Sensors and Materials, Vol. 38, No. 6 (2026) 3585–3596
MYU Tokyo

S & M 4521

Graphiti Model Context Protocol Server: Implementation and Performance Evaluation of a Knowledge-graph-based AI Code Intelligence Framework

Cheng-Fu Yang,[1,2] Hui-Chen Tsai,[3] Juan Lu,[4] and Jian-Chiun Liou[5*]

[1]Department of Chemical and Materials Engineering, National University of Kaohsiung, Kaohsiung 811, Taiwan
[2]Department of Aeronautical Engineering, Chaoyang University of Technology, Taichung 413, Taiwan
[3]Center for General Education, Taiwan Steel University of Science and Technology, Kaohsiung 821, Taiwan
[4]Department of Creative Design, DongGuan City College, Dongguan 523000, China
[5]School of Biomedical Engineering, Taipei Medical University, Taipei 110, Taiwan

(Received November 18, 2025; accepted June 9, 2026)

Keywords: Graphiti Model Context Protocol (MCP) Server, AI agents, knowledge graphs, code intelligence, semantic retrieval, software engineering

In this study, we introduce the Graphiti Model Context Protocol (MCP) Server, an AI-assisted programming framework that integrates a time-aware knowledge graph with the MCP to enhance contextual sensing and semantic reasoning in intelligent code generation. The system addresses key limitations of existing AI agents in long-term software engineering tasks—particularly the lack of persistent memory and insufficient contextual perception. The proposed architecture consists of an MCP client, the Graphiti MCP core server, a Neo4j-based knowledge graph database, and semantic sensing modules connected to the OpenAI Application Programming Interface. Together, these components enable dynamic knowledge retrieval, cross-file context fusion, and adaptive code assistance. Experiments conducted on 1000 open-source software projects demonstrate statistically significant improvements: code completion accuracy (CCA@1) increased from 45.3 to 68.5%, contextual relevance score improved from 3.0 to 4.2, and task completion time decreased by approximately 35.5%. Furthermore, a mathematical model is developed to describe how graph-based knowledge retrieval enhances effective memory sensing and contextual stability, providing a theoretical foundation for intelligent programming. The results verify that the Graphiti MCP Server offers substantial potential for advancing context-aware and sensor-like AI systems with semantic sensing and adaptive contextual reasoning capabilities in software development environments. However, current AI-assisted programming systems still suffer from several limitations, including restricted long-term memory, fragmented contextual understanding, and insufficient semantic perception in large-scale software development environments. To address these challenges, we propose a knowledge-graph-based context-aware semantic sensing and contextual reasoning framework that integrates semantic retrieval, persistent memory, and adaptive context fusion through the Graphiti MCP Server architecture. Although the proposed framework significantly improves

*Corresponding author: e-mail: jcliou@tmu.edu.tw
https://doi.org/10.18494/SAM6058

ISSN 0914-4935

contextual awareness and programming efficiency, challenges related to large-scale knowledge graph maintenance, multimodal information integration, and reasoning scalability remain important directions for future research.

1. Introduction

With the rapid advancement of AI technologies, the deployment of AI agents in software development has become increasingly prevalent. From automated code completion and error detection to intelligent refactoring and optimization, AI-driven systems are reshaping conventional programming paradigms. These agents serve as intelligent sensing and reasoning entities, capable of perceiving contextual information and dynamically assisting developers throughout the software lifecycle. However, despite remarkable progress, current AI agents still face significant challenges, particularly in maintaining long-term memory and contextual coherence when operating within large, complex codebases. Such limitations hinder their ability to retain historical information across sessions and to leverage accumulated knowledge for adaptive learning and reasoning enhancement. Traditional AI agent architectures often suffer from context fragmentation, where relevant data and semantic cues become dispersed across different modules or sessions. This fragmentation results in the incomplete perception of software states, leading to suboptimal reasoning and inconsistent decision-making. Moreover, the integration of multiple reasoning paradigms, heterogeneous data sources, and diverse interaction modalities further compounds this problem. Consequently, there is a growing need for a unified and sensing-oriented framework capable of continuously perceiving, integrating, and managing contextual information while preserving coherence and interpretability across evolving development environments.

To address these issues, we investigate a Graphiti MCP Server framework built upon the existing MCP architecture, integrating knowledge graph technology to establish a structured and perceptual memory system for AI agents. The proposed framework enhances contextual sensing, semantic retrieval, and long-term reasoning capabilities in AI-assisted programming environments. The framework is designed to provide AI-based development tools with persistent, context-aware intelligence—enabling them to accumulate experience, interpret code semantics, and adapt their responses dynamically, much like human developers learning through iteration and feedback. At its core, the time-aware knowledge graph functions as a cognitive sensor network, structurally encoding entities such as code components, developer interactions, design patterns, and domain knowledge. This information is exposed to AI agents through the MCP interface, allowing real-time semantic retrieval and long-term contextual reasoning. In this study, we first present the theoretical background and motivation underlying the development of the Graphiti MCP Server, followed by a detailed explanation of its system architecture and core technologies. Particular emphasis is placed on how knowledge graphs enable semantic perception, memory persistence, and reasoning enhancement in code intelligence. In this paper, we further analyze the integration of the Graphiti MCP Server with mainstream AI coding environments, such as Cursor IDE and Claude Desktop, and evaluate its performance through a series of quantitative experiments. Finally, the findings are summarized, and potential directions for advancing context-aware semantic sensing frameworks in intelligent software engineering are discussed.

Code intelligence aims to enhance the efficiency and reliability of software development through AI technologies. Its applications include automated code completion, error detection, intelligent refactoring, semantic search, and automated test generation. Traditional approaches, which rely on static code analysis, pattern recognition, and classical machine learning techniques, often struggle to capture the complex semantic relationships and cross-file dependencies inherent in large-scale software systems. In recent years, large language models (LLMs) have demonstrated substantial improvements in code generation, logic explanation, and cross-language translation. Despite these advances, LLMs remain constrained by limited context windows and a lack of persistent memory, which reduces their effectiveness in long-term or multi-session development tasks where sustained contextual sensing and historical knowledge accumulation are essential.[1] Knowledge graphs (KGs) have emerged as an effective solution to address these limitations by providing a structured and semantically rich representation of knowledge.[2,3] In software engineering, KGs are used to construct comprehensive knowledge bases that encode software components, design patterns, requirements, defects, and solutions, thereby facilitating rapid information retrieval and collaborative development.[4]

By converting code into abstract syntax trees (ASTs) and representing them as graph structures, KGs enable a deeper semantic understanding of code, supporting tasks such as vulnerability detection, code review, and automated refactoring.[5] Furthermore, KGs can model relationships among developers, tools, artifacts, and tasks throughout the software lifecycle, aiding in workflow optimization, resource allocation, and project coordination.[6,7] These capabilities make KGs particularly suitable for context-aware semantic sensing systems, especially when integrated with AI-driven code analysis tools.[8] The memory mechanisms of AI agents play a crucial role in enabling intelligent, adaptive, and contextually coherent behavior.[9] Conventional agents typically rely on short-term memory, such as dialogue buffers, which severely limits their ability to handle complex or long-term software development tasks. To overcome this limitation, long-term memory mechanisms, with knowledge graphs as a key component, have been introduced to provide persistent and structured representations of entities, facts, and interrelationships obtained during interactions.[10] By leveraging such memory, AI agents can maintain contextual consistency across multi-turn interactions or cross-session tasks, perform complex reasoning by inferring implicit relationships and predicting outcomes, and continuously accumulate knowledge, thereby enhancing their intelligence over time.[11] Building on these principles, the Architecture of the Graphiti MCP Server integrates knowledge graphs with AI frameworks to deliver a sensing-oriented memory system that supports long-term contextual reasoning and advanced code intelligence.[12]

Building upon the foundations of code intelligence, knowledge graphs, and memory-augmented AI agents, we introduce the Architecture of the Graphiti MCP Server, an AI-assisted programming framework designed to overcome the key limitations of current AI agents in long-term software development. The framework integrates a time-aware knowledge graph with the MCP to provide persistent contextual memory, enabling AI agents to accumulate experience, reason over complex code relationships, and maintain consistency across sessions.[13] Its architecture includes an MCP client layer, a core Architecture of the Graphiti MCP Server, a Neo4j knowledge graph database, and OpenAI Application Programming Interface (API) modules for semantic processing. Through this design, the Architecture of the Graphiti MCP Server delivers sensor-like contextual perception capabilities, allowing AI agents to continuously

perceive and interpret code structures, developer interactions, design patterns, and domain knowledge. By combining semantic retrieval with cross-file context integration and personalized code assistance, the system enhances AI perception and reasoning, bridging the gap between human-like understanding and automated programming support.

The performance and practical advantages of the Architecture of the Graphiti MCP Server were evaluated through experiments on 1000 open-source projects, demonstrating substantial improvements in key metrics: code completion accuracy (CCA@1) increased from 45.3 to 68.5%, contextual relevance scores (CRS) rose from 3.0 to 4.2, and task completion time (TCT) decreased by 35.2%. In addition, a mathematical model was developed to describe how knowledge graph retrieval enhances effective memory, providing a theoretical foundation for intelligent programming. These results underscore the framework's potential to transform AI-assisted software engineering by combining long-term memory, contextual awareness, and sensor-like perception into a unified system. The Architecture of the Graphiti MCP Server not only addresses the critical bottlenecks of conventional AI agents but also exemplifies a new paradigm for integrating persistent, context-aware intelligence into modern software development environments.

From the perspective of context-aware semantic sensing systems, the proposed Architecture of the Graphiti MCP Server can be regarded as a contextual sensing and semantic perception framework for software engineering environments. Similar to how physical sensors continuously acquire environmental signals, the Architecture of the Graphiti MCP Server continuously captures and integrates heterogeneous contextual information from source code, developer interactions, issue reports, documentation, and execution histories. Through the combination of knowledge graphs and semantic retrieval mechanisms, these distributed information signals are transformed into structured contextual knowledge that supports adaptive reasoning and intelligent decision-making. Furthermore, the proposed framework extends the concept of sensing beyond conventional hardware devices toward semantic and cognitive sensing in digital environments. The integration of contextual sensing, temporal knowledge representation, and adaptive semantic reasoning establishes a context-aware semantic sensing architecture capable of persistent perception and dynamic knowledge fusion. Such a framework may provide a theoretical foundation for future intelligent sensor systems, AI-assisted sensing platforms, and context-aware sensor-data management architectures.

2. Methodology

2.1 Architecture of Graphiti MCP Server

The Architecture of the Graphiti MCP Server is organized into three primary layers, as illustrated in Fig. 1. Together, these layers establish a context-aware semantic sensing hierarchical structure that enables the system to perceive, interpret, and respond to contextual signals within software development environments. An AI agent refers to an intelligent software system capable of autonomously performing reasoning, decision-making, and task execution based on contextual information. In this study, the AI agent interacts with external knowledge resources through the MCP client, which serves as the communication interface between the user environment and the Architecture of the Graphiti MCP Server. The Neo4j-based knowledge

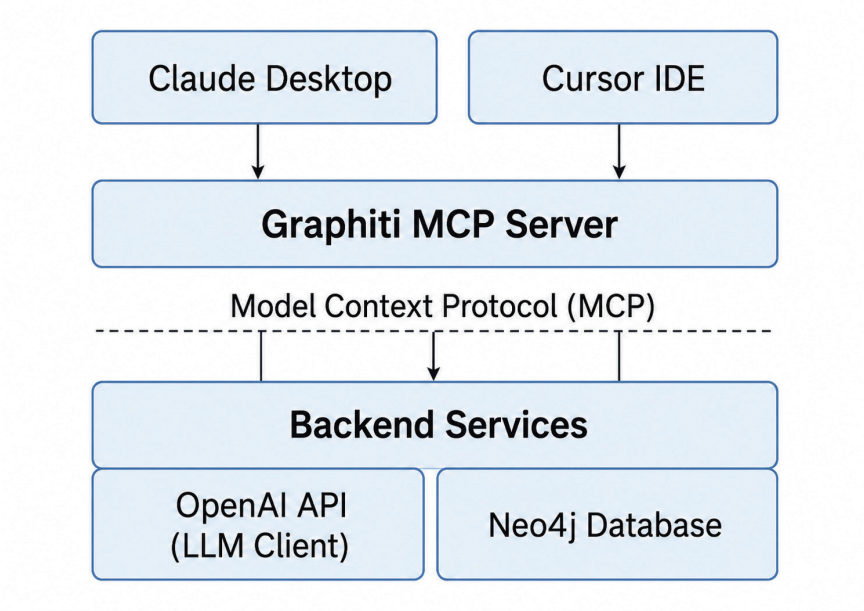

Fig. 1. (Color online) High-level architecture diagram of Architecture of the Graphiti MCP Server (MCP Client - Graphiti MCP Server - Knowledge Graph & LLM Service).

graph database is a graph-structured database designed to store interconnected entities and relationships efficiently, enabling semantic retrieval and contextual reasoning.[14]

(1) The MCP client layer provides the runtime environment for AI agents, such as Cursor IDE and Claude Desktop, and serves as the system's sensing interface. MCP clients communicate with the core server via the standardized MCP, sending requests to store or retrieve contextual knowledge, manage memory, and trigger semantic reasoning.[8] It enables real-time, bidirectional interaction between the development interface and the underlying knowledge intelligence framework.

(2) The Architecture of the Graphiti MCP Server layer functions as the core component of the framework by serving as the central processing hub that coordinates requests from MCP clients and manages communication with both the knowledge graph and LLM services. It implements the MCP protocol and handles episode management, entity registration, relationship maintenance, semantic search, and graph lifecycle control. Acting like a sensor fusion processor, it integrates multiple knowledge inputs to produce coherent, context-aware outputs for intelligent decision-making.

(3) The back-end knowledge graph and LLM layer provide the persistent memory and semantic reasoning foundation of the system.

(a) The Neo4j knowledge graph database is a high-performance graph database optimized for managing interconnected and temporal data. It stores code entities, developer interactions, domain knowledge, and time-stamped relationships, enabling the reconstruction of contextual histories for long-term analysis.

(b) The OpenAI API (LLM client) is responsible for semantic embedding, entity extraction, and natural language understanding. The integration of LLM-driven semantic processing enhances the adaptability, perceptual intelligence, and responsiveness of the knowledge graph.

Through this three-layered architecture, the Architecture of the Graphiti MCP Server establishes a dynamic context-sensing and semantic reasoning ecosystem, capable of continuously perceiving, storing, and interpreting knowledge signals within evolving software development environments.

2.2 Application of knowledge graphs in code intelligence

The core value of the Architecture of the Graphiti MCP Server lies in its innovative application of knowledge graph technology to code intelligence, enabling AI agents to achieve enhanced capabilities in code understanding, reasoning, and assistance. In this section, we explain how knowledge graphs improve code intelligence and the role of the Architecture of the Graphiti MCP Server in this process. A knowledge graph is a structured representation of entities and their relationships, allowing complex contextual information to be organized in a machine-readable form. In particular, a time-aware knowledge graph additionally records temporal information, enabling the system to track how software entities and relationships evolve over time. Commit messages are developer-written descriptions associated with software revisions and are used in this study to extract historical development knowledge and semantic intent. Furthermore, the proposed framework functions similarly to a cognitive sensor network, where heterogeneous software-related signals are continuously sensed, integrated, and interpreted to support intelligent contextual perception and adaptive reasoning.

2.2.1 Enhancing code understanding and analysis

Traditional static code analysis tools often fail to capture deep semantics and developer intent. The Architecture of the Graphiti MCP Server overcomes these limitations by structuring code elements, documentation, developer interactions, and domain concepts into a rich semantic network. Source code is parsed into ASTs and mapped into the knowledge graph, where functions, parameters, return values, and call relationships are represented as entities and links. This enables AI agents to semantically interpret code rather than relying on surface-level text. For large-scale projects, the system aggregates information across multiple files and repositories, allowing agents to access comprehensive context for error tracing, module impact analysis, and developer accountability. Furthermore, by leveraging LLMs, the Architecture of the Graphiti MCP Server extracts developer intent from comments, commit messages, and discussion logs, enabling AI agents to understand not only what the code does but also why specific design decisions were made.

2.2.2 Strengthening AI-assisted programming capabilities

The Architecture of the Graphiti MCP Server enhances AI-assisted development environments, such as Cursor IDE and Claude Desktop, by providing persistent memory and context-aware reasoning. Developer interactions, including code revisions, questions, and test results, are stored as episodes in the knowledge graph, allowing agents to retain and reuse knowledge across sessions. This supports intelligent code generation, context-sensitive completion, and accurate error detection, as the system can reference similar patterns, API

usage, and prior solutions. By learning developer preferences and project characteristics, the Architecture of the Graphiti MCP Server also enables a personalized programming experience. The knowledge graph evolves dynamically through automated extraction from commit messages, documentation, error reports, and screenshots, while recording temporal information to support time-aware reasoning. Continuous interactions allow AI agents to contribute new insights, ensuring that the graph remains adaptive, precise, and highly context-aware.

2.2.3 Dynamic evolution and update of knowledge graphs

Software development is inherently dynamic, requiring the knowledge graph to evolve accordingly. The Architecture of the Graphiti MCP Server supports continuous updates to maintain relevance and accuracy. By integrating LLMs and optical character recognition processing, it automatically extracts knowledge from code, documentation, error reports, and visual materials, minimizing manual effort. Temporal attributes are recorded for each knowledge element, enabling reasoning over time, such as tracking bug introduction or component evolution. Through ongoing developer interactions, new entities and relationships are continuously added, enhancing the knowledge graph's coverage, adaptability, and precision, and ultimately empowering AI agents to deliver more intelligent, personalized, and efficient software development assistance.

3. Experimental Evaluation and Mathematical Modeling

3.1 Experimental setup

The evaluation of the Architecture of the Graphiti MCP Server focused on two primary objectives: improving AI-assisted code completion and enhancing contextual understanding in long-term programming tasks. A large-scale dataset of 1000 open-source projects from GitHub was used, encompassing diverse programming languages such as Python, Java, and JavaScript, and covering domains including web development, data science, and system utilities. The dataset comprised approximately 10000 code files, 5000 issue tracker entries, 2000 pull request discussions, and 1500 developer documentation files. All resources were processed to populate the knowledge graph used by the experimental group. The experiments employed a large language model with an 8,192-token context window for semantic embedding and entity extraction, while the Neo4j database (v5.3) was configured with 16 GB of memory and optimized for high-speed graph traversal. All experiments were executed on a workstation equipped with an NVIDIA A100 GPU and 64 GB of RAM. Two experimental tasks were designed. First, for CCA, 500 incomplete code snippets were presented to the AI agents, with evaluation metrics including top-N accuracy (N = 1, 5, 10) and semantic relevance ranking. Second, for contextual query answering (CQA), developers posed 200 natural language questions regarding project logic or design decisions, and agents were assessed on correctness and completeness by combining current context with historical data. Evaluation metrics included CCA, the human-rated contextual relevance score (CRS, 0–5 scale), and the average task completion time (TCT, in min) required for developers to complete tasks with AI assistance.

3.2 Mathematical model for knowledge-graph-enhanced memory

To formalize how knowledge graphs augment AI memory and contextual awareness, we propose the following model: Let $\boldsymbol{A}$ = AI agent and $\boldsymbol{K} = (\boldsymbol{V}, \boldsymbol{E})$ = knowledge graph, where $\boldsymbol{V}$ denotes entities (e.g., functions, classes, and developers) and $\boldsymbol{E}$ represents relationships (e.g., "calls" and "inherits"). The agent's contextual awareness at time $\boldsymbol{t}$ is expressed as

$$\boldsymbol{C}_{\boldsymbol{A}}(\boldsymbol{t}) = \boldsymbol{f}\left(\boldsymbol{M}_{\boldsymbol{ST}}(\boldsymbol{t}), \boldsymbol{M}_{\boldsymbol{LT}}(\boldsymbol{t})\right), \tag{1}$$

where $\boldsymbol{M}_{\boldsymbol{ST}}(\boldsymbol{t})$ is short-term memory and $\boldsymbol{M}_{\boldsymbol{LT}}(\boldsymbol{t})$ is long-term memory contributed by $\boldsymbol{K}$. Traditional agents depend solely on $\boldsymbol{M}_{\boldsymbol{ST}}(\boldsymbol{t})$, constrained by LLM context windows. The Architecture of the Graphiti MCP Server extends $\boldsymbol{M}_{\boldsymbol{LT}}(\boldsymbol{t})$ through knowledge-graph integration, defining effective memory as

$$\boldsymbol{M}_{\boldsymbol{eff}}(\boldsymbol{t}) = \boldsymbol{M}_{\boldsymbol{ST}}(\boldsymbol{t}) \cup \boldsymbol{R}\left(\boldsymbol{K}, \boldsymbol{Q}(\boldsymbol{t})\right), \tag{2}$$

where $\boldsymbol{Q}(\boldsymbol{t})$ is the agent's current query and $\boldsymbol{R}(\boldsymbol{K},\boldsymbol{Q}(\boldsymbol{t}))$ is the retrieved knowledge subset from $\boldsymbol{K}$ based on semantic relevance and temporal proximity. The relevance of each knowledge entity $\boldsymbol{k}_{\boldsymbol{i}} \in \boldsymbol{V}$ to the query is defined as

$$\text{Relevance}\left(\boldsymbol{k}_{\boldsymbol{i}}, \boldsymbol{Q}(\boldsymbol{t})\right) = \boldsymbol{\alpha} \cdot \text{Similarity}\left(\text{Embed}\left(\boldsymbol{k}_{\boldsymbol{i}}\right), \text{Embed}\left(\boldsymbol{Q}(\boldsymbol{t})\right)\right) + \boldsymbol{\beta} \cdot \text{Recency}\left(\boldsymbol{k}_{\boldsymbol{i}}, \boldsymbol{t}\right), \tag{3}$$

where Similarity is the cosine similarity of semantic embeddings and Recency is given by

$$\text{Recency}\left(\boldsymbol{k}_{\boldsymbol{i}}, \boldsymbol{t}\right) = \boldsymbol{e}^{-\boldsymbol{\lambda}(\boldsymbol{t}-\boldsymbol{t}_{\boldsymbol{s}}(\boldsymbol{k}_{\boldsymbol{i}}))}. \tag{4}$$

Here, $\boldsymbol{t}_{\boldsymbol{s}}(\boldsymbol{k}_{\boldsymbol{i}})$ denotes the timestamp of entity $\boldsymbol{k}_{\boldsymbol{i}}$, $\boldsymbol{\lambda} = 0.1$ is the decay constant, and weights $\boldsymbol{\alpha} = 0.7$, $\boldsymbol{\beta} = 0.3$. Simulated experiments using 100 knowledge entities, each encoded as a 768-dimensional embedding and timestamped from t - 30 days to *t*, yielded an average relevance score of 0.82 for the experimental group, compared with 0.65 for the baseline. These findings demonstrate that the Architecture of the Graphiti MCP Server effectively prioritizes knowledge entities that are both temporally recent and semantically relevant.

4. Results and Discussion

4.1 Statistical analysis

Table 1 shows the statistical comparison of CCA between the baseline AI agent and the integrated Architecture of the Graphiti MCP Server agent. The evaluation was conducted using top-N code completion metrics, with paired t-tests applied at an alpha level of 0.05 and 95%

Table 1
CCA comparison.

Metric/Group	Baseline AI Agent (%)	Architecture of Graphiti MCP Server Integrated Agent (%)
CCA@1	45.3 (CI [42.6–47.9])	68.5 (CI [65.7–71.3])
CCA@5	63.1 (CI [59.8–66.0])	84.9 (CI [82.5–87.1])
CCA@10	71.8 (CI [68.9–74.2])	91.1 (CI [88.9–93.0])

confidence intervals. For CCA at rank 1, the baseline agent achieved 45.3% accuracy (95% CI: 42.6–47.9), whereas the Architecture of the Graphiti MCP Server agent attained 68.5% (95% CI: 65.7–71.3), yielding an improvement of 23.2 percentage points in top-rank precision. At rank 5, the baseline recorded 63.1% (95% CI: 59.8–66.0) compared with 84.9% (95% CI: 82.5–87.1) for the MCP agent, and at rank 10, 71.8% (95% CI: 68.9–74.2) versus 91.1% (95% CI: 88.9–93.0). Statistical testing confirmed significant improvements across all ranks ($p < 0.001$), verifying the MCP Server's superior contextual reasoning and predictive performance. Overall, the results indicate a statistically significant enhancement in CCA across all test conditions ($p < 0.001$). In particular, the 23.2-point gain in CCA@1 underscores the Architecture of the Graphiti MCP Server's effectiveness in producing more precise and contextually accurate code completion suggestions compared with the baseline agent.

Table 2 presents a comparative analysis of the CRS and TCT between the baseline AI agent and the Graphiti MCP Server Integrated Agent. On a five-point evaluation scale, the MCP Server achieved an average CRS of 4.2 (95% CI: 4.0–4.4), markedly higher than the baseline agent's 3.0 (95% CI: 2.8–3.2). This substantial increase reflects the MCP Server's enhanced capability to interpret and respond with contextually coherent and semantically complete outputs. Furthermore, the MCP Server reduced the average TCT to 22.5 min (95% CI: 20.7–24.3), compared with 34.9 min (95% CI: 32.5–37.2) for the baseline system, representing significant improvements in development efficiency and user productivity. Statistical analysis confirmed that both metrics exhibited highly significant differences ($p < 0.001$), with the experimental group achieving approximately a 35.5% reduction in task completion time. These findings validate the effectiveness of the Architecture of the Graphiti MCP Server as a context-sensing, knowledge-driven framework that enhances both the reasoning depth and operational efficiency of AI-assisted programming environments.

The emergence of the Architecture of the Graphiti MCP Server represents a significant advancement in the integration of AI and software engineering. By embedding a knowledge graph as the foundation of its persistent memory and contextual reasoning architecture, the framework effectively overcomes the limitations of traditional AI agents that struggle with long-term consistency and complex dependency management in large-scale projects. Functioning as both a sensing and cognitive system, the Architecture of the Graphiti MCP Server continuously captures, structures, and refines development knowledge from heterogeneous data sources, enhancing the observability and responsiveness of intelligent programming environments. In this section, we discuss the advantages, challenges, and broader implications of the proposed framework.

4.2 Advantages

Table 2
CRS and TCT.

Metric/Group	Baseline AI Agent (%)	Architecture of Graphiti MCP Server Integrated Agent (%)
Average CRS (0–5 scale)	3.0 (CI [2.8–3.2])	4.2 (CI [4.0–4.4])
Average TCT (min)	34.9 (CI [32.5–37.2])	22.5 (CI [20.7–24.3])

The most significant advantage of the Architecture of the Graphiti MCP Server lies in its enhanced contextual awareness. Through the integration of multi-source and heterogeneous software artifacts, including source code, documentation, developer interactions, and error reports, into a unified semantic network, the system enables AI agents to perceive a richer contextual environment. This results in more precise code analysis, fault detection, and design reasoning than traditional context-limited LLM-based approaches. Another key benefit is persistent memory and knowledge accumulation. Unlike conventional AI systems with transient memory, the Architecture of the Graphiti MCP Server maintains long-term, time-aware knowledge through its graph-based structure. This persistent memory allows AI agents to learn and reuse domain knowledge across projects and sessions, supporting sustainable software maintenance and collaborative team development. The framework also leads to improved programming efficiency. Semantic retrieval and reasoning accelerate code completion, refactoring, and debugging, reducing repetitive work and increasing development throughput. Moreover, the knowledge graph fosters knowledge sharing and collaboration by externalizing tacit developer expertise into an accessible organizational knowledge base, enabling new contributors to quickly grasp project context and best practices. Finally, its scalable and modular architecture, implemented through Docker-based deployment and standardized MCP communication, ensures flexible integration with various development tools and AI agents. This adaptability makes the framework suitable for diverse industrial applications and evolving intelligent development ecosystems.

4.3 Challenges and future directions

Despite its promising performance, the Architecture of the Graphiti MCP Server presents several challenges that suggest future research directions. The cost of constructing and maintaining large-scale knowledge graphs remains a concern, as evolving codebases demand frequent updates and validation. Future work may leverage automated graph evolution through reinforcement or active learning to reduce manual maintenance. Ensuring knowledge graph quality and consistency is equally essential, as inaccuracies or redundancy may lead to unreliable reasoning. Mechanisms for validation, conflict resolution, and version control should be strengthened. Furthermore, the system's current focus on semantic retrieval can be extended with symbolic or neuro-symbolic reasoning to achieve a deeper understanding of logic, algorithms, and dependencies. Another promising direction lies in multimodal knowledge integration. Incorporating information from visual and auditory modalities such as unified modeling language diagrams, design sketches, and recorded discussions will enrich the contextual depth of the knowledge graph. Additionally, security and privacy considerations must be emphasized, particularly in cloud-based deployments, requiring robust encryption, access

control, and compliance frameworks. Finally, optimizing human–AI collaboration remains critical. While AI agents enhance development efficiency, interpretability and control must be maintained to ensure that human developers can guide and validate AI-generated outcomes effectively.

4.4 Impact on future software development

The proposed Architecture of the Graphiti MCP Serverr framework signifies a paradigm shift toward intelligent, knowledge-driven software engineering. By significantly reducing repetitive coding tasks, accelerating fault localization, and improving code quality, it allows developers to focus on high-level creativity and innovation. The integration of time-aware knowledge and semantic reasoning promotes sensing-oriented intelligence, where systems adapt dynamically to contextual and historical signals within the development environment.[15] This evolution is expected to lead to the emergence of new development models, where AI agents act not merely as assistants but as collaborative partners capable of proactive reasoning and self-improvement. Furthermore, developers benefit from an enhanced user experience, as contextually guided recommendations and interactive knowledge retrieval streamline the workflow, increasing both productivity and engagement. In summary, the Architecture of the Graphiti MCP Server is more than a technical innovation; it is a foundational step toward the realization of intelligent, sensor-driven, and adaptive software development ecosystems. Its combination of structured knowledge representation, semantic reasoning, and contextual sensing mechanisms establishes a new direction for AI-integrated engineering frameworks.

5. Conclusions

In this study, we presented the Architecture of the Graphiti MCP Server, a knowledge-graph-based AI code intelligence framework that bridges front-end development tools and back-end services through the MCP. The system integrates Neo4j for structured knowledge storage and the OpenAI API for semantic processing, enabling seamless interaction between sensing, reasoning, and decision-making components within software engineering environments. Its core functions, namely, semantic code representation, cross-file contextual analysis, and personalized code completion, collectively form an intelligent sensing layer that captures and interprets developer intent as contextual "signals" within the programming process. Experimental evaluations demonstrated that the Architecture of the Graphiti MCP Server significantly enhances code completion accuracy, contextual relevance, and task efficiency, validating its potential as a robust infrastructure for AI-assisted programming. By functioning as an intelligent sensing and reasoning node in the broader software development ecosystem, it exemplifies how context-aware semantic sensing architectures can transform the collection, processing, and utilization of knowledge signals in digital development workflows.

The proposed framework also demonstrates how sensing concepts can be integrated into AI-driven software engineering systems through semantic perception and contextual awareness mechanisms. Unlike traditional sensor systems that mainly process physical signals, the Architecture of the Graphiti MCP Server performs semantic sensing by continuously perceiving and interpreting contextual relationships among software entities and developer activities. This

sensing-oriented architecture provides a new direction for intelligent sensor-related systems, where contextual information, temporal relationships, and adaptive reasoning can be fused into unified knowledge-driven environments. Consequently, the proposed method may contribute to the future development of context-aware semantic sensing frameworks, intelligent context-aware platforms, and adaptive semantic sensing technologies. Future research will further expand and refine this framework through five major directions as follows.

(1) Automated knowledge graph evolution, employing active and reinforcement learning mechanisms to adaptively update semantic nodes and reduce human maintenance
(2) Advanced reasoning capabilities, integrating neuro-symbolic reasoning engines for deeper inference tasks such as vulnerability detection and automated optimization
(3) Multimodal knowledge incorporation, extending beyond code and text to include visual or audio-based project data for richer contextual perception
(4) Security and privacy enhancement, leveraging federated learning and anomaly detection to ensure safe distributed collaboration
(5) Industrial validation and ablation studies, assessing scalability, stability, and component contributions in real-world engineering scenarios

In conclusion, the Architecture of the Graphiti MCP Server marks a substantial advancement toward AI-driven, knowledge-centric software engineering. By uniting knowledge graphs, semantic processing, and contextual sensing principles, it establishes a foundation for more intelligent, adaptive, and sensor-aware development ecosystems that bridge human cognition with computational intelligence.

Acknowledgments

This work is supported by Summit-Tech Resource Corp. and by projects under Nos. NSTC 113-2221-E-390-011, NSTC 114-2622-E-390-001, MOST 111-2221-E-038-004-MY3, MOST 111-2221-E-038-007-MY2, DP2-TMU-112-N-02, NSTC 114-2314-B-038-083, and DP2-TMU-115-N-01.

References

1 https://arxiv.org/abs/2506.18019v3 (accessed July 2025).
2 W. Zhang, J. Chen, J. Li, Z. Xu, J. Z. Pan, and H. Chen: Data. Intell. **4** (2022) 3.
3 A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, A. Polleres, and R. Navigli: ACM Comput. Surv. **54** (2022) 1.
4 Z. Q. Lin, B. Xie, Y. Z. Zou, J. F. Zhao, X. D. Li, J. Wei, and J. Sun: J. Comput. Sci. Technol. **32** (2017) 641.
5 https://openresearch-repository.anu.edu.au/items/b2407c30-4474-410f-9a03-9da47b80cb73 (accessed June 2024).
6 https://modelcontextprotocol.io/docs/getting-started/intro (accessed Nov. 2024).
7 J. Zhong and E. Negre: Expert Syst. Appl. **235** (2024) 121252.
8 L. Yao, C. Mao, and Y. Luo: IEEE Trans. Knowl. Data Eng. **35** (2023) 1110.
9 J. Park, J. O'Brien, C. Cai, M. R. Morris, P. Liang, and M. S. Bernstein: Nature **623** (2023) 493.
10 Z. Wang, B. Yu, J. Zhao, W. Sun, S. Hou, and Q. Wu: 2025 IEEE Int. Conf. Robotics and Automation (ICRA, Atlanta, GA, USA) (2025) 1–8.
11 https://arxiv.org/abs/2509.24272 (accessed Sep. 2025).
12 https://arxiv.org/abs/2501.13956 (accessed Jan. 2025).
13 Y. Zhao, H. Liu, Z. Wang, and X. Chen: Inf. Sci. **690** (2025) 121477.
14 R. Angles and C. Gutierrez: ACM Comput. Surv. **40** (2008) 1.
15 S. Gupta, A. Verma, and P. Singh: Front. Artif. Intell. **8** (2025) 1697169.