IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

# Code Generation and Debugging Assistant using Deep Learning and Large Language Models

**Nikhitha H S[1], Priyarani Suramanji[2], Dr Indumathi S K[3]**

Department of MCA, Dr. Ambedkar Institute of Technology, Bangalore – 560056, Karnataka, India[1-3]

**Abstract:** Novice programmers frequently struggle to translate cryptic compiler diagnostics into actionable understanding, while contemporary generative Artificial Intelligence (AI) assistants optimized for professional productivity tend to resolve reported errors by supplying complete corrected code immediately. This mode of instant resolution, though efficient for experienced developers, is pedagogically counterproductive for learners, as it discourages active engagement with the debugging process. This paper presents CodeMentor AI, a full-stack hybrid web application combining a locally hosted statistical Machine Learning (ML) pipeline with a cloud based Large Language Model (LLM) to deliver both rapid error triage and structured, Socratic-style tutoring. The FastAPI backend integrates a TF-IDF vectorizer with a Logistic Regression classifier for baseline error category prediction, and a cosine-similarity Nearest Neighbors model for code retrieval, with the Groq-hosted openai/gpt oss 120b LLM used for explanation, progressive hinting, and solution verification.The React frontend provides an integrated Monaco-based workspace, a three-tier Learning Mode hint system, and an administrative analytics dashboard with Firebase Authentication and a Supabase (PostgreSQL) backend for persistence. This paper documents the system's architecture, module design, and AI pipeline as implemented, and situates it within recent literature on AI-assisted programming education, retrieval-augmented debugging, and comparative studies of commercial AI coding tools. Formal empirical evaluation of learning outcomes and system accuracy has not yet been conducted and is identified as future work.

**Keywords:** Artificial Intelligence in Education, Large Language Models, Retrieval Augmented Generation, Programming Tutoring Systems, Automated Debugging, Hybrid Machine Learning, FastAPI, React, Supabase, Socratic Scaffolding

## 1. INTRODUCTION

Debugging occupies a disproportionate share of a novice programmer's effort, since compiler and interpreter diagnostics are designed for practitioners who already possess a working model of the language's execution semantics [2], [6]. The emergence of LLMs capable of generating, explaining, and correcting code has been proposed as a remedy [5],[10], but general purpose assistants such as ChatGPT and GitHub Copilot are built to maximize developer throughput: presented with buggy code, they typically return a corrected version immediately [6], [10]. For a professional this is a productivity gain for a student it risks substituting passive code copying for the active reasoning that produces durable understanding [6], [7].

CodeMentor AI responds to this tension. Rather than routing every request directly to a cloud LLM, its FastAPI backend first classifies buggy code using a locally trained TF-IDF + Logistic Regression model and retrieves similar code templates for generation requests using a TF-IDF + cosine-similarity Nearest Neighbors model. These low latency local predictions condition prompts sent to the Groq hosted LLM, which performs the higher value reasoning: producing explanations, generating/correcting code, formulating guiding questions, issuing progressive hints, and verifying whether a students own edit resolves a bug. Two operational modes are exposed a direct “Quick Fix” flow, and a multi step “Mentor” flow that withholds the corrected code and scaffolds the student toward independently locating and fixing the defect.

**Problem Statement.** Native error messages are diagnostically inadequate for beginners instant resolution AI tools short circuit active reasoning, and few systems in the literature combine authenticated sessions, persistent history, administrative analytics, and a cost-aware hybrid AI pipeline in one deployable application[6],[7].

**Objectives.** (1) Implement a hybrid ML-triage + LLM-reasoning backend; (2) implement a Socratic Mentor flow with progressive hints and LLM-based fix verification; (3) provide a unified Monaco based workspace for debugging, code generation, and AI assisted chat; (4) implement authenticated, persistent sessions via Firebase and Supabase; (5) provide an admin panel with usage analytics and user management.

  

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

**Scope and Limitations.**Verified against the source code, the implemented feature set covers AI-assisted debugging (/debug-explain), code generation (/generate), a Code Companion Chat, the Mentor flow, unified history, and admin analytics and user management. Several items are explicitly not fully implemented: session rename/delete persist only to browser LocalStorage, not Supabase; admin route access is gated only client-side (a localStorage flag) with no server-side authorization on the /admin/* endpoints themselves; admin status is granted via a single hardcoded email allowlist; and the local debugging classifier is trained on a dataset skewed 95% toward C++ with only three coarse error classes (Runtime Error, Time Limit Exceeded, Memory Limit Exceeded), despite Python and JavaScript being offered in the UI. These gaps are reported transparently rather than omitted, consistent with treating the implementation as the sole source of truth.

## 2. LITERATURE REVIEW

Recent research in programming education has increasingly focused on AI-assisted debugging, intelligent tutoring systems, retrieval-augmented generation, and large language models for code generation and automated error correction.

**Kathiroli et al.** introduce a Rubber Duck Assistant combining Retrieval Augmented Generation with dialogue-driven Python debugging, reporting ~92% accuracy and 91% F1 in query classification against a ChromaDB-backed retrieval store closely paralleling CodeMentor AI's Learning Mode, though scoped to Python only [1].

**Shreeya et al.** present CodeGuide, a VS Code extension using a fine-tuned, offline LLaMA 3.2 model for Python generation, error detection, and correction, comparing LoRA/QLoRA/Adaptive-Fusion fine-tuning strategies (~90–92% accuracy) demonstrating that fine-tuned local models can approach cloud-LLM correction quality at lower cost, a possible future direction for CodeMentor AI's own local layer [2].

**Chandu et al.** move away from text-only retrieval toward a voice-driven programming assistant, reporting ~93% voice-recognition accuracy and ~90% debugging success a considerably more extensive voice feature set than CodeMentor AI's current single-page dictation control [3]

**Uyyala et al.** return to retrieval-grounded architecture with a RAG and FAISS backed terminal assistant for command generation, summarization, and debugging, architecturally similar in spirit to CodeMentor AI's retrieval stage but targeting CLI rather than tutoring workflows [4].

**Vethika et al.** merge an LLM (LLaMA 3) with document retrieval for context-grounded code generation, conceptually mirroring CodeMentor AI's own retrieve then generate /generate pipeline [5].

**Naman et al.** shift the focus from system architecture to learner behaviour, analyzing 126 students' interaction with ChatGPT and Copilot over a 13-week software engineering project, finding that direct code generation access can crowd out active problem-solving unless deliberately structured direct empirical support for CodeMentor AI's decision to withhold full corrections in Learning Mode [6].

**Ouaazki et al.** ask a related question, evaluating ChatGPT as a Computational Thinking learning assistant and examining whether LLM augmented learning widens or narrows the gap between stronger and weaker students directly relevant to the Mentor mode's scaffolding goal [7].

**Elhayany and Meinel** stay with the theme of adaptive pedagogy in ARIA, a FAISS/GPT-backed system that dynamically generates retrieval-practice quiz items for introductory programming, validated through a pilot with eight educators — paralleling CodeMentor AI's runtime generation of guiding questions rather than static hint banks [8].

**Cotroneo et al.** step back from individual classrooms to a 500,000-sample study comparing human-written and AI-generated code for defects, vulnerabilities, and complexity, finding AI code tends toward simpler, more repetitive structure — substantiating CodeMentor AI's prompt-level instruction to preserve a student's original algorithmic structure during correction rather than silently rewriting it [9].

**Čerňanský et al.** close the picture out with a broader survey of the LLM programming-tool landscape (ChatGPT, Copilot, Cursor) for CS curricula, providing a map against which CodeMentor AI's Socratic positioning can be contrasted [10].

  

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

Table I . Comparative Analysis of Existing Studies

| Study | Methodology | AI Technique | Key Contribution | Limitation |
|---|---|---|---|---|
| Kathiroli et al. | RAG + hybrid classifier | ChromaDB retrieval + LLM | Dialogue-driven Python tutor | Python-only |
| Shreeya et al. | Fine-tuning comparison | LoRA/QLoRA local LLM | Offline, cost-efficient correction | Single-language, IDE-scoped |
| Chandu et al. | System + accuracy eval | Speech + NLP | Hands-free coding/debugging | Broader scope than typical tutors |
| Uyyala et al. | RAG + FAISS | Retrieval-Augmented Gen. | NL-to-terminal-command translation | CLI-focused, not tutoring |
| Vethika et al. | RAG-grounded generation | LLaMA 3 + retrieval | Domain-grounded code synthesis | Limited evaluation detail |
| Naman et al. | 13-week classroom study | ChatGPT, Copilot | Evidence on student-LLM interaction | Observational only |
| Ouaazki et al. | Field study, UX survey | ChatGPT | CT learning equity impact | Not debugging-specific |
| Elhayany & Meinel | Pilot study (8 educators) | GPT + FAISS | Adaptive retrieval-practice quizzes | Small pilot; not debugging |
| Cotroneo et al. | Large-scale empirical study | ChatGPT/DeepSeek/Qwen | Quantifies AI-code defect/complexity | Not tutoring-focused |
| Čerňanský et al. | Literature/tool survey | Survey of LLM tools | Curriculum-oriented tool map | No novel system |

**Research Gap.** Across these studies, retrieval grounding is a common architectural default, and empirical scrutiny of AI-code quality is maturing, but three gaps remain (i) few systems combine authenticated persistence, history, and administrative analytics in one deployable platform; (ii) explicit multi-level progressive hinting with LLM-based fix verification is rare only the Rubber Duck Assistant's dialogue approach partially overlaps; (iii) most generation/debugging tools rely on a single AI technique rather than a cost-aware local-triage-plus-cloud-LLM split. CodeMentor AI is positioned to address all three, while itself inheriting a data-composition limitation (Section 1) that none of the reviewed papers directly address.

## 3. PROPOSED SYSTEM AND ARCHITECTURE

CodeMentor AI follows a layered client-server architecture: a React 19 + Vite frontend (Monaco Editor, React Router, Axios, Recharts) communicates with Firebase Authentication (Google Sign-In and email/password) directly for identity, and with a FastAPI backend over REST for all AI, ML, and persistence operations. The backend fans out to three services: a local ML layer (predict.py: TF-IDF + Logistic Regression for debugging, TF-IDF + Nearest Neighbors for generation), the Groq-hosted LLM (groq_service.py, model openai/gpt-oss-120b, note: the project's own README states “LLaMA 3.3,” which does not match the code), and a Supabase PostgreSQL client persisting users, debug_history, and generation_history tables. Outputs from all three are merged into a single JSON response per request.

**Quick Fix workflow:** submitted code is classified locally (predict_debug), then passed with the ML label to explain_debug_result(), which prompts the LLM for a refined error type, severity, corrected code, and a five-part structured explanation; a regex-based Python-only heuristic (basic_auto_fix) serves as a last-resort fallback if the LLM call fails or is a no-op.

**Mentor (Learning Mode) workflow:** generate_mentor_question() produces a non-revealing guiding question; generate_mentor_hint() returns one of three progressively specific hints (proximity → conceptual → near-solution); verify_user_fix() compares the student's edited code to the original logically, via the LLM, rather than by string equality.

**Generation workflow:** a Nearest Neighbors retrieval step (k=1, cosine similarity) proposes a candidate snippet from a local dataset; the LLM then synthesizes the final solution, with the retrieved snippet returned transparently as a fallback. Other confirmed modules include a Code Companion Chat (/chat), which supports follow-up conversational queries with up to six turns of history, a unified History module (/user/history) that merges debugging and code generation records in chronological order, and an Admin module providing aggregate statistics, user management (CRUD operations), and

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

global history views. The admin routes are protected only through client-side route gating, which is recognized as a security limitation rather than a recommended deployment practice.

**Key Benefits**

- Reduced LLM cost and latency: local TF-IDF/Nearest Neighbors triage adds negligible latency before the metered Groq API call.
- Explicit pedagogical scaffolding: guiding question, three-level hints, and LLM-based verification give a structured path to independent problem-solving.
- Persistent, authenticated session history across devices via Firebase and Supabase.
- Consolidated workspace: debugging, code generation, and AI-assisted chat integrated into a single Monaco-based interface, reducing context switching.

Table II. Feature comparison

| Feature | Compilers/IDEs | Auto-Graders | General AI Chatbots | CodeMentor AI |
|---|---|---|---|---|
| Feedback style | Raw diagnostics | Pass/fail + logs | Direct corrected code | Structured explanation or guided hints |
| Educational focus | Low | Medium | Low | High (explicit tutoring flow) |
| Cost pattern | None | Sandbox execution | Every request billed to LLM | Local ML triage first, LLM for reasoning only |
| Admin analytics | None | Gradebook | None | Present (client-gated) |

**Comparative Study.** Against standard IDEs and compilers, CodeMentor AI's principal advantage is interpretive: raw diagnostics identify a syntactic or runtime violation but do not explain the learner's likely misconception, whereas the system's structured five-part explanation and Mentor-mode guiding questions address the conceptual gap directly. Against automated grading platforms, CodeMentor AI trades exhaustive test-suite verification for interpretive depth, offering reasoning about why a submission fails rather than only whether it passes. Against general-purpose AI chatbots such as ChatGPT and GitHub Copilot, the most direct competitors architecturally, CodeMentor AI's distinguishing choice is procedural rather than purely technical: the same underlying LLM capability is deliberately withheld from producing an immediate answer in Learning Mode, and is instead directed toward generating questions and graduated hints, a workflow general-purpose assistants do not enforce by default.

## 4. MODULE DESIGN AND DATABASE

Table III. Module Design

| # | Module | Purpose |
|---|---|---|
| 1 | Authentication | Google and email/password sign-up/sign-in, password reset, admin-flag assignment |
| 2 | Dashboard / AI Dashboard | Landing view and navigation hub; client-side streak computation from session timestamps |
| 3 | Unified Workspace | Monaco code editor hosting debugging, code generation, and AI chat in a unified workspace |
| 4 | Code Debugging (Quick Fix + Mentor) | Hybrid ML+LLM debugging and three-level Socratic tutoring flow |
| 5 | Code Generation | Hybrid ML nearest-neighbor retrieval + LLM code synthesis |
| 6 | Code Companion Chat | Follow-up conversational Q&A grounded in the current workspace code |
| 7 | User History | Unified, chronologically merged debug + generation history |

  

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

| | | |
|---|---|---|
| 8 | Profile | User profile view and local customization |
| 9 | Admin Dashboard / Analytics | Aggregate platform statistics with chart-based visualization |
| 10 | Admin User Management | Create, read, update, and delete operations on registered users |
| 11 | Admin History Views | Global (all-user) debug and generation history tables |
| 12 | Route Protection | Client-side gating of admin routes via a localStorage flag |
| 13 | Database Layer | Supabase (PostgreSQL) table access for users and history |
| 14 | Local ML / API Layer | TF-IDF, Logistic Regression, and Nearest Neighbors inference |
| 15 | Groq AI Integration | Prompt construction, JSON-contract enforcement, all LLM calls |

**Database Design.** Supabase (managed PostgreSQL) is the system's sole persistent store, with three tables: users (firebase_uid, name, email), debug_history (firebase_uid, code, predicted_error, explanation, corrected_code, language), and generation_history (firebase_uid, prompt, generated_code, language). No formal foreign-key constraints link the history tables to users; the relationship is enforced only at the application layer by matching on the authenticated user's email, and the admin user-update/delete endpoints manually cascade changes across all three tables in Python code rather than relying on a database-level cascade rule.

**AI Request Lifecycle**

- 1. User submits code, a prompt, or a question from the React frontend.
- 2. The frontend packages the request as an Axios POST call to the FastAPI backend.
- 3. The route handler validates the payload against its Pydantic model.
- 4. For debugging and generation requests only, the local ML layer (predict.py) produces a baseline prediction or a retrieved candidate snippet.
- 5. groq_service.py constructs the task-specific system and user prompts, incorporating the ML output where applicable.
- 6. The Groq Chat Completions API is called against the openai/gpt-oss-120b model.
- 7. extract_json() parses the model's JSON response, recovering from minor formatting deviations.
- 8. If the LLM call or JSON parsing fails, a hardcoded fallback response is substituted.
- 9. The result is written to the relevant Supabase history table on a best-effort basis; failures are logged but do not block the response.
- 10. The final JSON result is returned to the frontend and rendered in the workspace UI.

**Applications**

- University computer science laboratories, where students obtain immediate, scaffolded feedback on compilation and logical errors without exclusive reliance on teaching-assistant availability.
- Coding boot camps and self-paced online courses, where the Mentor flow's progressive hinting can substitute for continuous human supervision outside scheduled class hours.
- Individual self-directed learners, who can use the workspace history and client-computed activity streak to build consistent practice habits.
- Instructors and teaching staff, who can use the Admin Analytics dashboard to review aggregate debug/generation volume and per-user activity.

## 5. IMPLEMENTATION, TESTING, AND RESULTS

The back end uses Fast-API 0.127.0, sci-kit-learn 1.8.0, and pandas/numpy, with Pydantic request validation and permissive CORS suitable only for local development. The local models are trained offline (train_model.py): a TfidfVectorizer + Logistic Regression (max_iter=1000, 80/20 split) for debugging, and a separate TfidfVectorizer + Nearest Neighbors (cosine, k=1) for generation, both serialized via pickle. Direct inspection of the shipped datasets confirms 41,905 debugging rows (39,931 C++ vs. 1,974 Python, three error classes only) and 86,327 generation rows. Every LLM call in groq_service.py follows a consistent system/user prompt pattern requesting strict JSON, parsed defensively by an extract_json() helper that recovers from minor formatting deviations; every Supabase write and every Groq call is wrapped to fail gracefully rather than interrupt the user-facing response.

  

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

Table IV. Technology Stack

| Layer | Technology |
|---|---|
| Frontend Framework | React 19 + Vite |
| Code Editor | Monaco Editor (@monaco-editor/react) |
| Routing / HTTP | React Router v7, Axios |
| Charts | Recharts |
| Auth SDK | Firebase Authentication (Google + email/password) |
| Backend Framework | FastAPI 0.127.0 (uvicorn 0.40.0) |
| ML Libraries | scikit-learn 1.8.0, pandas 2.3.3, numpy 2.3.1, joblib |
| Validation | Pydantic 2.12.5 |
| LLM Provider | Groq Cloud SDK (openai/gpt-oss-120b) |
| Database | Supabase (managed PostgreSQL) |

**Testing.** No automated test suite, CI pipeline, or frontend test files exist in the codebase at present only a handful of ad hoc manual scripts used during development to sanity check individual endpoints. Any performance or accuracy claims made about CodeMentor AI going forward should therefore rest on dedicated experimental testing rather than on development-time observation alone.

**Results.** The developed system was functionally validated through manual testing of authentication, debugging, code generation, mentor mode, history management, and administrative modules.

## 6. DISCUSSION, FUTURE SCOPE, AND CONCLUSION

CodeMentor AI's two-stage local-classifier-then-LLM design and its explicit three-level Mentor hint flow with LLM-based verification are comparatively uncommon among the reviewed literature, most of which relies on a single AI technique [4],[7],[9] or studies existing commercial tools rather than deploying a novel scaffolded system [6],[10]. At the same time, the training-data imbalance (Section 4) means the local classifier's practical value is presently concentrated on C++ runtime/performance errors rather than the Python syntax errors the system's pedagogy emphasizes, and the admin security model requires hardening before any deployment beyond a trusted local or demonstration context.

### Future Work

- Rebalance and expand the local training dataset with a finer-grained, Python/JavaScript-inclusive error taxonomy.
- Add server-side authorization to admin endpoints and replace the single hardcoded admin allowlist with a proper role model.
- Persist session rename/delete operations to Supabase instead of LocalStorage only.
- Conduct controlled comparative evaluation of Learning Mode versus Quick Fix Mode on learning-outcome measures.
- Explore a locally fine-tuned correction model as a lower-cost complement to the current cloud-LLM dependency.

In conclusion, this paper has documented CodeMentor AI's architecture, module design, and AI pipeline directly against its source code, situating it within ten recent IEEE-published studies on AI-assisted programming education and debugging. The system's design is well motivated and its pedagogical positioning Socratic scaffolding over instant answers addresses a gap only partially covered by the reviewed literature. Equally, this paper has reported the respects in which the implementation currently falls short of that framing dataset composition, admin security, and the absence of recorded testing as a concrete, actionable agenda for the empirical and engineering work required before CodeMentor AI's educational benefits can be substantiated rather than merely designed for.

## REFERENCES

[1] R. Kathiroli, L. S. V., A. S., and S. K., "Learning and Debugging Assistant for Python with Retrieval Augmented Generation Capabilities," *Proceedings of the 2026 International Conference on Computing, Communication, Control and Cyber-Physical Systems (I5CPS)*, 2026.

[2] S. Shreeya, K. B. Kumar, and C. V. Krishna, "CodeGuide: An Artificial Intelligence Powered VS Code Extension for Automated Python Code Generation, Error Detection and Code Correction," *Proceedings of the 2025 4th International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 2025.

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 7, July 2026

DOI: 10.17148/IJARCCE.2026.15726

[3] D. Chandu, D. S. Sriram, and V. Ulagamuthalvi, "Voice-Controlled Smart Programming Assistant," *Proceedings of the 2025 7th International Conference on Intelligent Sustainable Systems (ICISS)*, 2025.

[4] R. Uyyala, M. Sajid, V. Madhav, and P. Dansena, "AI-Based Terminal Assistant: Code Generator, Summarizer & Smart-Debugger using RAG, FAISS Vector Retrieval, and Large Language Models," *Proceedings of the 2025 International Conference on Cognitive, Green and Ubiquitous Computing (IC-CGU)*, 2025.

[5] V. Vethika, Jenifer, R. Rohitha, and M. Revathi, "LLM-Powered Code Generation Using RAG Framework with LLaMA 3," *Proceedings of the 2025 9th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, 2025.

[6] A. Naman, R. Shariffdeen, G. Wang, S. Rasnayaka, and G. N. Iyer, "Analysis of Student-LLM Interaction in a Software Engineering Project," *Proceedings of the 2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code)*, 2025.

[7] A. Ouaazki, K. Bergram, and A. Holzer, "Leveraging ChatGPT to Enhance Computational Thinking Learning Experiences," *Proceedings of the 2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, 2023.

[8] M. Elhayany and C. Meinel, "Rethinking Retrieval Practice with AI: A GPT-Based Approach for Introductory Programming," *Proceedings of the 2025 IEEE Digital Education and MOOCS Conference (DEMOcon)*, 2025.

[9] D. Cotroneo, C. Improta, and P. Liguori, "Human-Written vs. AI-Generated Code: A Large-Scale Study of Defects, Vulnerabilities, and Complexity," *Proceedings of the 2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*, 2025.

[10] M. Čerňanský, P. Hafner, and I. Dirgová Luptáková, "LLM Tools for Programming," *Proceedings of the 2025 International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2025.

  