CAPRA: Scaling Feedback on Software Architecture Deliverables with a Multi-Agent LLM System

Marco Becattini[1], Niccolò Caselli[1], Matteo Minin[1], Roberto Verdecchia[1⋆], and Enrico Vicario[1]

Department of Information Engineering, University of Florence, Florence, Italy
marco.becattini@unifi.it
niccolo.caselli3@stud.unifi.it
matteo.minin@stud.unifi.it
roberto.verdecchia@unifi.it
enrico.vicario@unifi.it

Abstract. Automated assessment in software engineering education has advanced significantly for code grading and essay scoring. However, reviewing software architecture deliverables, which requires analyzing structural completeness and requirements traceability, has not yet been fully automated. Applying Large Language Models (LLMs) to this task requires robust architectures to ensure technical feedback is accurate and reliable for students. This paper presents CAPRA (Configurable Architecture Proficiency Report Assessment), a multi-agent LLM system that analyzes software architecture deliverables to generate personalized, template-compliant LaTeX feedback. As a core design choice, CAPRA coordinates multiple specialized agents and employs a Python-based microservice for multi-modal document extraction, utilizing PyMuPDF and vision-enabled LLMs (specifically `gpt-4o`) to parse text and UML diagrams. To ensure educational reliability and mitigate hallucinations, CAPRA introduces a deterministic Evidence Anchoring step using fuzzy matching via normalized Levenshtein distance, along with a `ConsistencyManager` agent that cross-verifies, deduplicates, and merges findings. System performance is assessed using a structured eight-criterion binary evaluation taxonomy covering: (i) extraction completeness, (ii) feature validation, (iii) issue grounding and severity detection, (iv) recommendation specificity and traceability, and (v) template and tone compliance. A preliminary empirical evaluation on 10 student reports shows that CAPRA satisfied 88.8% of the evaluated criteria under a strict two-rater aggregation rule, achieved moderate inter-rater agreement with human evaluators ($\kappa = 0.582$), and processed each report in slightly over 4 minutes. While these results support the viability of LLM-supported architectural feedback, human oversight remains essential for subjective assessment dimensions.

Keywords: Large Language Models · Multi-Agent Systems · Automated Assessment · Software Engineering Education

⋆ Corresponding author.

1 Introduction

Software engineering education heavily relies on project-based learning, in which students are expected to produce comprehensive documentation, such as requirements specifications, UML diagrams, and architectural designs [37]. Reviewing these technically dense artifacts is highly time-consuming, requiring instructors to possess both deep domain knowledge and the capacity to provide tailored, constructive feedback [12]. As class sizes grow, this review bottleneck significantly limits the frequency and quality of the formative feedback students receive [30], while industry continues to report significant gaps between graduate abilities and workplace expectations in software engineering contexts [34].

While automated tools have matured significantly for code grading (e.g., through static analysis and unit testing) [19,1], providing formative feedback on open-ended software architecture documentation remains a difficult challenge. Recent advances in Large Language Models (LLMs) [5] have demonstrated promising capabilities for automated review tasks [10,18]. However, applying LLMs to complex, multi-modal architectural documents requires careful orchestration to ensure that the generated feedback is reliable, accurate, and pedagogically valuable, avoiding the risk of hallucinated [17] or redundant critiques that could misguide students.

To address these challenges, this paper introduces the Configurable Architecture Proficiency Report Assessment (CAPRA), an automated tool designed to support software engineering students by providing reliable, formative feedback on architectural deliverables without assigning grades. CAPRA adopts a multi-agent architecture in which specialized components handle multi-modal extraction and dimension-specific analysis, while a verification and consistency stage checks cited evidence and consolidates overlapping feedback.

Our preliminary empirical evaluation on 10 student reports suggests the viability of the approach. CAPRA satisfied 88.8% of the evaluated criteria under a strict two-rater aggregation rule. The system processes a complete architectural report in slightly over 4 minutes at a cost of roughly $0.44, compared to an estimated 30–45 minutes for a thorough manual review.

This contribution is intended for software engineering educators and researchers seeking pedagogical insights into solutions able to provide scalable, high-quality, customized feedback. The main contributions of this work are summarized as follows:

- **A Multi-Agent Assessment Workflow**: A workflow for software architecture deliverable assessment that integrates multi-modal extraction, dimension-specific analysis, and template-compliant feedback generation.
- **Deterministic Evidence Verification**: A mechanism based on fuzzy matching [31] via normalized Levenshtein distance [43] that checks cited findings against the source document before they are included in the final feedback.
- **An Exploratory Empirical Study**: A preliminary evaluation of CAPRA on 10 student project reports, using a custom eight-criterion taxonomy and a separate set of 10 reports for knowledge-base construction.

2 Related Work

In this section, we discuss the work more closely related to our research. An overview of the most representative related literature is presented in Table 1 and is further discussed below.

Table 1. Comparison of representative AI/LLM-based assessment and multi-agent approaches across the dimensions relevant to architectural-deliverable review. CAPRA is listed last to position the gap it fills.

Approach	**Artifact**	**Method**	**Grounding**	**Output**
Static-analysis graders, e.g., ArTEMiS [22]	Code	Tests / rules	Deterministic oracle	Grade + feedback
LLM UML grading [4]	UML diagrams	LLM scoring	Compared with TA scores; no source-span anchoring	Score
LLM code feedback [32]	Code errors	LLM repair + explanation	Runtime / exact-match validation	Feedback
LLM short-answer grading [42]	Text answers	Multi-step LLM grading	Rubric + audit loop	Grade
LLM-as-a-judge methods [44,28,20]	Open-ended text	Rubric/reference judging	Human/rubric alignment; no source-span anchoring	Score / preference
Generative MAS for software/ tasks [13,33,41]	Software tasks	Role-based agents	Internal review / tool checks	Software artifacts
CAPRA (our contribution)	Architecture reports	Multi-agent assessment	**Deterministic evidence anchoring**	**Formative feedback**

2.1 Automated Assessment in SE Education

Automated assessment is widely used to scale programming education: tools like ArTEMiS [22] provide immediate feedback through static analysis and unit testing [16,36], increasingly aided by LLM-based assistants. Such tools, however, target *deterministic* artifacts. Source code can be checked against an objective oracle (compilation, unit tests), whereas open-ended documentation such as requirements, designs, and UML diagrams cannot, and its assessment remains a persistent challenge [30,14].

To bridge this gap, a growing body of work applies Large Language Models (LLMs) to open-ended artifacts [8], grading UML diagrams [4], generating feedback on programming syntax errors [32], and scoring short-answer responses [42]. As Table 1 shows, these efforts generally focus on a single artifact type or representation and do not provide deterministic evidence anchoring of feedback claims to source spans in multi-modal architecture documents. This can produce *hallucinations*, that is, fabricated errors absent from the source [10,15,17], which is especially problematic in education, where incorrect feedback can misguide students [12].

2.2 Multi-Agent Systems in SE

Multi-Agent Systems (MAS) improve performance on complex reasoning tasks [11]: frameworks such as MetaGPT [13], AutoGen [41], ChatDev [33], CAMEL [25], and AgentVerse [7] decompose problems into specialized roles that can outperform single-agent or monolithic prompting on complex tasks [26].

Assessment is a recent MAS application. Grade-Like-a-Human [42] pairs a multi-step grading pipeline with a post-grading audit loop, but grades short-answer text and neither handles multi-modal documents nor anchors findings to verifiable source spans.

2.3 LLM-as-a-Judge for Open-Ended Tasks

The "LLM-as-a-Judge" paradigm establishes that frontier LLMs can effectively evaluate open-ended textual outputs [44], yet they exhibit systematic biases [39], favoring longer texts and being sensitive to information order. Researchers counter this by decomposing it into independent, rubric-driven steps, improving reliability [42,28,20]. This motivates two design principles for trustworthy assessment: decomposing the evaluation into focused, single-purpose checks, and grounding each judgment in concrete evidence rather than an opaque verdict.

3 The CAPRA Approach

This section details the architecture and operational workflow of CAPRA. To address the complexity of educational software architecture assessment, the system is designed as a pipeline divided into four main stages (see Figure 1): Document Parsing, Parallel Verification, Evidence Anchoring, and Report Generation.

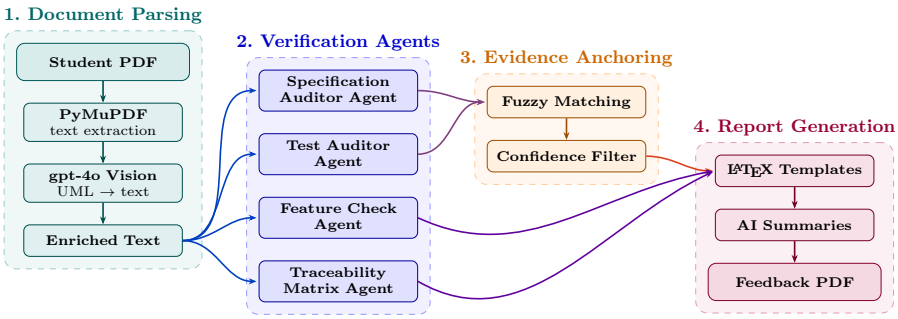

Fig. 1. CAPRA system architecture: four-stage pipeline from PDF ingestion (Document Parsing), through parallel multi-agent evaluation (Verification Agents), evidence-anchored deduplication (Evidence Anchoring), to final report generation (Report Generation).

3.1 Document Parsing and Extraction

Input: Student's raw architectural deliverable (PDF format).
Output: Enriched text representation of the document.

This phase takes the student's PDF report and converts it into a single text format. First, it extracts the main text using the PyMuPDF library [2]. Since architectural documents contain many visual elements like UML diagrams, standard text extraction is not enough [27]. To solve this, the system uses the `gpt-4o` vision model [**?**] to parse images and turn them into structured text descriptions. Finally, these descriptions are inserted into the original text exactly where the images appeared. This creates an enriched, sequential text document that preserves the original flow, making it ready for the next analysis steps.

3.2 Parallel Verification Agents

Input: Enriched text representation of the document.
Output: A collection of raw analytical findings, each with textual evidence and a confidence score.
This stage takes the enriched text as input and analyzes it to identify architectural issues. A central orchestrator activates several specialized AI agents that run at the same time. Each agent examines the entire document, but focuses only on a small task [13,26]. This strategy provides a deep analysis without the confusion that often happens when a single prompt tries to do everything at once.

To guarantee reliable and consistent feedback, the system forces the language models to respond deterministically (by setting a seed for OpenAI's models and temperature zero for all the models used). When an agent finds a problem, it reports the issue, attaches a direct quote from the document as proof, and assigns a confidence score. The main agents are:

- *SpecificationAuditorAgent*: Performs a comprehensive audit to identify flaws in functional and non-functional requirements (e.g., incompleteness, business logic contradictions) and use case specifications (template structure, actor roles). It also verifies the alignment between UML diagrams and the textual narrative, and evaluates the architectural description for pattern correctness and separation of concerns, providing literal quotes as evidence for each finding.
- *TestAuditorAgent*: Audits the test plan to identify critical coverage gaps and verify adherence to the stated testing strategy, extracting relevant evidence quotes.
- *FeatureCheckAgent*: Checks if the document contains the required features based on a grading rubric stored in a database. Instructors can write these rules manually, or the system can automatically extract them from past student reports. In our project, we used a SALLMA workflow [3] to extract all the features of historical documents. The system then groups similar features together using a clustering algorithm, namely HDBSCAN [6], and keeps only the clusters that are present in at least two-thirds of the reports.

For each valid cluster, a representative feature is selected to generate a specific checklist (see Figure 2), which is then used to verify if that feature is present in a new report. This allows instructors to easily update the grading rules as the course evolves.

– *Traceability Matrix Agent*: Analyzes the full document text to build a comprehensive mapping matrix (`Requirement` → `UseCase` → `Design` → `Test`) and identify architectural disconnects.

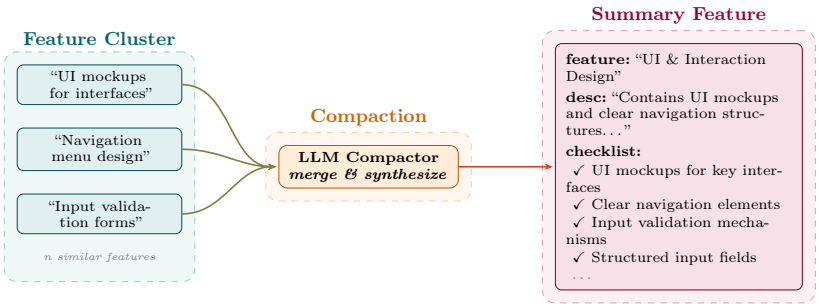

Fig. 2. LLM Compactor: Similar features within a cluster (e.g., "UI mockups for interfaces", "Navigation menu design", "Input validation forms") are merged by an LLM into a single canonical Summary Feature with a structured checklist used by the *FeatureCheckAgent*.

3.3 Evidence Anchoring and Deduplication

Input: The collection of raw analytical findings.
Output: A verified, evidence-backed set of architectural critiques.
The *Evidence Anchoring Service* is a deterministic algorithm that checks if the student actually wrote what the agent claims.

This stage connects the raw issues found by the agents with the actual proof in the report. Specifically, it takes the findings of the two main review agents (*SpecificationAuditorAgent* and *TestAuditorAgent*), each carrying an initial confidence score ($C_{initial}$) assigned by the originating agent, and modulates that score based on how well the agent's cited quote matches the original document text.

After cleaning up the text by removing punctuation and extra spaces, each issue is handled based on the citation length as follows:

– **Missing or Empty Quote:** If there is no quote, the issue is not immediately discarded, but its confidence score is halved (50% penalty). This conservative choice avoids losing potentially valid issues that the agent failed to ground properly.
– **Short Quote (<15 characters):** Very short texts are hard to match safely. Instead of running a complex search, the system applies a 30% penalty to the confidence score (multiplying it by 0.7). This keeps good findings alive without trusting them blindly.

– **Standard Quote:** The system scans the document using a sliding window (W) of length $|Q|$ to find the quote (Q). It attempts an exact match first. If that fails, it applies a fast trigram overlap pre-filter. A trigram is a sequence of 3 characters. The system extracts all trigrams from W and counts how many of Q's trigrams are present within them. This number is divided by the total number of trigrams in Q, producing an overlap ratio from 0 to 1. Windows with an overlap below 0.27 are immediately discarded. Because calculating the precise Levenshtein distance [24] is computationally expensive, this fast trigram check efficiently filters out completely different paragraphs, reserving the true similarity calculation only for highly probable matches.

For a given extracted window W and the LLM-generated quote Q, the normalized similarity is mathematically defined as [29]:

$$Sim(W, Q) = 1 - \frac{Levenshtein(W, Q)}{\max(|W|, |Q|)} \tag{1}$$

For each candidate issue, the system verifies its textual grounding using the maximum similarity score $S = Sim_{max}$ calculated across all candidate document windows. This grounding is processed through a two-tier verification logic:

1. **Discard Threshold ($\tau_{min} = 0.45$):** If $S < \tau_{min}$, the quote is considered non-verifiable (a potential hallucination) and the entire issue is discarded.
2. **Confidence Modulation:** For all surviving issues ($S \geq 0.45$), the initial confidence score C_i is updated to a modulated value C_{new} based on the similarity score, adjusting down for poor matches and slightly boosting for high matches (see Figure 3).
   To illustrate the effect of similarity on the final confidence score, Table 2 provides simple examples. If the similarity is very high ($S \geq 0.70$), the confidence score slightly increases. For example, with an initial confidence of 0.85, a perfect match ($S = 1.0$) boosts it to ~ 0.98. If similarity drops to 0.70, it stays at 0.85. Below 0.70, a penalty logic is activated. For instance, with an initial confidence of 0.85, a similarity of 0.50 reduces the confidence score to ~ 0.65.

Table 2. Examples of confidence score modulation based on similarity (S).

Similarity (S)	**Initial Conf. (C_i)**	**New Conf. (C_{new})**
1.0 (Perfect Match)	0.85	0.98
0.8 (Good Match)	0.85	0.89
0.7 (Neutral Match)	0.85	0.85
0.5 (Poor Match)	0.85	0.65
0.5 (Poor Match)	0.70	0.53

After adjusting the scores, a strict *Confidence Filter* deletes any issue that scores below 0.65. The remaining solid findings go to the *ConsistencyManager*, inspired by the self-consistency principle of cross-checking multiple outputs to

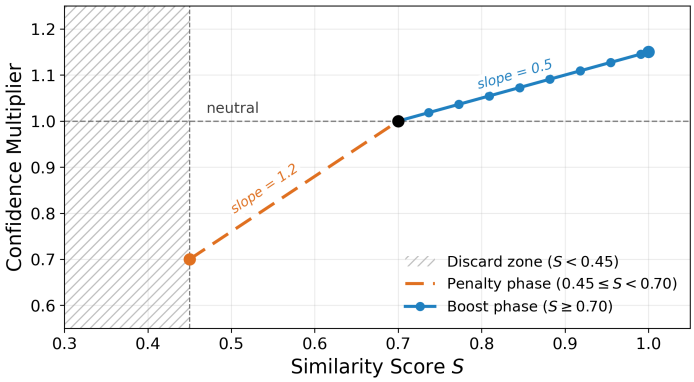

Fig. 3. Confidence modulation function: the multiplier applied to the initial confidence score C_i as a function of the similarity score S.

improve reliability [40]. This is a final AI agent that reads all the critiques, merges repeated ones (e.g. combining multiple repeated issues into a single clear issue), and gives the final approval. Issues are then sorted by severity and renumbered sequentially by category, keeping the feedback focused and useful.

3.4 Report Generation

Input: Verified architectural critiques and extracted document entities.
Output: A formative pedagogical feedback report in PDF format.
The final phase transforms the verified issues into a well-formatted pedagogical PDF report. By relying on deterministic LaTeX templates rather than generating the entire document via LLM, CAPRA ensures that the compilation process is significantly faster, cheaper (due to drastically reduced token usage), and free of formatting errors. The system combines these pre-built structural skeletons with three targeted AI requests, which are delegated to a lightweight, cost-efficient model (`claude-haiku-4.5`). Since these requests produce short natural-language narratives that are injected into pre-validated LaTeX skeletons, rather than raw LaTeX code which remains a challenging task for many LLMs, a smaller model is sufficient for this stage. These requests write short narrative summaries:

1. **Document Context:** Synthesizes a structured overview of the system under analysis, covering its objectives, main use cases, functional and non-functional requirements, architecture, and testing strategy.
2. **Executive Summary:** Provides a concise narrative framing the overall quality of the document, highlighting the patterns of issues found, the critical areas, and the top priority actions for the student.
3. **Strengths:** Identifies the positive aspects of the student's work (e.g., clear structure, good use case coverage, thorough testing), each grounded in concrete evidence from the document.

All other sections, such as the feature coverage checklist, issue details, traceability matrix, and the audit findings themselves, are populated deterministically

using the structured data extracted by the agents. Finally, a compilation service renders the `pdflatex` document, delivering a structured and actionable feedback report to the student.

4 Exploratory Empirical Study Design

To assess the viability of CAPRA as a teaching assistant, we designed our preliminary evaluation to answer the following three Research Questions (RQs):

- **RQ1 (Feedback Quality and Reliability):** *How valid, reliable, and constructive is the feedback generated by CAPRA?*
- **RQ2 (Processing Efficiency):** *How does CAPRA's processing time compare to the manual review effort typically required for software architecture deliverables?*
- **RQ3 (System Customization):** *How well does CAPRA identify the presence or absence of Knowledge Base features within student architectural deliverables?*

4.1 Study Design and Data Collection

To rigorously answer the RQs, our empirical evaluation was structured into three distinct phases, following established guidelines for empirical research in software engineering [21,35].

Phase 1: Experimental Context The study examines project reports from the Software Engineering course within the Bachelor of Science (BSc) Computer Engineering program at the University of Florence, a course that has been running for more than 20 years (more than 5 years in its current form), and typically enrolls approximately 75 students per academic year. We selected a sample of 10 student reports, produced in recent academic years and awarded the highest grades in the course, which served to construct the knowledge base (Figure 2). High-scoring reports were deliberately chosen for this role as they represent well-structured, complete deliverables, allowing the system to extract high-quality, representative features without noise introduced by structural deficiencies. A second, separate sample of 10 student reports was then employed for evaluation purposes. Deliverable documents of the considered course are notably complex and lengthy, typically ranging between 10 and 70 pages. **Phase 2: Data Collection Framework** To systematically assess the output of the CAPRA pipeline, we defined a comprehensive rubric consisting of eight distinct evaluation criteria organized into four dimensions, as detailed in Table 3. These criteria investigated both the extraction capabilities and the qualitative assessment traits of the multi-agent system.

Phase 3: Data Analysis Strategy The feedback generated by CAPRA for each of the 10 reports was independently reviewed by two authors of this article. Both raters evaluated the tool's output for each category in a binary manner (1 for Pass/Valid, 0 for Fail/Invalid). We adopted binary ratings in

Table 3. Evaluation taxonomy: dimensions, criteria IDs, and binary questions used by the two independent raters (Pass = 1, Fail = 0).

Dim.	ID	Binary evaluation question
A Extraction Completeness	A1	Were all functional & non-functional requirements correctly identified and extracted?
	A2	Were all use cases extracted with actors, pre/post-conditions, and alternative flows?
	A3	Were all architectural components, layers, and design patterns correctly identified?
	A4	Were all test types (unit/integration) and coverage metrics correctly extracted?
B Feature Validation	B1	Are the identified Software Engineering (SWE) features real and present, and is each checklist status (Present/Partial/Absent) correctly assigned?
C Issues & Recomm.	C1	Are reported issues real, grounded in source evidence, with no false positives and no missed HIGH-severity problems?
	C2	Are recommendations specific and directly tied to each identified issue?
D Template & Tone	D1	Are all expected sections present and written in formal, objective tone?

this exploratory study to simplify cross-rater comparison and aggregation across criteria, and to keep the evaluation protocol reproducible and easy to calibrate across evaluators. These human ratings serve as the reference against which CAPRA's output quality is interpreted; consequently, inter-rater agreement quantifies the reliability of that reference and provides context for judging how meaningful CAPRA's observed agreement levels are. To scientifically measure the consistency of their judgments, we computed Cohen's Kappa coefficient (κ) [9]. This metric accounts for chance agreement, providing a more reliable measure of consistency than a simple percentage [23].

5 Results

In this section, we present the objective findings from our empirical evaluation, structured according to the research questions.

5.1 Accuracy and Reliability of Architectural Feedback

Overall, CAPRA produced structurally coherent, template-compliant feedback reports for all submissions. The system was particularly strong on *extractive* tasks (e.g., requirements and use-case identification) and on surfacing testing-related anomalies in a consistent manner. Importantly, the Evidence Anchoring mechanism acted as a verification gate: findings that could not be verified against the source text were filtered out, substantially reducing ungrounded critiques.

Pass rates under lenient vs. strict aggregation. Table 4 reports two complementary pass-rate aggregations computed from the two human raters' binary judgments on CAPRA's output. For a given report and criterion, let $r_1, r_2 \in \{0, 1\}$ denote the two binary rater scores. The lenient aggregation

is computed as Avg. $= \frac{r_1+r_2}{2}$, whereas the strict aggregation is defined as Min. $= \min(r_1, r_2)$. Thus, *Avg.* can take values $\{0, 0.5, 1\}$, while *Min.* equals 1 only when both raters assign *Pass*. Under strict aggregation, the overall pass rate is **88.8%**; under lenient aggregation it is **91.9%**.

Inter-rater reliability and class imbalance. Across all 80 aligned rater decisions (8 categories × 10 reports), the two raters reached unanimous agreement on 75 cases, yielding a raw agreement rate of **93.75%**. However, this figure must be interpreted with caution: the evaluation corpus consists exclusively of high-scoring reports, so the majority of decisions are *Pass* across almost every category. This strong class imbalance inflates raw agreement, since both raters are likely to agree on the dominant class regardless of any genuine alignment in judgment. Cohen's Kappa corrects for this chance agreement and is therefore the primary reliability metric we report.

Where the two diverge (A3, C1, D1), genuine inter-rater disagreements occurred, directly mirrored by the κ values. For four categories (A2, A4, B1, C2) all 10 decisions were unanimous *Pass*: κ is undefined due to zero variance (n/a). The global $\kappa = 0.582$ places CAPRA at the upper bound of *moderate agreement* [23] once class imbalance is accounted for.

These figures highlight how different architectural aspects behave under automated evaluation. For extractive tasks like *Requirements Extraction* (A1, $\kappa = 1.00$), both raters consistently agreed, confirming that structured information retrieval is a clear strength of the pipeline. Architecture & Design Patterns (A3, $\kappa = 0.615$) showed moderate agreement, likely reflecting the inherently interpretive nature of assessing architectural descriptions against a rubric. *Grounded Issues* (C1, $\kappa = 0.348$, *fair*) is the most structurally complex criterion: it simultaneously tests for false positives (fabricated issues) and false negatives (missed HIGH-severity problems), two failure modes that individual raters weighted differently. *Template & Tone* (D1, $\kappa = 0.000$) is a well-known statistical artifact: with the first reviewer assigning 9 passes and the second reviewer 10, the single disagreement falls in precisely the configuration where expected and observed agreement coincide, collapsing κ to zero despite 90% raw agreement. This interaction between class imbalance and Kappa is a known limitation of the metric [23] and underscores the importance of reporting raw agreement alongside κ.

Concrete Example of CAPRA Detecting a Semantic Inconsistency
In a ticketing system specification, a specific use case UC-7 (*Ticket Purchase*) is defined with *Guest* as the actor. Yet the Use Case (UC) template includes an alternative flow condition that only makes sense for staff management: *"If the user is not in the event staff: do nothing."*. This introduces a semantic contradiction between the declared actor role and the described behavior, strongly suggesting a copy-paste artifact in the requirements documentation. **Human Rater Action.** While reviewing a long

Table 4. Consolidated results: pass rates under lenient (**Avg.**, arithmetic mean of the two binary ratings) and strict (**Min.**, minimum of the two ratings) aggregation, per-criterion raw agreement, and Cohen's κ. *n/a*: zero variance (all decisions unanimous *Pass*); κ undefined. See Table 3 for criterion definitions.

ID	Criterion	Avg. (%)	Min (%)	Raw agr.	Cohen's κ
A1	Req. Extraction Completeness	90	90	100%	1.000
A2	Use Case Completeness	100	100	100%	*n/a*
A3	Architecture & Design Patterns	85	80	90%	0.615
A4	Test Types & Coverage Metrics	100	100	100%	*n/a*
B1	Feature Validation	100	100	100%	*n/a*
C1	Grounded Issues	65	50	70%	0.348
C2	Actionable Recommendations	100	100	100%	*n/a*
D1	Formal Tone & Template	95	90	90%	0.000
Overall		**91.9**	**88.8**	**93.75%**	**0.582**

n/a: zero variance; κ undefined. Global raw agreement: 93.75% (75/80 decisions).

specification with multiple use cases, the human evaluator overlooked that the alternative flow constraint is incompatible with the UC's actor semantics. **CAPRA Output:**

```
ISS-004 - HIGH [100%] - Page 8
UC-7 (Ticket Purchase) contains an evidently
incorrect and contradictory alternative flow:
‘‘If the user is not
in the event staff: do nothing.’’ ...appears to be a copy-paste
from staff management use cases and conflicts with Actors: Guest.
```

Result. This critique demonstrates CAPRA's effectiveness at identifying UML/UC-level semantic inconsistencies that undermine requirements correctness, highlighting subtle specification defects that are easy to miss in manual reviews.

Concrete Example of CAPRA Detecting an Architectural Gap
In a barber shop project, the Domain Model defines an abstract `User` superclass with `Customer` and `Barber` subclasses. However, the persistence design stores both roles in a single relational table: `Users(email, name, surname, pass_hash, phone, role)`. The report never documents the mapping rule between the OO inheritance hierarchy and the relational schema (e.g., how the `role` field is consistently used to instantiate the correct subclass across all DAOs). **Human Rater Action.** The human evaluator, focusing on the presence of both a UML domain model and a database schema, overlooked that the deliverable does not specify any explicit inheritance-to-table mapping strategy, leaving the cross-layer design rationale implicit and non-verifiable. **CAPRA Output:**

```
ISS-001 - LOW [100%] - Page 36
...Users table stores both roles in a single table
without a discriminator-based mapping strategy being described ...
The document does not clarify how the role field is mapped
to the correct subclass in all DAOs ...
```

Result. This critique demonstrates CAPRA's ability to detect cross-layer architectural gaps where UML-level abstractions are declared but not formally connected to persistence design decisions, weakening architectural traceability and verifiability.

Answer to RQ1 (Highlights). CAPRA's outputs satisfied **88.8%** of criteria under strict aggregation (**91.9%** under lenient aggregation). Human judges agreed on **93.75%** of the 80 evaluation decisions; however, this figure is inflated by the Pass-dominant (high-quality) corpus. After correcting for chance agreement, overall human-assessment reliability is $\kappa = 0.582$ (*moderate*). The most challenging dimension is C1 (*Grounded Issues*), with $\kappa = 0.348$ (*fair*) and a **65%** pass rate under lenient aggregation.

5.2 Processing Efficiency (RQ2)

To evaluate CAPRA's practical viability as a teaching assistant, we measured end-to-end processing times across the 10 evaluated reports.

The average total processing time was slightly over 4 minutes per report (mean = 248s, SD = 73.1s across the 10 reports). Document complexity (page count ranging from 10 to 70 pages) had a moderate impact on processing time, with longer reports requiring up to 6 minutes.

In terms of API cost, processing a single report consumed an average of approximately $0.39 in OpenAI API calls (using `gpt-5.1` pricing) and $0.04 in Anthropic API calls (using `claude-haiku-4.5` pricing), for a total of roughly $0.44 on average for each report after rounding.

The initial system setup requires minimal manual effort: no explicit model training or fine-tuning is needed. Instructors only provide a set of high-quality reports to automatically generate the evaluation Knowledge Base (KB). In our setup, the automatic extraction of the KB from 10 reference reports required processing roughly 786K input tokens and 130K output tokens, incurring a one-time setup cost of approximately $0.20.

Compared to the 30–45 minutes typically required for a thorough manual review by an instructor, CAPRA achieves a speedup factor of approximately 7.2–10.8×. For a class of 30 student groups, this translates from 15–22.5 hours of manual review effort to approximately 2 hours of automated processing, perhaps to be complemented by a concise manual review, enabling instructors to provide high-frequency formative feedback cycles.

Answer to RQ2 (Highlights). CAPRA processes a complete architectural deliverable in slightly over 4 minutes (∼$0.44 per report), achieving a 7.2–10.8× speedup over manual review (30–45 min). The parallelized agent architecture ensures that processing time scales sub-linearly with document complexity, making the system practical for large class sizes.

5.3 System Customization (RQ3)

To evaluate the configurability of CAPRA, we examined how the Knowledge Base (KB) generation pipeline enables automatic adaptation to course-specific rubrics. As described in Section 3.2, the *FeatureCheckAgent* relies on a checklist mined from historical student reports using a SALLMA [3] workflow with HDBSCAN clustering.

From the 10 reference reports used for KB generation, the SALLMA extraction phase identified a total of 680 raw feature mentions across all documents. After HDBSCAN clustering with a minimum cluster size of 3, applying the two-thirds prevalence threshold (i.e., features present in at least 7 out of 10 reports) reduced the validated features to 7, each represented by a canonical description selected from a cluster representative. The resulting checklist is reported in Table 5.

Table 5. Validated Features Extracted via Automated KB Generation

Feature	**Mentions**
Separation of concerns in architecture	85
Unit testing framework implementation	63
UI and interaction design principles	56
Use of UML diagrams for system modeling	36
Data Access Object (DAO) pattern	25
Definition and documentation of use cases	20
Identification and definition of system actors	8
Total	**293**

The effectiveness of this automatically generated KB was assessed by analyzing the *FeatureCheckAgent*'s performance on our evaluation corpus. As reported in Table 4, the agent achieved a 100% pass rate on category B1 (Feature Validation) under the adopted human evaluation protocol, with unanimous agreement between the two human raters. This was determined via manual cross-referencing: raters verified that features flagged as "present" existed in the documents, and confirmed that those flagged as "missing" were absent from the documents. This confirms the agent correctly evaluated feature presence and absence.

The KB generation process is fully automated and requires only a set of reference reports as input, with no manual rubric authoring. However, the resulting checklist can also be manually refined by the instructor, for example, to add domain-specific criteria, remove irrelevant features, or adjust the granularity of individual items, allowing a hybrid approach that combines automated extraction with human revision. An instructor can regenerate the KB at the start of each academic year by providing new reference materials, ensuring that CAPRA remains aligned with evolving course requirements.

Answer to RQ3 (Highlights). CAPRA achieved a 100% pass rate on feature validation (B1) under the adopted human evaluation protocol, with unanimous inter-rater agreement. Across all 10 evaluated reports, both raters judged the presence or absence of all 7 KB-generated features as correctly identified by the system. The KB was automatically generated from 680 raw mentions via HDBSCAN clustering, requiring no manual rubric authoring.

6 Discussion

The results presented in Section 5 paint a nuanced picture of the capabilities and current limitations of LLM-based multi-agent systems for educational assessment. In this section, we interpret the key findings and discuss their implications.

Interpretation of Key Results. Overall, CAPRA generates valid architectural feedback (88.8% of criteria under strict aggregation), but its reliability varies by dimension. CAPRA results are most dependable on well-defined extractive tasks, where presence/absence rests on objective evidence, and least so on interpretive criteria (A3, C1), where the lower inter-rater agreement reflects genuine subjectivity among raters rather than a pipeline failure (D1's $\kappa = 0$ being a known class-imbalance artifact; see Section 5.1). The highest pass rates fall on evidence-bound criteria such as actionable recommendations (C2), confirming that deterministic grounding is an effective way to curb LLM hallucinations in educational settings.

Implications for Educators. Our results suggest that CAPRA is best deployed as a *first-pass teaching assistant* rather than a standalone grader. The 7.2–10.8× speedup enables instructors to offer rapid, formative feedback cycles, e.g., providing preliminary comments within hours of submission rather than weeks. This also makes CAPRA suitable for intermediate evaluations of work in progress before the final submission. In this role, CAPRA can surface likely issues early, while human review concentrates on dimensions that require greater interpretive judgment, particularly *Architecture & Design Patterns* (A3) and *Grounded Issues* (C1), where the evaluation task showed lower agreement among human raters. A practical integration strategy would involve CAPRA generating initial feedback, which instructors then review and refine, focusing their limited time on the most nuanced aspects.

Limitations. Several practical limitations should be noted. First, all student reports in our evaluation were written either in Italian or in English, while CAPRA's prompts and agent instructions are authored in English. Although `gpt-5.1` supports multilingual processing, we observed a concrete instance of language contamination that directly contributed to the single failure in the *Template & Tone* dimension (D1): one rater identified that a generated feedback report mixed Italian and English across different sections. Despite all agents being explicitly prompted to respond in English, prolonged exposure to a large Italian-language input document appears to have caused the model to drift toward the source language in isolated sections. While this was an isolated case, it highlights a subtle but real risk in cross-lingual deployments.

Second, calibrating the scope and granularity of generated issues proved non-trivial. We observed a tension between overly generic feedback and production-oriented issues outside the pedagogical scope of an undergraduate course (e.g., security concerns or database transaction strategies). Resolving this currently requires iterative prompt refinement, undermining the plug-and-play usability of the system. A key open challenge is therefore enabling instructors to control feedback scope through high-level configuration rather than direct prompt engineering.

Third, the system currently depends on OpenAI's proprietary `gpt-5.1` model, introducing both a cost dependency (∼$0.44/report) and a reproducibility constraint, as model behavior may change across API versions. Finally, the deterministic seed and temperature-zero settings mitigate but do not fully eliminate non-determinism in LLM outputs, as acknowledged by OpenAI's documentation.

Comparison with Existing Approaches. Unlike ArTEMiS [22], which provides automated feedback primarily for code-based assignments through static analysis, CAPRA targets the complementary and underserved domain of open-ended architectural documentation. Compared to single-LLM approaches for essay and diagram grading [4,42], CAPRA's multi-agent decomposition with evidence anchoring provides two structural advantages: (i) specialized agents help decompose the task into more focused checks; and (ii) the deterministic anchoring layer provides a verifiable trust boundary absent in end-to-end LLM evaluations.

7 Threats to Validity

We discuss the threats to validity of our study following established categorizations [35].

Construct Validity. The evaluation rubric (categories A1–D1) was designed by the authors to capture the key quality dimensions of architectural deliverables. This rubric may not cover all relevant aspects of software architecture assessment, potentially causing relevant quality dimensions to go unmeasured and leading to an incomplete picture of system performance. Furthermore, the binary Pass/Fail evaluation scale sacrifices granularity compared to ordinal or continuous scales, which may obscure partial correctness by forcing borderline cases into fully positive or fully negative judgments. This discretization may in turn inflate or deflate pass rates, and can introduce both false positives and false negatives at the decision boundary. As a mitigation, the rubric was grounded in established software architecture quality criteria and refined through a pilot evaluation. Future work could adopt finer-grained rubrics (e.g., Likert scales) to capture partial compliance more accurately.

Internal Validity. Both raters were authors of this paper, which introduces a potential evaluator bias toward favorable assessments of the system. This could have artificially inflated pass rates and inter-rater agreement. To mitigate this risk, the two raters performed their evaluations independently, without prior discussion, and Cohen's Kappa was adopted as the primary reliability metric to account for chance agreement. An external replication with independent raters would further strengthen confidence in the results.

Additionally, the Evidence Anchoring thresholds ($\tau_{min} = 0.45$, confidence filter at 0.65, trigram overlap at 0.27) were determined empirically through preliminary experiments. Different threshold configurations could yield different filtering behaviors, potentially retaining more hallucinated findings or discarding valid ones, thus directly affecting the pass rates reported for criteria such as C1. A systematic sensitivity analysis of these parameters is left as future work, as the

current dataset size does not provide sufficient statistical power to derive robust optimal values.

Furthermore, the exclusive use of top-scoring reports introduces an additional threat to internal validity. This selection bias may have inflated pass rates and limited the discriminative power of the evaluation, as the system is never challenged with severely deficient deliverables, where hallucination rates and false positives could be substantially higher. This choice represents a deliberate trade-off between exploratory evaluation feasibility and internal validity: low-scoring reports, while more representative of the full student population, tend to contain pervasive structural deficiencies that would make it significantly harder to conduct a focused assessment of subtle architectural shortcomings in this study. Extending the evaluation to the full spectrum of student performance levels is planned as future work.

External Validity. Our evaluation was conducted on 10 reports from a single software engineering course at one university, limiting generalization to other institutional contexts, course structures, and grading cultures. This could mean that the system's performance, and in particular the relevance of the automatically generated Knowledge Base features, may not transfer to courses with different architectural conventions or documentation standards. Furthermore, the system was evaluated using a single LLM configuration (`gpt-5.1` for analysis and `claude-haiku-4.5` for LaTeX generation); performance may vary with different models, particularly open-source alternatives with potentially lower reasoning capabilities, which could reduce feedback quality and Evidence Anchoring accuracy. Replication across multiple institutions, languages, student performance levels, and model configurations is needed to establish broader validity, and is planned as a primary direction for future work.

8 Replication Package

To support reproducibility, a replication package is publicly available on Zenodo at DOI: 10.5281/zenodo.20629900. It contains: (i) a self-contained, Java-based re-implementation of the CAPRA pipeline on the Spring AI framework [38], with the Python microservices for document parsing and multi-modal extraction; (ii) the CSV export of the SWE features mined via the SALLMA workflow and HDBSCAN clustering for Knowledge Base generation; (iii) the 10 CAPRA-generated PDF feedback reports, one per evaluated team (Teams 06, 12–20); (iv) the full annotation spreadsheet with both raters' binary scores across the 8 criteria and 10 teams plus the evaluation taxonomy (Table 3), used for all inter-rater statistics in Section 5.1; and (v) one original student deliverable used as input (shared with explicit consent), the remaining nine being withheld to protect student privacy, as public-release consent was not obtained.

9 Conclusion

This paper presented CAPRA, a multi-agent LLM system that automates formative feedback on software architecture deliverables. By orchestrating multiple specialized agents through a four-stage pipeline, namely Document Parsing, Parallel Verification, Evidence Anchoring, and Report Generation, CAPRA addresses the scalability bottleneck of manual review in software engineering education.

Our preliminary empirical viability assessment on 10 student reports demonstrated that CAPRA achieves moderate inter-rater agreement with human evaluators ($\kappa = 0.582$) and satisfies 88.8% of the evaluated criteria under strict aggregation, while processing each report in slightly over 4 minutes compared to the 30–45 minutes required for manual review. The Evidence Anchoring mechanism proved particularly effective in grounding LLM findings and mitigating hallucinations, with category C2 (Actionable Recommendations) achieving a 100% pass rate under the adopted evaluation protocol. Meanwhile, the automated Knowledge Base generation pipeline enabled course-specific customization without manual rubric authoring.

Future Work. Several directions emerge from this study. First, we plan to extend the evaluation to a larger and more diverse dataset spanning multiple universities, languages, and student performance levels. Second, we intend to investigate open-source LLM alternatives to reduce API cost dependencies and improve reproducibility. Third, integration with established learning management systems such as Moodle and ArTEMiS [22] would enable seamless adoption in real classroom workflows. Finally, since this work focused strictly on the technical quality and reliability of the system from a teacher's perspective, our next step is to directly assess its educational usefulness with students to understand how this automated feedback impacts their learning experience and project outcomes.

References

1. Ala-Mutka, K.M.: A survey of automated assessment approaches for programming assignments. Computer Science Education **15**(2), 83–102 (2005)
2. Artifex Software, Inc.: PyMuPDF: Python bindings for MuPDF (2024), https://pymupdf.readthedocs.io, version 1.24, accessed 2024
3. Becattini, M., Verdecchia, R., Vicario, E.: SALLMA: A software architecture for LLM-based multi-agent systems. In: Proc. IEEE/ACM International Workshop New Trends in Software Architecture (SATrends). pp. 5–8 (2025). https://doi.org/10.1109/SATrends66715.2025.00006
4. Bouali, N., Gerhold, M., Rehman, T.U., Ahmed, F.: Toward automated UML diagram assessment: Comparing LLM-generated scores with teaching assistants. In: Proceedings of the 17th International Conference on Computer Supported Education, CSEDU 2025. vol. 1, pp. 158–169 (2025). https://doi.org/10.5220/0013481900003932
5. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., et al.: Language models are few-shot learners. In: Advances in NeurIPS. vol. 33, pp. 1877–1901 (2020)

6. Campello, R.J.G.B., Moulavi, D., Sander, J.: Density-based clustering based on hierarchical density estimates. In: Advances in Knowledge Discovery and Data Mining – PAKDD 2013. Lecture Notes in Computer Science, vol. 7819, pp. 160–172. Springer (2013). https://doi.org/10.1007/978-3-642-37456-2_14
7. Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.M., Yu, H., Lu, Y., Hung, Y.H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., Zhou, J.: AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In: Proc. ICLR (2024)
8. Chu, Z., Wang, S., Xie, J., Zhu, T., Yan, Y., Ye, J., Zhong, A., Hu, X., Liang, J., Yu, P.S., Wen, Q.: LLM agents for education: Advances and applications. In: Christodoulopoulos, C., Chakraborty, T., Rose, C., Peng, V. (eds.) Findings of the Association for Computational Linguistics: EMNLP 2025. pp. 13782–13810. Association for Computational Linguistics, Suzhou, China (Nov 2025). https://doi.org/10.18653/v1/2025.findings-emnlp.743, https://aclanthology.org/2025.findings-emnlp.743/
9. Cohen, J.: A coefficient of agreement for nominal scales. Educational and Psychological Measurement **20**(1), 37–46 (1960)
10. Emirtekin, E.: Large language model-powered automated assessment: A systematic review. Applied Sciences **15**(10), 5683 (2025). https://doi.org/10.3390/app15105683, https://www.mdpi.com/2076-3417/15/10/5683
11. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges. In: Larson, K. (ed.) Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24. pp. 8048–8057. International Joint Conferences on Artificial Intelligence Organization (8 2024). https://doi.org/10.24963/ijcai.2024/890, https://doi.org/10.24963/ijcai.2024/890, survey Track
12. Hattie, J., Timperley, H.: The power of feedback. Review of Educational Research **77**(1), 81–112 (2007)
13. Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S.K.S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., Schmidhuber, J.: MetaGPT: Meta programming for a multi-agent collaborative framework. In: Proc. ICLR (2024)
14. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J., Wang, H.: Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology **33**(8), 1–79 (2024). https://doi.org/10.1145/3695988, article 220
15. Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., Liu, T.: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Trans. Inf. Syst. **43**(2), 1–55 (Jan 2025). https://doi.org/10.1145/3703155, https://doi.org/10.1145/3703155
16. Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O.: Review of recent systems for automatic assessment of programming assignments. In: Proc. Koli Calling. pp. 86–93 (2010)
17. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12), 1–38 (Mar 2023). https://doi.org/10.1145/3571730, https://doi.org/10.1145/3571730
18. Kasneci, E., Seßler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T.,

Stadler, M., Weller, J., Kuhn, J., Kasneci, G.: ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences **103**, 102274 (2023)
19. Keuning, H., Jeuring, J., Heeren, B.: A systematic literature review of automated feedback generation for programming exercises. ACM Trans. Comput. Educ. **19**(1), 1–43 (2019)
20. Kim, S., Shin, J., Cho, Y., Jang, J., Longpre, S., Lee, H., Yun, S., Shin, S., Kim, S., Thorne, J., Seo, M.: Prometheus: Inducing fine-grained evaluation capability in language models. In: Proc. ICLR (2024)
21. Kitchenham, B.A., Pfleeger, S.L., Pickard, L., Jones, P.W., Hoaglin, D.C., Emam, K.E., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. **28**(8), 721–734 (2002)
22. Krusche, S., Seitz, A.: Artemis: An automatic assessment management system for interactive learning. In: Proc. SIGCSE. pp. 284–289 (Feb 2018). https://doi.org/10.1145/3159450.3159602
23. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics **33**(1), 159–174 (1977)
24. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**, 707–710 (1966)
25. Li, G., Hammoud, H.A.A.K., Itani, H., Khizbullin, D., Ghanem, B.: CAMEL: Communicative agents for "mind" exploration of large language model society. In: Advances in NeurIPS. vol. 36 (2023)
26. Li, X., Wang, S., Zeng, S., Wu, Y., Yang, Y.: A survey on LLM-based multi-agent systems: Workflow, infrastructure, and challenges. Vicinagearth **1**(1) (2024). https://doi.org/10.1007/s44336-024-00009-2
27. Li, Y., Yang, G., Liu, H., Wang, B., Zhang, C.: dots.ocr: Multilingual document layout parsing in a single vision-language model. arXiv preprint arXiv:2512.02498 (2025)
28. Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., Zhu, C.: G-Eval: NLG evaluation using GPT-4 with better human alignment. In: Proc. EMNLP (2023)
29. Marzal, A., Vidal, E.: Computation of normalized edit distance and applications. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(9), 926–932 (1993). https://doi.org/10.1109/34.232078
30. Messer, M., Brown, N.C.C., Kolling, M., Shi, M.: Automated grading and feedback tools for programming education: A systematic review. ACM Trans. Comput. Educ. **24**(1), 1–43 (2024). https://doi.org/10.1145/3636515
31. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (Mar 2001). https://doi.org/10.1145/375360.375365, https://doi.org/10.1145/375360.375365
32. Phung, T., Cambronero, J., Gulwani, S., Kohn, T., Majumdar, R., Singla, A., Soares, G.: Generating high-precision feedback for programming syntax errors using large language models. In: Proc. EDM (2023)
33. Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., Sun, M.: ChatDev: Communicative agents for software development. In: Proc. ACL (2024)
34. Radermacher, A., Walia, G.: Gaps between industry expectations and the abilities of graduates. In: Proc. SIGCSE. pp. 525–530 (2013)
35. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009)
36. Singh, R., Gulwani, S., Solar-Lezama, A.: Automated feedback generation for introductory programming assignments. In: Proc. PLDI. pp. 15–26 (2013)

37. Tenhunen, S., Männistö, T., Luukkainen, M., Ihantola, P.: A systematic literature review of capstone courses in software engineering. Inf. Softw. Technol. **159**(C), 107191 (Jul 2023). https://doi.org/10.1016/j.infsof.2023.107191, https://doi.org/10.1016/j.infsof.2023.107191
38. VMware, Inc. and Spring Contributors: Spring AI: An application framework for AI engineering. https://spring.io/projects/spring-ai (2024), version 1.0.x. Accessed: February 2026
39. Wang, P., Li, L., Chen, L., Cai, Z., Zhu, D., Lin, B., Cao, Y., Kong, L., Liu, Q., Liu, T., Sui, Z.: Large language models are not fair evaluators. In: Proc. ACL (2024)
40. Wang, X., Wei, J., Schuurmans, D., Le, Q.V., Chi, E.H., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models. In: Proc. ICLR (2023)
41. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A.H., White, R.W., Burger, D., Wang, C.: AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv preprint arXiv:2308.08155 (2023)
42. Xie, W., Niu, J., Xue, C.J., Guan, N.: Grade like a human: Rethinking automated assessment with large language models. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems. pp. 1–8. RACS '25, Association for Computing Machinery, New York, NY, USA (2026). https://doi.org/10.1145/3769002.3769962, https://doi.org/10.1145/3769002.3769962
43. Yujian, L., Bo, L.: A normalized levenshtein distance metric. IEEE Transactions on Pattern Analysis and Machine Intelligence **29**(6), 1091–1095 (2007). https://doi.org/10.1109/TPAMI.2007.1078
44. Zheng, L., Chiang, W.L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E.P., Zhang, H., Gonzalez, J.E., Stoica, I.: Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In: Advances in NeurIPS. vol. 36 (2023)