across different test contexts. Research has identified several factors that can compromise item validity, including ambiguous wording, culturally biased content, and syntactic complexity unrelated to the construct being measured. These sources of construct-irrelevant variance can affect test performance independently of the language knowledge or skills the assessment is designed to measure, potentially threatening the validity of score interpretations.

Despite some acknowledged advantages, empirical studies and expert guidelines highlight the complexity and resource intensity of developing high-quality MCQs. Item development frameworks specify multiple quality criteria that must be simultaneously satisfied, including: (1) alignment between each item and the target construct, (2) clarity and defensibility of the correct answer, and (3) plausibility of distractors while maintaining their incorrectness. Additional considerations such as ensuring adequate content coverage, avoiding item overlap, and calibrating difficulty levels further increase development demands.

The ease of access to large language models (LLMs) offers new possibilities for automating MCQ generation. However, existing LLM-based systems often treat the task as a monolithic text generation problem, without modular control or transparency across subtasks. To address this, we introduce a multi-agentic system built using LangChain that decomposes the MCQ generation pipeline into discrete, interpretable agents. Each agent specializes in one aspect of question creation, such as topic selection, question formation, distractor generation, or coverage balancing. This allows for targeted improvements and fine-grained evaluation. This modular architecture enhances flexibility, facilitates model benchmarking, and supports human-in-the-loop validation.

The remainder of the paper is structured as follows. Section 2 reviews related work on question generation, educational NLP, and agent-based language processing. Section 3 describes the system architecture and functionality of each agent in the pipeline. Section 4 details the integration of large language models and system infrastructure. Section 5 presents both automated and human evaluation of the system output. Section 6 discusses the results and limitations, while Section 7 concludes the paper with future directions.

2. Related Work

Automatic Question Generation (AQG) has been studied primarily through natural language processing techniques such as Named Entity Recognition (NER) and Semantic Role Labeling (SRL). Early work focused on rule-based and template-driven methods, including cloze questions, discourse-based transformations, and ontology-guided multiple-choice generation. Later systems incorporated semantic features and multi-agent frameworks to produce questions aligned with Bloom's taxonomy or tailored to multimedia learning. Recent approaches increasingly emphasise se-

mantic information and leverage transformer-based architectures for efficient question generation [4]. Zhang [5] proposed two complementary methods for multiple-choice question generation—TP3, a T5-based end-to-end pipeline with preprocessing and postprocessing, and MetaQA, a meta-sequence learning scheme alongside a novel distractor generation approach that harnesssed a three-pronged approach involving (1) conversion to numerical or ordinal numbers, (2) looking up entities, or (3) utilizing word embeddings and WordNet [6]. While these approaches have shown promise in generating MCQs and WH-questions, most research remains language- and domain-specific, with evaluations limited to small-scale studies. Current AQG systems often lack adaptability across question types and still face challenges in scalability, linguistic quality, and semantic depth. It should also be noted that there is still no consensus on a reliable automatic evaluation metric specifically designed for question generation [4]. A survey of 37 AQG papers reveals highly diverse and inconsistent evaluation practices, highlighting the urgent need for a common framework to enable reliable comparison across systems [7].

Agent-based NLP systems have emerged as a promising paradigm, emphasizing autonomous decision-making and interaction with external environments to tackle complex information processing tasks. Earlier approaches relied on rule-based or standalone NLP models, which often lacked flexibility across domains and task types. Recent advances, exemplified by systems like Eunomia, integrate large language models as the central agent, enabling not only text generation but also planning, reasoning, and tool utilization [8]. Such systems demonstrate strong potential in scientific literature mining, educational assessment, and multimodal learning, particularly by automating structured data creation and handling complex tasks such as relation extraction, co-reference resolution, and argument mining. Compared to conventional NLP pipelines, agent-based frameworks focus on collaboration, domain-specific toolkits, and chain-of-verification mechanisms, marking a significant shift toward autonomy, robustness, and explainability in NLP applications [9].

LangChain is a widely adopted framework for constructing LLM-based applications, offering modular abstractions such as chains, tools, memory, and agents to support linear and retrieval-augmented workflows. Its design enables rapid prototyping and integration of external APIs, but its default orchestration model remains largely sequential. To address the need for adaptive and cyclical workflows, LangGraph extends LangChain with a stateful, graph-based paradigm, supporting dynamic branching, retries, parallel execution, and shared memory across agents. Recent studies highlight the suitability of LangGraph for multi-agent orchestration, enabling patterns such as planner–executor, critic–reviewer, and concurrent specialist models, which improve scalability and resilience in complex domains. Beyond orchestration, comparative surveys further position LangChain and LangGraph within a broader toolchain that includes LangSmith for observability and governance. This triad delineates a lifecycle where LangChain accelerates construction, LangGraph operationalizes adaptive orchestration, and LangSmith ensures monitoring, evaluation, and ethical compliance. Together, these frameworks advance Artificial Intelligence (AI) from prototypes to production; nevertheless, open challenges remain in complexity management, interoperability, and transparency. Emerging directions include modular state abstraction, performance-aware routing, and auto-evaluation pipelines that promise more robust and accountable agentic AI systems [10, 11].

3. System Design

3.1. Architectural Overview

The proposed system adopts a modular, agent-oriented architecture designed to support the scalable and interpretable generation of multiple-choice questions (MCQs) for language assessment. As illustrated in Figure 2, each agent in the system is responsible for a specific subtask within the question creation pipeline. These agents are instantiated through the LangChain agent abstraction, while their orchestration is governed by LangGraph, which enables the dynamic flow of control through a directed graph. This architecture enhances transparency by isolating each functional component, making it easier to debug and evaluate system behavior at a granular level. It also promotes modularity, as individual agents can be fine-tuned, re-prompted, or replaced without altering the overall pipeline. Furthermore, the design lends itself to scalability, as multiple agents can be deployed in parallel, or reused across tasks, domains, or even languages. The separation of concerns across agents is a key factor in making the system adaptable to future expansions, including additional item types or feedback loops.

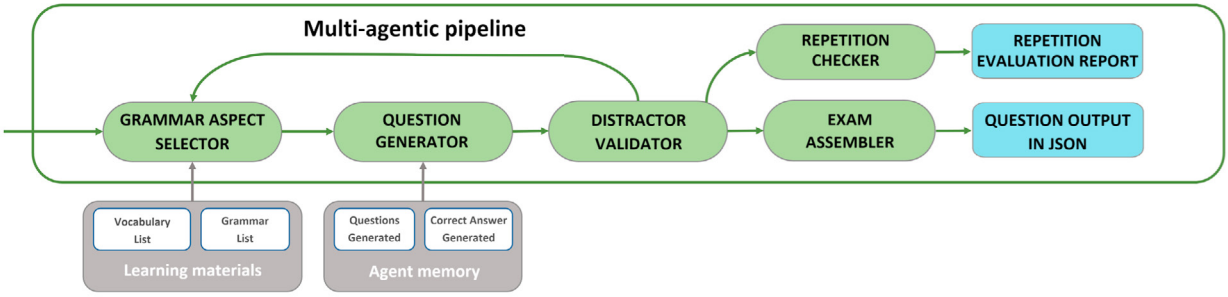

Fig. 2. System architecture of the multi-agent MCQ generation pipeline.

3.2. Agent Responsibilities

Table 1. Agent responsibilities in the MCQ generation system.

Agent	Responsibility
Grammar Aspect Selector	Selects grammar and vocabulary from pre-defined lists based on the generation history.
Question Generator	Applies assigned grammar and vocabulary to generated questions.
Distractor Validator	Ensures distractors differ from the correct answer and are grammatically or semantically inappropriate.
Repetition Checker	Analyses and summarises repetition of language points covered in generated questions.
Exam Assembler	Formats validated questions into a structured `JSON` file for downstream use.

The system functionality is realized through five agents, whose responsibilities are summarized in Table 1. First, an agent selects and assigns vocabulary items and grammatical concepts as references for the next generation step, and a question generator responsible for creating single-sentence questions with four choices, i.e. one answer and three distractors, during each generation cycle. The grammar aspect selection agent completes its task by referencing the past selection records, while the question generator considers the history of generated questions by examining the conversation history records, enhanced by the LangChain framework.

Multiple validation steps are integrated into the system. For instance, the answer validation agent processes the correct answer provided by the question generator. This agent accomplishes its function by first extracting the designated correct answer and then using it to construct a new sentence. Subsequently, the model verifies whether both the grammar and meaning of the sentence are correct and reasonable. Another agent, the distractor validator, examines the three options designated as inappropriate. Similarly, sentences are first formed using the distractors. If any sentence formed by distractors is recognized as reasonable, the question is considered unacceptable since it would confuse learners with multiple possible correct answers. The failure of either the answer validator or distractor validator results in the rejection of the current question.

Questions that pass the validation process are considered qualified for summarization. The system workflow involves repetition checking and exam assembly, but it is notable that one major difference between these processes and the previously mentioned validation procedures is that neither of these non-validation processes eliminates existing questions. The repetition checker accepts and analyzes the generated and validated appropriate questions collectively at the end of the workflow, then generates a report regarding the potential for repetition in the words and grammar focused on during this specific generation cycle. The generated report serves as a reminder and reference for system users. Furthermore, all questions that have passed the validation process are assembled into a `JSON` file and exported automatically.

3.3. Memory and State Management

To support continuity and avoid redundancy in the generation process, the system implements memory functionality using both code-based grammar marking technology and framework-based memory modules within LangChain.

To select different vocabulary items and grammatical structures for question generation, the grammar point selection agent marks the selected objects each time and calculates the coverage rate for each grammar category. The marking and calculation process is rule-based, implemented directly in source code without the use of an LLM. In subsequent selection rounds, the source code excludes previously used vocabulary items from the selection range and conveys information about the coverage rate of each word and grammar category to the model for new LLM-based selection. Moreover, the system applies the memory functionality of LangChain to conversation records for the question generator. The scope of information in memory falls into two categories. The first consists of prompt messages, which include the assigned words and grammar structures. The second consists of response messages, which primarily contain the question, the correct answer, and the distractors. Additionally, the prompts consistently include instructions requiring the model to check conversation history and avoid repetitiveness.

3.4. Workflow Control and Error Handling

The system workflow is structured around a loop that runs from the topic generator through to the distractor validator. The number of iterations of this loop can be specified and adjusted by users directly in the source code. Once the loop completes, the workflow proceeds to repetition checking and final assembly, each of which is executed only once in the main system. System execution terminates if errors occur during the selection of words or grammar structures as language materials. In the case of validation failures, the system excludes the unqualified questions and either continues to the next iteration of the loop (if additional rounds remain) or, if all assigned rounds have been completed, moves on to the final repetition analysis.

3.5. Output Format and Interoperability

The system exports its final output as a structured `JSON` object. Each question includes metadata such as the question text, correct answer, distractors, topic, difficulty level, timestamp of generation, and the model used. This structure facilitates downstream analysis, including the evaluation of question difficulty, usage statistics, or learner response patterns. The `JSON` format is designed for interoperability with Learning Management Systems (LMSs), digital assessment tools, and item banking platforms. By including detailed metadata, the system also enables longitudinal tracking of content coverage and supports future feedback loops for quality assurance.

4. Implementation

The core functionality of each agent is implemented using large language models (LLMs) accessed through the LangChain interface. In our initial implementation, we evaluated three state-of-the-art models: GPT-4o, Claude 3 Opus, and Gemini 1.5 Pro. These models were selected for their competitive performance, broad context windows, and compatibility with LangChain LLM wrappers. Agents were selected according to the most reliable results for a particular task. All agents were built based on the same model to improve overall coherence. For instance, the Question Generator agent, as well as the validator and checker, are created based on the adaptation of GPT-4o model to ensure they enjoy logical consistency. Moreover, based on the task features of each agent, their temperature was adjusted accordingly. Table 2 summarizes the temperature settings. For the Question Generator, a temperature of 0.7 is set for pursuing high creativity while a temperature of 0.3 is set for the repetition checker to increase reliability. Prompts for each agent were carefully engineered to enforce formatting conventions and semantic clarity. Such pattern of design also applied in the temperature setting of other agents. We used prompt templates that specified input constraints, output structure (e.g., `JSON` or tabular format), and evaluation criteria.

The backend is implemented in Python and built around the LangChain. LangChain provides the abstraction layers for defining agents, tools, chains, and memory. Each agent is implemented as a `Runnable` object with a custom prompt template and a model binding. To support persistent and searchable memory across sessions, we integrated separate vocabulary and grammar lists. This allows agents to query within the specific scope assigned and controlled by experts, as well as make full use of previous outputs via semantic similarity, enabling context-aware generation and avoiding duplication. All input, as well as output, is structured in `JSON` to ensure interoperability with downstream applications. Each generated question is stored with rich metadata, including the question, answer options, correct answer, and generation time as shown in Fig. 3.

Table 2. Temperature settings for agent tasks.

Agent	Temperature
Grammar Aspect Selector	0.5
Question Generator	0.7
Answer Validator	0.3
Distractor Validator	0.3
Repetition Checker	0.7


```
{
  "question_number": 6,
  "content": "The executive has been
      working () to meet the deadline for
       the new product lauch. (A) fast (B
      ) fastest (C) faster (D) the
      fastest",
  "correct answer": "A",
  "timestamp": "2025-04-26 05:59",
}
```

```
{
  "question_number": 7,
  "content": "By implementing these new
      policies, we () potential conflicts
       within the team. (A) eliminate (B)
       eliminating (C) eliminated (D)
      eliminates",
  "correct answer": "A",
  "timestamp": "2025-05-12 12:27",
}
```


Fig. 3. Example JSON Output of Exam Assembler.

5. Evaluation

To assess the performance and usability of the proposed system, we conducted a two-part evaluation comprising: (1) automated evaluation towards the performance of each agent, and (2) LLM and human expert evaluation towards the overall accuracy of generatiion. This dual approach enables us to analyze both the technical efficiency of the system and the educational validity of its outputs.

5.1. Automated Evaluation

In the first stage of evaluation, we focused on measuring the runtime efficiency of individual agents. By analyzing their generation times on core tasks, we aimed to quantify the computational cost of each component and assess the system responsiveness from a technical perspective. Table 3 presents the runtime statistics of three core agents in the proposed system. The Question Generator required on average 2.25 seconds, with a minimum of 1.47 seconds and a maximum of 3.99 seconds, while its median runtime was 2.22 seconds. The Distractor Validator demonstrated the fastest execution, averaging only 1.43 seconds and ranging from 0.89 to 2.05 seconds. In contrast, the Repetition Checker exhibited the highest runtime, with an average of 2.94 seconds and a maximum of 4.27 seconds. These results indicate that runtime requirements vary across agents, reflecting the different computational complexities of their tasks. Overall, the measured runtimes are relatively low, all remaining within a few seconds, which demonstrates the technical feasibility of integrating the agents into real-time or near-real-time educational applications. The Distractor Validator shows the highest efficiency, while the Repetition Checker, although slower, still maintains acceptable performance. This balance between speed and functionality suggests that the system can support scalable deployment without imposing excessive computational overhead.

5.2. Human Evaluation

In addition to automated metrics, we conducted a human evaluation to assess the pedagogical quality of the generated MCQs. A sample of 100 questions was prepared and independently reviewed. Each question was evaluated for grammatical correctness, ensuring that the correct choice was both grammatically accurate and meaningful in context. Choice clarity was examined to confirm that no question contained multiple or ambiguous correct answers. Finally,

Table 3. Running of LLMs for key agent tasks.

Key Agents	Generation Time (s)			
	Mean	Minimum	Maximum	Median
Question Generator	2.25	1.47	3.99	2.22
Distractor Validator	1.43	0.89	2.05	1.49
Repetition Checker	2.94	2.09	4.27	2.93

suitability was evaluated to verify that the questions were appropriately challenging and conformed to the style of TOEIC items. The evaluation results are summarised in Table 4. The assessment procedure followed a sequential approach, beginning with grammatical correctness. Items containing major grammatical errors were assigned a score of 0 and excluded from subsequent stages. Only questions free from such errors proceeded to the evaluation of choice clarity. Likewise, any item with multiple or ambiguous correct answers received a score of 0 and was excluded from the final stage, which examined the suitability of each question in terms of difficulty and stylistic conformity to TOEIC standards. The score underlines the strength in grammatical correctness and choice clarity of the generation result. However, some of the generated results failed in aligning with the style of TOEIC, negatively impacting the suitability score.

Table 4. Human evaluation results for MCQ quality (0–10 scale, with higher values indicating better performance).

Aspect	Grammatical Correctness	Choice Clarity	Suitability
Mean	9.52	9.22	8.34
Median	9.62	9.21	8.23
Mode	9.00	9.50	8.50

6. Discussion

The results of our evaluation indicate the current system can generate questions with satisfactory quality, including suitable difficulties in grammar and vocabulary, as well as acceptable speed. The topic and style of the generation result are appropriate for TOEIC-style practice activities. In the typical idea sample introduced, the whole generation process of each question took approximately $6-8$ seconds. Considering that Part 5 of the TOEIC test consists of a set of 30 questions, a similar volume of TOEIC-style test questions can be generated in about 3 minutes. This can meet the need regarding speed, as well as address the remaining room for improvement [12]. One promising direction involves the tune regarding the usage of memory, both langchain-related and independent, which shows potential in further improving the balance between working load and speed. Regarding the accuracy of the generation, a satisfactory grammatical correctness can be achieved under the current multi-agent system. Although unsatisfactory results are observed in the initial generation results, the answer validator and distractor can perform their duty to effectively eliminate them. Additionally, the overall weaker part, namely suitability, is solvable. With a clearer assignment of sample questions, the quality regarding this aspect has improved, and the same approach is considered adaptable for future improvement.

Although the system demonstrates promising results, human verification is still needed; the current system, therefore, substantially speeds up the creation process, but does not fully automate it. The limitations impacting the system include the following. First, the current generation result is not as stable as expected. The update of the model greatly influences the output of this multi-agent system, and sometimes even causes the previous results to be unreproducible [13]. Even more seriously, degradation occurred numerous times during the development process, and the reactions towards such issues are very limited, which weakens the robustness of this system. This happens when the function adjustment is applied to the model, and even if the update is described as an upgrade officially, the behavior change of the system should go through a new round of evaluation. The export of the generation result, as a result, becomes a must, and the evaluation is also performed based on one of the kept outcomes, instead of newest output. Similarly, another issue involves the relatively large gap regarding the maximum and minimum value of running time

cost. External influence, including the load of netork and web server should be taken into consideration, while the tolerant to a unstable service is expected to the system user [14]. Finally, we aim to enhance a more reasonable setting of temperature for each agent. Although the current temperature settings produce satisfactory output, the temperature settings for each agent is a topic that deserves further investigation. Regarding the single agent, the balance of creativity and reliability can be achieved in a consistent and agile adjustment of temperature. Moreover, when the scope borders, a more systematic tuning of temperature could balance the task and responsible load of agents, leading to improved stability, diversity, and overall coordination among agents [15].

7. Conclusion

This paper presented a modular, multi-agent system for the automated generation of multiple-choice questions using LangChain. Each agent in the system fulfills a distinct role in the question creation pipeline, enabling precise control over generation quality, system behavior, and model configuration. Evaluation results show that the system produces usable questions with acceptable quality on both grammatical correctness, choice clarity, and topic properness. Furthermore, the running time of each agent can ensure the productivity in the generation of questions in a set unit. The instability of the service is one of the main challenges. Future work includes upgrading the robustness of the system as well as gaining more control over the overall behaviour of the system and improving the output of each agent.

References

[1] D. T. Vu, J. Blake, Design and development of a question generator for learners of English, in: SHS Web of Conferences, volume 102, EDP Sciences, 2021, p. 01011. doi:https://doi.org/10.1051/shsconf/202110201011.

[2] N. Mulla, P. Gharpure, Automatic question generation: a review of methodologies, datasets, evaluation metrics, and applications, Progress in Artificial Intelligence 12 (2023) 1–32. doi:https://doi.org/10.1007/s13748-023-00295-9.

[3] B. Das, M. Majumder, S. Phadikar, A. A. Sekh, Automatic question generation and answer assessment: A survey, Research and Practice in Technology Enhanced Learning 16 (2021) 1–15. doi:https://doi.org/10.1186/s41039-021-00151-1.

[4] C.-Y. Lu, S.-E. Lu, A survey of approaches to automatic question generation: from 2019 to early 2021, in: L.-H. Lee, C.-H. Chang, K.-Y. Chen (Eds.), Proceedings of the 33rd Conference on Computational Linguistics and Speech Processing (ROCLING 2021), The Association for Computational Linguistics and Chinese Language Processing (ACLCLP), 2021, pp. 151–162. URL: https://aclanthology.org/2021.rocling-1.21.

[5] C. Zhang, Automatic Generation of Multiple-Choice Questions, Ph.D. thesis, University of Massachusetts Lowell, 2022. URL: https://arxiv.org/abs/2303.14576.

[6] C. Zhang, Y. Sun, H. Chen, J. Wang, Generating adequate distractors for multiple-choice questions, arXiv preprint arXiv:2010.12658 (2020). URL: https://arxiv.org/abs/2010.12658.

[7] J. Amidei, P. Piwek, A. Willis, Evaluation methodologies in automatic question generation 2013-2018, in: E. Krahmer, A. Gatt, M. Goudbeek (Eds.), Proceedings of the 11th International Conference on Natural Language Generation, Association for Computational Linguistics, 2018. doi:https://doi.org/10.18653/v1/W18-6537.

[8] M. Ansari, S. M. Moosavi, Agent-based learning of materials datasets from the scientific literature, Digital Discovery 3 (2024) 2607–2617.

[9] D. Calvaresi, et al., Exploring agent-based chatbots: A systematic literature review, Journal of Ambient Intelligence and Humanized Computing 14 (2023) 11207–11226.

[10] K. Pelluru, LangChain & LangGraph in production: Architectures for multi-agent LLM systems, Journal of Data and Digital Innovation (JDDI) 2 (2025) 1–9.

[11] R. Sapkota, et al., LangChain vs. LangGraph vs. LangSmith: Taxonomies of agentic AI toolchains for end-to-end orchestration, Authorea Preprints (2024). URL: https://www.authorea.com.

[12] I. Gim, et al., Prompt cache: Modular attention reuse for low-latency inference, in: Proceedings of Machine Learning and Systems, volume 6, 2024, pp. 325–338.

[13] J. Echterhoff, et al., MUSCLE: A model update strategy for compatible LLM evolution, arXiv preprint arXiv:2407.09435 (2024). URL: https://arxiv.org/abs/2407.09435.

[14] Y. Wang, et al., Towards efficient and reliable LLM serving: A real-world workload study, CoRR (2024). URL: https://arxiv.org/abs/.

[15] J. Xie, et al., Calibrating language models with adaptive temperature scaling, arXiv preprint arXiv:2409.19817 (2024). URL: https://arxiv.org/abs/2409.19817.