

The effectiveness of ChatGPT in extracting architectural patterns and tactics

Hind Milhem, Naderah Al-Jawabrah, Raghad Abu Wadi

Department of Information Technology, Faculty of Prince Al-Hussein bin Abdullah II for Information Technology,
Hashemite University, Zarqa, Jordan

Article Info

Article history:

Received Jan 12, 2025

Revised Jul 14, 2025

Accepted Aug 6, 2025

Keywords:

Architectural pattern

Architectural tactic

Artificial intelligence

ChatGPT

Software engineering

ABSTRACT

This work investigates the potential of ChatGPT, a cutting-edge large language model (LLM), for software design analysis specifically in detecting architectural patterns and tactics. The evaluation involves comparing ChatGPT's performance with that of Archie, a traditional Eclipse plugin designed for architectural analysis. The study uses the source code of five open-source software systems as the testing ground. Results reveal that ChatGPT achieves noteworthy performance in both pattern and tactic detection tasks. Specifically, for pattern detection, ChatGPT demonstrates an accuracy of up to 47.06%, while for tactic detection, it achieves a precision of 28.25%. While ChatGPT's current capabilities are not yet a replacement for specialized tools like Archie, it offers significant potential as a complementary tool in architectural analysis workflows. By bridging the gap between natural language understanding and software engineering, ChatGPT could pave the way for more intelligent and automated solutions in the field. However, a key limitation is its difficulties in handling foundational or traditional tactics, resulting in a lower detection rate in certain areas. This research contributes valuable insights into the application of LLMs in software engineering, highlighting both the strengths and the limitations of ChatGPT in addressing complex architectural tasks.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Hind Milhem

Department of Information Technology

Faculty of Prince Al-Hussein bin Abdullah II for Information Technology, Hashemite University

Zarqa, Jordan

Email: hinda_is@hu.edu.jo

1. INTRODUCTION

Software architecture plays a crucial role in determining software systems' maintainability, scalability, and overall quality. It encompasses architectural patterns, tactics, and quality attributes. Architectural patterns [1] are reusable, high-level design solutions that provide structured approaches to organizing software systems, often derived from proven best practices in framework development such as broker pattern [2] and layer pattern [3]. Tactics [4] are design decisions that influence the control of a system's quality attributes [5], such as performance, security, or modifiability, which collectively define the system's structure, design, and behaviour. Identifying these elements in existing codebases is critical for architectural reviews, quality assessments, system re-engineering, and refactoring tasks. Traditionally, this identification process relies heavily on expert knowledge and manual analysis, which can be time-consuming and error-prone, such as the traditional tool Archie [6]–[8]. Traditional tools like Archie depend on predefined rules and static heuristics to detect patterns/tactics, making them inflexible when analyzing systems. Archie also has a limitation in its ability to interpret code comments, documentation, or implicit design intent

because of the lack of natural language processing (NLP) [9]. Additionally, these tools also require manual updates to their rule sets to add new patterns, which can be time-consuming. While Archie is good at detecting traditional tactics such as Kerberos and authentication, it misses modern tactics such as circuit breakers and retry logic, which are critical tactics in cloud-native or AI-driven systems. All of these limitations highlight the need for more adaptive solutions and motivate us to establish this work.

Recent advances in NLP [9] and the emergence of large language models (LLM) [10] like ChatGPT [11] have opened new avenues for automating software analysis tasks. While these technologies have applied to different areas such as requirements extraction [12] and bug detection [13], their application to software architecture analysis—specifically in detecting patterns and tactics—remains underexplored. A more cohesive integration between NLP capabilities and architectural analysis could bridge this gap, resulting in more intelligent and context-aware automation. ChatGPT, with its deep understanding of human language and pre-trained knowledge of programming constructs, can assist in extracting and reasoning about architectural patterns, tactics, and quality attributes directly from the source code. However, the effectiveness of such language models in performing these specific tasks remains relatively unexplored.

The exploration of ChatGPT's capabilities extends across various domains. For instance, Tan [14] highlights its ability to extract design concepts from narratives, showcasing its transformative potential in creative fields. Similarly, Gilson *et al.* [15] emphasize NLP's role in identifying quality attributes from user stories, aiding early architectural decisions. Further, Das *et al.* [16] streamline goal modeling processes by automating the extraction of goals from unstructured requirements, improving stakeholder alignment. In healthcare, Huang *et al.* [17] demonstrate ChatGPT's proficiency in clinical data extraction, outperforming traditional methods. Moreover, Sun *et al.* [18] leverage ChatGPT for pharmacovigilance event extraction, and Mohajer *et al.* [19] reveal its effectiveness in static analysis for bug detection. Terzi *et al.* [20] analyze developer interactions with ChatGPT's code suggestions, showing improved outcomes with refined prompts. Mahmoudi *et al.* [21] propose ChatGPT-based frameworks for systematic reviews, while Pragyani *et al.* [22] highlight its potential in automating use case extractions. Ahmad *et al.* [23] examine how the AI can assist software architects by fostering collaboration throughout the design process. Despite this growing interest, limited work has focused on applying ChatGPT to the conceptual elements of software architecture, such as identifying patterns and tactics. This gap represents an opportunity to extend the capabilities of AI models into impactful areas, contributing novel insights to the field.

This paper aims to evaluate the effectiveness of ChatGPT in identifying architectural patterns and tactics from software systems' source code. Specifically, we address two key research questions:

RQ1: how effective is ChatGPT in extracting architectural patterns from software systems' source code?

RQ2: how effective is ChatGPT in extracting architectural tactics from software systems' source code?

By answering these questions, this research seeks to provide insights into the capabilities and limitations of using ChatGPT for architectural analysis, potentially informing the design of more intelligent, automated tools for software engineering tasks. Our contributions to this work are:

- Conducting experiments to extract architectural patterns and tactics from the source code of five open-source systems using ChatGPT which are: Apache Storm [24], Apache Flink [25], Apache Spark [26], Gradle [27], and Maven [28].
- Performing a comparative analysis of ChatGPT's performance against Archie, a traditional architectural analysis tool.
- Measuring and evaluating ChatGPT's performance through precision, recall, and accuracy metrics.
- Addressing the defined research questions by analyzing the experimental results.

The remainder of this paper is organized as follows: section 2 details the research methodology employed. Our results and discussions are presented and analyzed in section 3. Section 4 addresses potential threats to validity. Finally, section 5 concludes the paper and outlines directions for future research.

2. METHOD

2.1. Overview

Figure 1 provides an overview of the pipeline used in our work. The process is structured into four main steps:

- i) Input source code:
 - Inputs: the pipeline begins with two types of input: the complete source code (zipped) and selected code snippets and
 - These inputs are used as prompts for ChatGPT, initiating the process of identifying architectural patterns and tactics. However, only the complete source code (zipped) is used as input for Archie.
- ii) Extract patterns and tactics:

- Archie: the source code undergoes preprocessing, followed by training, and then detection proper and this results in two outputs: extracted patterns and extracted tactics.
 - ChatGPT: the input is processed through pre-trained understanding, analysis, and recognition stages and ChatGPT produces two similar outputs: extracted patterns and extracted tactics.
- iii) Compare results: the patterns and tactics identified by Archie and ChatGPT are compared to analyze their similarities, differences, and effectiveness in identifying these elements.
 - iv) Calculate accuracy: the final step involves calculating the accuracy of the outputs from both Archie and ChatGPT to evaluate their performance in extracting architectural patterns and tactics from the given source code.

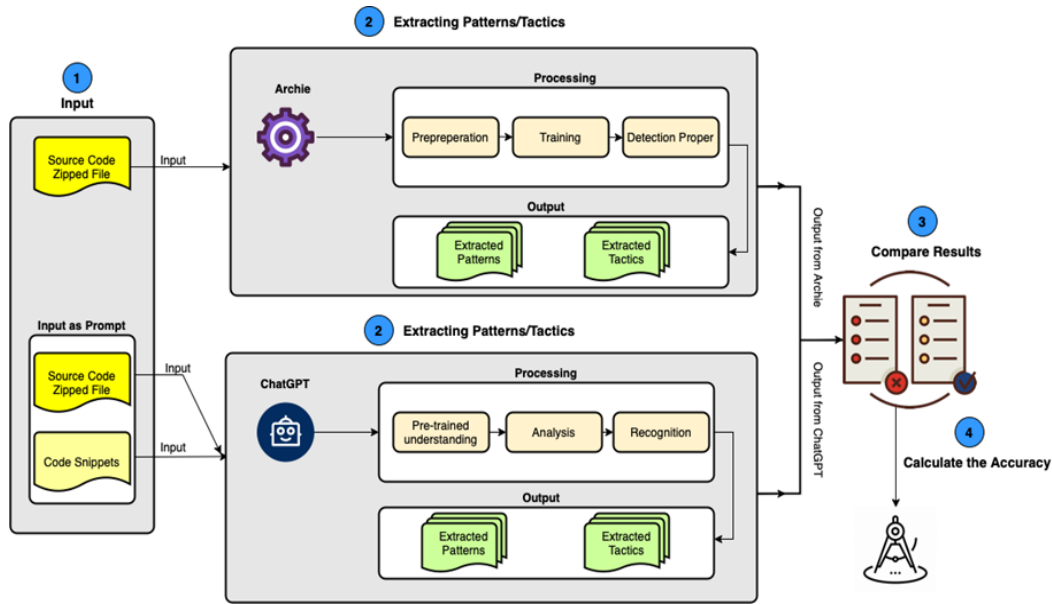


Figure 1. The overview of our work

2.2. Prompting Engineering Strategies

In this study, we utilized specific prompts to extract architectural patterns and tactics using ChatGPT. For Figures 2 and 3, the entire source code file of each system was provided, and ChatGPT was tasked with identifying patterns and tactics from the full codebase. In Figure 4, we supplied selected snippets of the code and requested ChatGPT to extract patterns and tactics based solely on these excerpts. Additionally, we asked ChatGPT to extract specific code snippets from the system's source code and then determine the architectural patterns and tactics based on the extracted portions. Figure 4(a) shows the prompt used after providing the code snippets, while Figure 4(b) presents the prompt before any snippets were given.

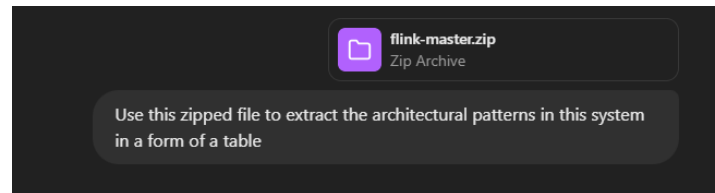


Figure 2. Prompt utilizing the entire source code to identify architectural patterns

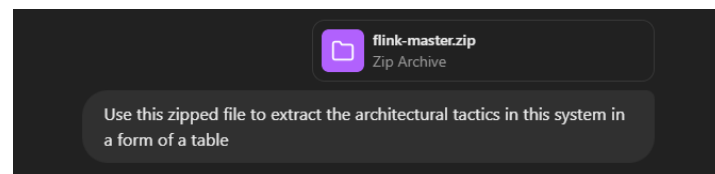


Figure 3. Prompt utilizing the entire source code to identify architectural tactics

Use the snippets of code I provided above to extract the Architectural Patterns you find in a table including Patterns, Description, and Code Reference.

(a)

What snippets of code do you need from the zipped file to extract Architectural Patterns

(b)

Figure 4. Prompt using specific snippets of code: (a) after providing snippets of code and (b) before providing snippets of code

2.3. Evaluation metrics

In this study, we evaluate the performance of ChatGPT using recall, precision, and accuracy metrics to assess the effectiveness of our experiments. For this purpose, we calculate the true positives (TP), false positives (FP), and false negatives (FN) required for these metrics. However, we do not compute the FN as all patterns and tactics were evaluated and accounted for during detection. Specifically:

- TP: the number of patterns or tactics correctly identified by ChatGPT or Archie.
- FP: the number of patterns or tactics incorrectly identified by ChatGPT or Archie.
- FN: the number of patterns or tactics missed by ChatGPT or Archie.

3. RESULTS AND DISCUSSION

3.1. RQ1: how effective is ChatGPT in extracting architectural patterns from software systems source code?

To address this question, we utilized ChatGPT to identify patterns from the complete codebase and compared the results with those from the traditional tool Archie. We present the results of pattern and tactic detection in Apache Flink using both tools. Table 1 summarizes the comparison of pattern detection outcomes between Archie and ChatGPT, while Figure 5 illustrates the percentage of patterns detected by each tool. As observed, ChatGPT achieved a pattern detection rate of 55.6%, compared to Archie's rate of 44.4%.

Table 1. Comparison results of the patterns detection for Apache Flink

Architectural tactic	Also known as	Archie	ChatGPT
Pipeline pattern	Streaming pipeline	✗	✓
Master-slave pattern		✗	✓
Layered architecture	Multitier architecture	✗	✓
Event-driven architecture	Message-driven architecture	✗	✓
Service component pattern		✗	✓
Layers	Tiered system	✓	✗
Broker	Message broke	✓	✗
Observer/publish-subscribe		✓	✗
Pipes and filters		✓	✗
Shared-repository	Common data repository	✗	✗

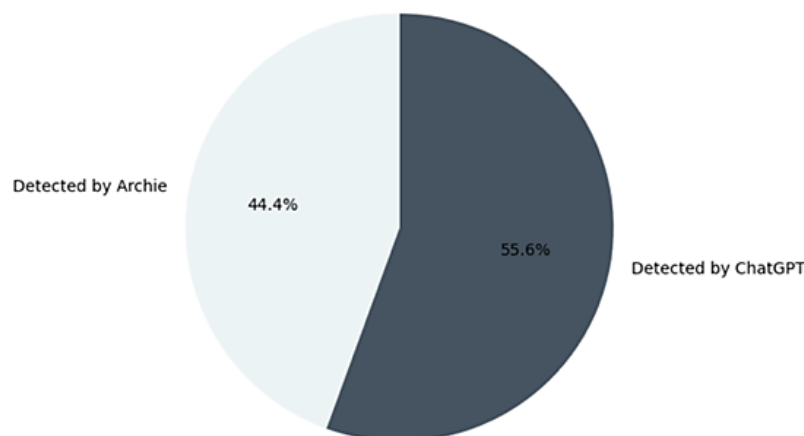


Figure 5. Patterns detection percentages for ChatGPT and Archie for Apache Flink

3.2. RQ2: how effective is ChatGPT in extracting architectural tactics from software systems source code?

To address this question, we used ChatGPT to identify tactics from the full codebase and compared its performance to the traditional tool Archie. We present the results of tactic detection in Apache Flink by both tools are presented. Table 2 provides a comparison of the tactics detected by Archie and ChatGPT, while Figure 6 illustrates the detection percentages. As shown, ChatGPT detected 25.0% of the tactics, whereas Archie achieved a higher detection rate of 75.0%. This is expected, as Archie was specifically designed to identify traditional tactics, whereas ChatGPT is more capable of discovering modern tactics.

Table 2. The comparison results of the tactics detection for Apache Flink

Architectural tactic	Also known as	Archie	ChatGPT
Kerberos		✓	✓
Heartbeat		✓	✓
Ping/Echo	Connectivity probe	✓	×
Exception handling	Error handling/fault handling	✓	×
Authenticate		✓	×
Time stamp		✓	×
Resource pooling	Resource sharing	✓	×
Audit trail		✓	×
PBAC	Policy-based access control	✓	×
RBAC	Role-based access control	✓	×
Resource scheduling	Task scheduling	✓	×
Session management		✓	×
Load balancing	Load management	✓	×
Restart	System reboot	✓	×
Time-out		✓	×
Cancel		✓	×
Active redundancy	Data duplication	✓	×
Checkpoint		✓	×
Retry		✓	×
Retry logic for fault tolerance		×	×
Data partitioning for scalability		×	✓
Resource pooling for efficient resource use	Resource sharing	×	✓
Circuit breaker for fault isolation		×	✓

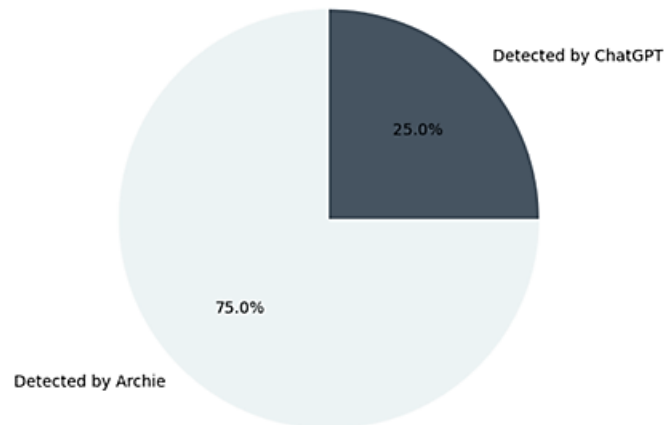


Figure 6. Tactics detection percentages for ChatGPT and Archie

Tables 3 and 4 present the results for TP, true negative (TN), FP, and FN for both Archie and ChatGPT. ChatGPT identified 5 patterns and 6 tactics, demonstrating its strength in detecting modern architectural patterns and tactics, such as fault-tolerance strategies. In contrast, Archie identified 4 patterns and 18 tactics, showcasing its proficiency in recognizing traditional architectural elements.

ChatGPT missed 5 patterns and 17 tactics (FN), underscoring its limitations in identifying foundational or traditional tactics. On the other hand, Archie missed 6 patterns and 5 tactics, highlighting its challenges in detecting modern or nuanced techniques. ChatGPT also falsely identified 4 patterns and 16 tactics (FP), whereas Archie falsely identified 5 patterns and 4 tactics. Since all patterns and tactics were

evaluated, TN are not applicable in this context. The results for the remaining systems will be available online (i.e., attached to the submission paper at the submission website).

Table 3. Metrics results of Archie

Archie detection metrics	Architectural patterns	Architectural tactics
TP	4	18
TN	0	0
FP	5	4
FN	6	5

Table 4. Metrics results of ChatGPT

ChatGPT detection metrics	Architectural patterns	Architectural tactics
TP	5	6
TN	0	0
FP	4	16
FN	5	17

4. THREATS TO VALIDITY

This section outlines the potential threats to the validity of the findings in this study. While the research explores the capabilities of ChatGPT in architectural analysis, certain limitations could impact the robustness, reliability, and generalizability of the results. These threats are categorized into three main types: construct validity, focusing on the design and measurement of the study; internal validity, addressing factors that could influence the interpretation of results; and external validity, concerning the applicability of findings to broader contexts. Each category highlights specific challenges and areas for improvement, ensuring a balanced evaluation of the study's strengths and limitations.

4.1. Construct validity

One of the threats of this work is that the study uses precision, recall, and accuracy to evaluate performance but omits metrics like F1-score, which could better balance the trade-off between precision and recall. We mitigate this threat by clearly define the calculation methods for each metric and ensure they align with standard practices in architectural analysis. Another threat is that the effectiveness of ChatGPT heavily depends on prompt quality, and variations in prompt design might influence results. The lack of detailed discussion about prompt optimization could affect reproducibility. We mitigate this by use a standardized prompt evaluation framework to ensure consistency and reproducibility. The also paper might not cover all possible architectural patterns and tactics comprehensively, potentially leading to biased results. We mitigate this threat by consult domain experts to ensure the selected patterns and tactics represent a comprehensive and balanced subset of architectural elements.

4.2. Internal validity

One of the internal threats of this work is that the choice of five open-source systems might not generalize to other types of software systems, limiting the scope of the findings. We mitigate this threat by selecting one project from different domain, so we cover most of the software engineering domains. The method for manually verifying TP and FP isn't explicitly detailed, leaving room for subjectivity and potential error. We mitigate this threat by judging what are TP and FP.

4.3. External validity

One of the generalizability threats is that the findings are based on specific open-source projects, and the results may not be applicable to proprietary or less-structured codebases. We have s future work to expand the study to include proprietary systems and unstructured codebases to assess generalizability. Other threat is that since the study evaluates ChatGPT (a specific LLM), the results may not generalize to other LLMs or AI-based tools for architectural analysis. We have another future work to evaluate the performance of other LLMs and AI-based tools to provide a broader perspective.

5. CONCLUSION

This work explored the effectiveness of ChatGPT, a modern LLM, in identifying architectural patterns and tactics within software systems, comparing it is performance to that of the traditional tool Archie. The findings highlight the unique strengths and limitations of both tools. ChatGPT demonstrated a strong capability in detecting modern architectural concepts, such as fault-tolerance strategies, reflecting its ability to adapt to evolving software practices. However, it struggled with foundational or traditional tactics, resulting in a lower detection rate in certain areas. In contrast, Archie excelled in identifying traditional architectural elements but showed limitations in addressing modern, nuanced techniques. These results underscore the complementary nature of ChatGPT and Archie in software design analysis. While Archie remains highly effective in its niche, ChatGPT's NLP capabilities provide significant potential for extending architectural analysis workflows, especially in contexts requiring a broader understanding of contemporary patterns and tactics. Future work could focus on enhancing ChatGPT's training data to improve its

recognition of traditional tactics, as well as integrating the capabilities of both tools to create a hybrid solution. Such advancements could pave the way for more intelligent, automated systems that bridge the gap between natural language understanding and software engineering tasks, ultimately contributing to improved software maintainability and scalability.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Hind Milhem	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	
Naderah Al-Jawabrah		✓		✓		✓		✓	✓	✓		✓	✓	
Raghad Abu Wadi	✓		✓	✓			✓			✓	✓		✓	

C : Conceptualization	I : Investigation	Vi : Visualization
M : Methodology	R : Resources	Su : Supervision
So : Software	D : Data Curation	P : Project administration
Va : Validation	O : Writing - Original Draft	Fu : Funding acquisition
Fo : Formal analysis	E : Writing - Review & Editing	

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

DATA AVAILABILITY

The data that support the findings of this study are openly available in 4TU Research Data at https://data.4tu.nl/private_datasets/MLZZc3Br6lMHHIG7vzCE9rca_vElivgINZjzC__X35U.

REFERENCES

[1] O. E. Olukunle and I. M. Oyerinde, "A review on software architectural patterns," *Global Scientific Journals*, vol. 9, no. 8, pp. 26–30, 2021.

[2] M. Kassab, M. Mazzara, J. Lee, and G. Succi, "Software architectural patterns in practice: an empirical study," *Innovations in Systems and Software Engineering*, vol. 14, no. 4, pp. 263–271, Dec. 2018, doi: 10.1007/s11334-018-0319-4.

[3] Z. Tu, "Research on the application of layered architecture in computer software development," *Journal of Computing and Electronic Information Management*, vol. 11, no. 3, pp. 34–38, Nov. 2023, doi: 10.54097/jceim.v11i3.08.

[4] G. Márquez, H. Astudillo, and R. Kazman, "Architectural tactics in software architecture: a systematic mapping study," *Journal of Systems and Software*, vol. 197, Mar. 2023, doi: 10.1016/j.jss.2022.111558.

[5] A. Mishra, Y. I. Alzoubi, and N. Gavrilovic, "Quality attributes of software architecture in iot-based agricultural systems," *Smart Agricultural Technology*, vol. 8, Aug. 2024, doi: 10.1016/j.atech.2024.100523.

[6] M. Mirakhorli, "Preserving the quality of architectural tactics in source code," M.S. thesis, College of Computing and Digital Media, DePaul University, Chicago, United States, 2014.

[7] M. Mirakhorli and J. C.-Huang, "Detecting, tracing, and monitoring architectural tactics in code," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 205–220, 2016, doi: 10.1109/TSE.2015.2479217.

[8] M. Mirakhorli, A. Fakhry, A. Grechko, M. Wieloch, and J. C.-Huang, "Archie: a tool for detecting, monitoring, and preserving architecturally significant code," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2014, pp. 739–742, doi: 10.1145/2635868.2661671.

[9] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, no. 3, pp. 3713–3744, Jan. 2023, doi: 10.1007/s11042-022-13428-4.

[10] M. U. Hadi *et al.*, "Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects," *TechXriv*, pp. 1-54, Nov. 2023, doi: 10.36227/techrxiv.23589741.

[11] R. Islam and O. M. Moushi, "GPT-4o: the cutting-edge advancement in multimodal llm," *TechXriv*, pp. 1-6, Jul. 2024, doi: 10.36227/techrxiv.171986596.65533294/v1.

[12] L. Zhao *et al.*, "Natural language processing for requirements engineering: a systematic mapping study," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–41, 2022, doi: 10.1145/3444689.

[13] R.-W. Bello and S. J. Tobi, "Software bugs: detection, analysis and fixing," *SSRN Electronic Journal*, 2024, doi: 10.2139/ssrn.4662187.




[14] L. Tan, "Using ChatGPT to extract design concepts from stories," in *The 7th International Conference for Design Education Researchers*, 2024, pp. 1-11, doi: 10.21606/drsldx2024.054.

[15] F. Gilson, M. Galster, and F. Georis, "Extracting quality attributes from user stories for early architecture decision making," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany, 2019, pp. 129–136, doi: 10.1109/ICSA-C.2019.00031.




- [16] S. Das, N. Deb, A. Cortesi, and N. Chaki, "Extracting goal models from natural language requirement specifications," *Journal of Systems and Software*, vol. 211, May 2024, doi: 10.1016/j.jss.2024.111981.
- [17] J. Huang *et al.*, "A critical assessment of using ChatGPT for extracting structured data from clinical notes," *npj Digital Medicine*, vol. 7, no. 1, May 2024, doi: 10.1038/s41746-024-01079-8.
- [18] Z. Sun, G. Pergola, B. C. Wallace, and Y. He, "Leveraging ChatGPT in pharmacovigilance event extraction: an empirical study," *arXiv-Computer Science*, pp. 1-14, Feb. 2024.
- [19] M. M. Mohajer *et al.*, "Effectiveness of ChatGPT for static analysis: how far are we?," in *AIware 2024-Proceedings of the 1st ACM International Conference on AI-Powered Software, Co-located with: ESEC/FSE 2024*, 2024, pp. 151–160, doi: 10.1145/3664646.3664777.
- [20] A. Terzi, S. Bibi, N. Tsitsimiklis, and P. Angelidis, "Using code from ChatGPT: finding patterns in the developers' interaction with ChatGPT," *Reuse and Software Quality (ICSR 2024)*, pp. 137–152, 2024, doi: 10.1007/978-3-031-66459-5_9.
- [21] H. Mahmoudi, D. Chang, H. Lee, N. Ghaffarzadegan, and M. S. Jalali, "A critical assessment of large language models for systematic reviews: utilizing ChatGPT for complex data extraction," *SSRN Electronic Journal*, 2024, doi: 10.2139/ssrn.4797024.
- [22] K. C. Pradhan, R. Slavin, S. Ghanavati, T. Breaux, and M. B. Hosseini, "An analysis of automated use case component extraction from scenarios using ChatGPT," *arXiv-Computer Science*, pp. 1-12, Aug. 2024.
- [23] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, "Towards human-bot collaborative software architecting with chatgpt," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 279–285.
- [24] M. H. Iqbal and T. R. Soomro, "Big data analysis: apache storm perspective," *International Journal of Computer Trends and Technology*, vol. 19, no. 1, pp. 9–14, Jan. 2015, doi: 10.14445/22312803/IJCTT-V19P103.
- [25] A. Katsifodimos and S. Schelter, "Apache flink: stream analytics at scale," in *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, IEEE, Apr. 2016, pp. 193–193, doi: 10.1109/IC2EW.2016.56.
- [26] E. Shaikh, I. Mohiuddin, Y. Alufaisan, and I. Nahvi, "Apache spark: a big data processing engine," in *2019 2nd IEEE Middle East and North Africa COMMUNICATIONS Conference*, IEEE, Nov. 2019, pp. 1–6, doi: 10.1109/MENACOMM46666.2019.8988541.
- [27] T. Berglund and M. McCullough, *Building and testing with gradle: Understanding next-generation builds*. Sebastopol, California: O'Reilly Media, 2011.
- [28] M. Jesick *et al.*, "MAVEN navigation overview," *Advances in the Astronautical Sciences: Spaceflight Mechanics 2016*, vol. 158, pp. 1235–1254, 2016.

BIOGRAPHIES OF AUTHORS






Hind Milhem    holds a Doctor of Software Engineering degree from Ottawa University, Canada in 2020. She also received her B.Sc. and M.Sc. (Computer Science) from Yarmouk University, Jordan in 2007 and 2009, respectively. She is currently an assistant professor at Department of Information in Faculty of IT at The Hashemite University, Zarqa, Jordan. Her research includes software architecture design, architectural patterns, architectural tactics, software evaluation, and artificial intelligence (AI). She published six papers in international journals and conferences. She has also five papers under processing- waiting for the final decision of their acceptance. From 2019 till now, she is working as a reviewer (PC member) at the International Journal of Software Engineering and Knowledge Engineering (IJSEKE) and SEKE conference. She can be contacted at email: hinda_is@hu.edu.jo.



Naderah Al-Jawabrah    holds a Doctor of Software Engineering degree from Szeged University, in 2021. She also received her M.Sc. (Software Engineering) from the Hashemite University, Jordan in 2015 and she received her B.Sc. (Computer Science) from Yarmouk University, Jordan in 2004. She is currently an assistant professor at Department of Information in Faculty of IT at The Hashemite University, Zarqa, Jordan. Her research includes software testing, artificial intelligence, and machine learning. She can be contacted at email: naderam@hu.edu.jo.



Raghad Abu Wadi    holds her B.Sc. degree in Business Information Technology from the Hashemite University, Jordan in 2024. She is currently working as a researcher in machine learning and artificial intelligence topics. She can be contacted at email: raghad.abuwadi@gmail.com.