Architecture Exploration and Reflection meet LLM-based Agents

J. Andrés Diaz-Pace*†, Antonela Tommasel*, Rafael Capilla‡ and Yamid E. Ramírez‡§
*CONICET & UNICEN University, Argentina Email: {*andres.diazpace, antonela.tommasel*}*@isistan.unicen.edu.ar*
† Data & IA Studio, Globant, Argentina
‡Rey Juan Carlos University, Spain Email: *rafael.capilla@urjc.es*
§Institución Universitaria Politécnico Grancolombiano, Colombia Email: *yeuramirez@poligran.edu.co*

***Abstract*—The exploration of architecture alternatives is an essential part of the architecture design process, in which designers search and assess solutions for their requirements. Although automated tools and techniques have been proposed for this process, they still face adoption challenges. Nowadays, the emergence of generative AI techniques creates an opportunity for leveraging natural language representations in architecture design, particularly through LLM-based agents. To date, these agents have been mostly focused on coding-related tasks or requirements analysis. In this work, we investigate an approach for defining design agents, which can autonomously search for architectural patterns and tactics for a particular system and requirements using a textual format. In addition to incorporating architectural knowledge, these agents can reflect on the pros and cons of the proposed decisions, enabling a feedback loop towards improving the decisions' quality. We present a proof-of-concept called `ReArch` that adapts elements from the *ReAct* and *LATS* agent frameworks, and discuss initial results of applying our LLM-based agents to a case study considering different patterns.**

I. Introduction

The architecture design process involves analysis, synthesis, and evaluation activities, which are typically performed iteratively, until establishing a design solution that satisfies the system requirements. During architecture synthesis, two key tasks are: i) exploring design alternatives using patterns, styles, or tactics; and ii) evaluating these alternatives against the requirements. Although automated approaches have been proposed for these tasks [1, 2], many of them rely on formal specifications of requirements and architectural models to leverage search methods (e.g., evolutionary algorithms, model checkers) and quality-attribute solvers. Other works have framed architecture synthesis as a planning problem. While these approaches have shown promise, their reliance on formal representations presents practical adoption challenges.

The emergence of LLMs (Large Language Models) coupled with agents [3, 4] has brought new opportunities for assistive techniques and tools, as showed by recent developments for software engineering tasks. Agents for code assistance are a typical example of this trend. This technology has demonstrated the ability to start with a goal, plan a series of tasks to achieve it, and leverage information sources for completing them. In the architecture domain, LLM-based tools are scarce. Some experiments using LLMs for analyzing requirements and co-creating designs have been reported [5, 6]. These efforts primarily rely on standalone LLMs (e.g., *ChatGPT* or *Copilot* prompts for specific tasks) [7], but are still behind in terms of the capabilities exhibited by code agents. Thus, there is a challenge on how to evolve architecture tools towards *design agents*, which can take advantage of (existing) architectural knowledge and autonomy to make architects more productive.

In previous work [8], we proposed a research prototype called `ArchMind` to help (novice) architects make better design decisions. `ArchMind` consists of various LLM-based modules that can assist a human in searching for candidate patterns for requirements, analyzing their pros/cons, and semi-automatically capturing decisions. Although our initial results were encouraging, these modules do not operate autonomously, requiring human commands to perform each search and evaluation task. This limitation precludes `ArchMind` from performing "chains" of reasoning that multiple patterns and assess them until returning the most suitable option. To address this issue, we present an enhanced approach called `ReArch`, which adapts LLM-based planning, reflection, and tree-of-thought techniques to enable an more autonomous architecture exploration. `ReArch` follows a generate-and-test strategy [9] in which an agent relies on tools and prompting instructions to generate patterns for a given textual requirement, evaluate their suitability to the system context, and finally recommend a ranking of design decisions.

The proposed approach is a novel *research idea* for automated architecture design, which retains the characteristics of existing (manual) design processes, but leverages the reasoning and generative capabilities of LLMs. A motivation for this idea is to increase the quality of the critical analysis and decision-making processes that architects typically engage in when discussing solution alternatives [10]. Thus, we aim to test to what extent LLM-based technologies can assist architects in this process. The added value of design agents over traditional architecture tools is their ability to take textual inputs and support verbal interactions and reasoning. By emulating human-like problem-solving strategies for design, these agents can be more accessible to architects with varying levels of expertise, reducing the barriers of formal representations or specialized technical knowledge and improving the quality of decisions.

II. Background: Agents and LLMs

Agents are software entities that can autonomously perceive and act on a surrounding environment to achieve specific goals,

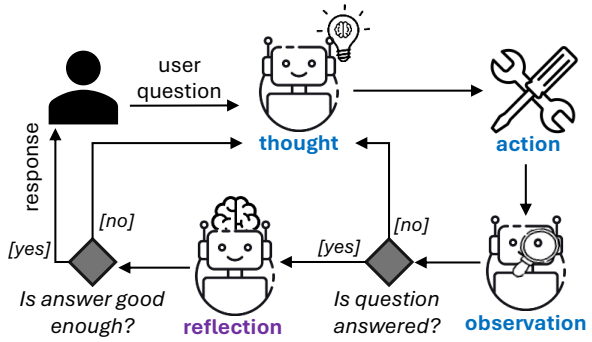

Fig. 1. *ReAct* workflow including reflection at the end of the cycle.

often using reasoning techniques (e.g., planning) to select appropriate actions. While the agent concept is not new, the generative capabilities of LLMs have opened new possibilities for building agents. In an LLM-based agent, the LLM acts as the agent's controller that drives decision-making.

An agent typically consists of three components: memory, tools, and a decision-making mechanism. The memory serves as a storage that tracks the agent's tasks to support in-context learning. Tools represent the actions available to the agent, such as functions for gathering information from the environment or interacting with it. An action is often a function that takes input parameters and returns an output, e.g., a search API or a query to a database. Finally, the decision-making mechanism is the logic that orchestrates both LLM calls and tools via specialized prompting techniques. *ReAct* and *LATS* [11, 12] are frameworks implementing such techniques.

A. The ReAct Framework

LLMs with adequate instructions (e.g., chain-of-thought prompting) have shown the ability to perform multi-step reasoning to answer questions. To ground this reasoning using external information, agents can also leverage domain-specific tools and use a controller to select and execute them.

The *ReAct* framework [11] is a planning process that combines *reasoning and acting*, enabling the design of flexible autonomous agents. The tools are a key *ReAct* feature that define the agent's available actions, accompanied by textual rules specifying when and how each tool should be used. The framework prompts the LLM to generate verbal reasoning traces for a task, as depicted in Fig. 1. A trace (or task trajectory) consists of one or more reasoning steps. Once a user question is submitted, each step begins with a thought, which drives an action (i.e., a tool execution), and in turn triggers an observation about the action's outcome. Based on this observation, the agent can either stop and return an answer or initiate a new reasoning cycle. This process continues until an answer is found or a maximum number of cycles is reached.

Fig. 2 exemplifies a *ReAct* trajectory for a scenario in which a user wants to compare the price of a laptop in the US and EU markets. For its task, the agent relies on two tools: a price checker and a currency converter. We argue that analogous scenarios can be exercised in the architecture domain, if an agent is equipped with adequate design-oriented

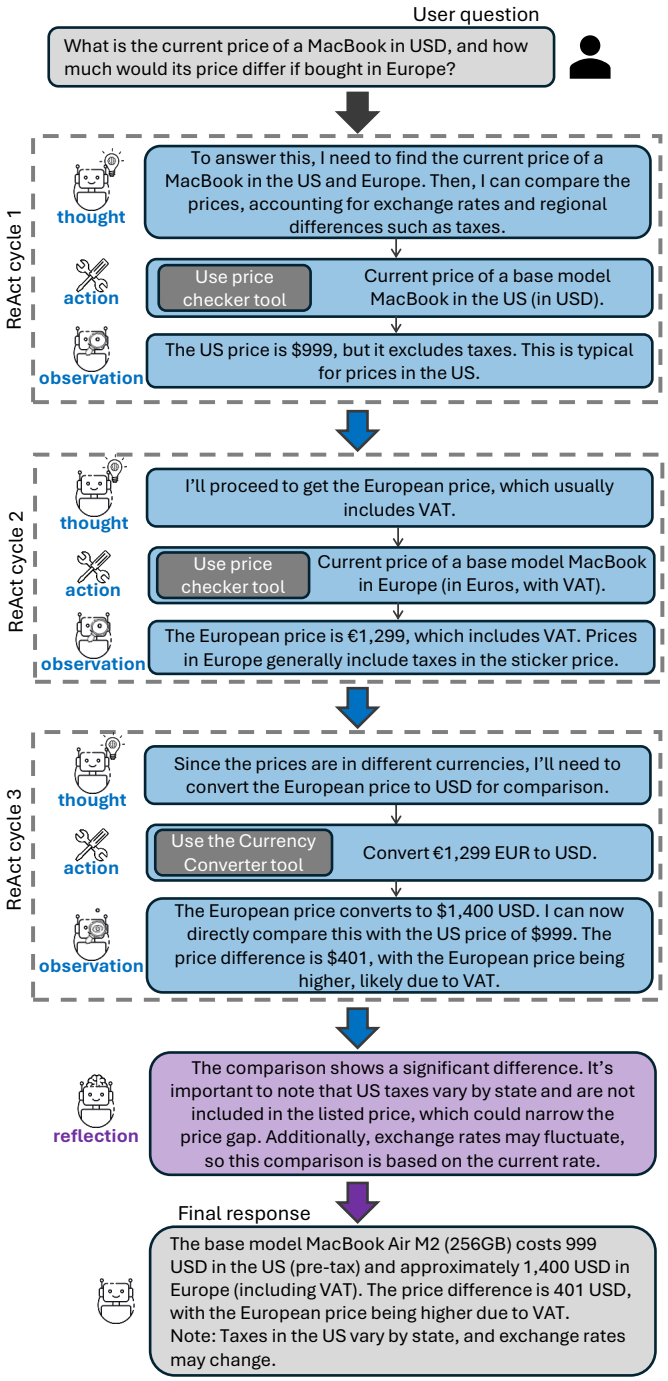

Fig. 2. A *ReAct* trajectory involving tools, reasoning cycles and reflection.

tools and instructions. In fact, the `ArchMind` modules [8] can be wrapped as tools and orchestrated in a *ReAct* cycle to explore patterns for a given requirement and system context.

Although a *ReAct* agent can generate trajectories and successfully answer questions, its response quality varies, and refining its reasoning or exploring alternative paths is challenging. One technique to address this issue is through verbal reinforcement learning, known as *reflection* [13], which often consists of a textual summary providing a concrete feedback for the LLM to improve upon. The reflection is intended

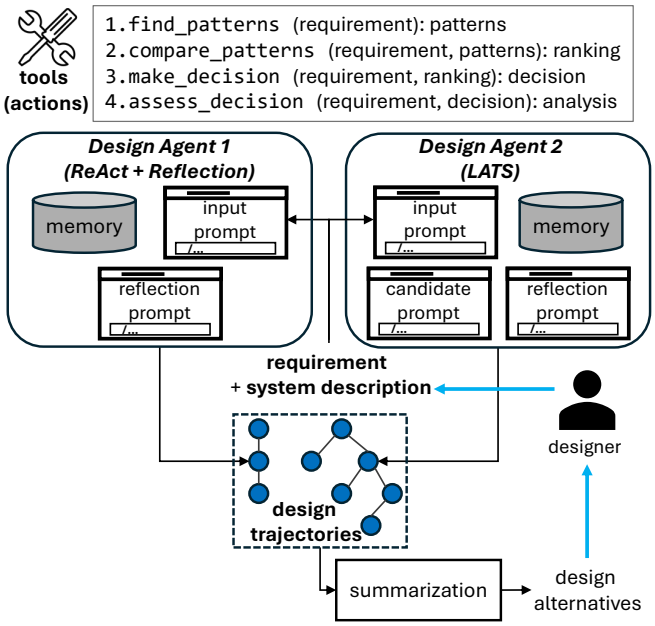

Fig. 3. LLM-based Agents for exploration of architecture alternatives.

to evaluate the response quality and is stored in the agent's memory. In our example in Fig. 2, the reflection happens at the end of the third cycle, Often, after being exposed to a reflection, a new *ReAct* cycle can trigger alternative thoughts leading to additional answers.

In general, implementing a reflection summary requires tailored LLM instructions based on the problem domain. Specifically, in the architecture domain, a reflection can assess whether a requirement is fully met by a design decision, check the decision quality or identify quality-attribute tradeoffs, potentially triggering the exploration or subsequent decisions.

B. The LATS Framework: Monte Carlo Tree Search

Augmenting LLMs with observations and feedback (i.e., reflection) enhances the reasoning and acting capabilities of a *ReAct* agent. However, these agents face limitations when tackling questions that require multiple reasoning chains to produce a response. Once a *ReAct* trajectory is established, the LLM-based controller tends to follow a reasoning chain in a predetermined direction, with limited flexibility to explore alternative directions. While repeated *ReAct* cycles (aided with reflection) can introduce variations, this is often insufficient for problems that benefit from a comprehensive search of the solution space. For example, in the laptop comparison scenario, the agent could consider additional factors influencing prices. as alternative thoughts.

Supporting multiple reasoning chains transforms a trajectory into a branching structure, resulting in a tree that allows a deeper and more diverse exploration. Language Agent Tree Search (*LATS*) [12] is a framework that enables the generation of multiple *ReAct* trajectories organized as a tree. This tree is iteratively expanded using a Monte Carlo Tree Search (MCTS) strategy, which balances the exploration and exploitation of solution options by generating one or more candidate thoughts.

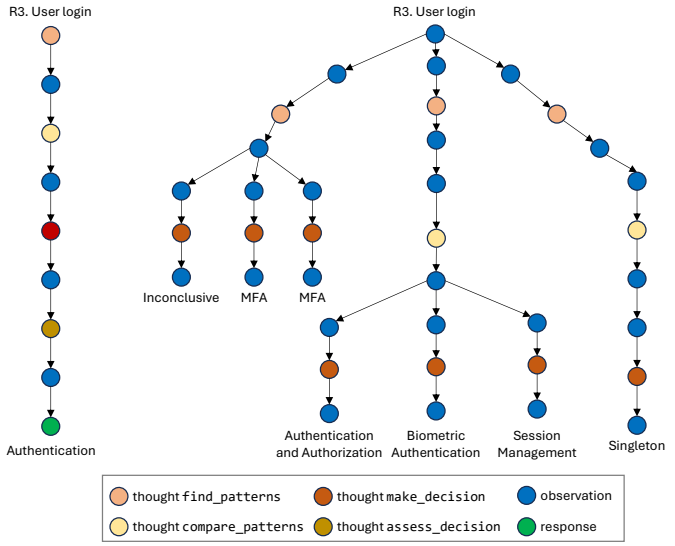

Fig. 4. Examples of design trajectories for a requirement and the corresponding patterns for *ReAct* (left) and *LATS* (rigth)

LATS also integrates reflection into the MCTS process to compute the reward of each node. A reward represents a measure of success in a (partial) trajectory, providing a reinforcement signal to explore promising nodes of the tree. Both node expansion and reflection mechanisms are implemented through LLM instructions, which can be tailored to specific domains for improved performance.

LATS nicely models a human-like reasoning process in which alternative architectural scenarios are explored and evaluated. These scenarios can refer to design decisions, assumptions, or technology selection. For instance, the agent can try to apply different patterns for a requirement, or perform quality-attribute analyses by changing its design assumptions.

III. ReArch: Agents for Architectural Reasoning

To investigate the role of LLM-based agents in architecture design, we developed the `ReArch` approach, which adapts *ReAct* and *LATS* agents with design-specific prompts and tools, as illustrated in Fig. 3. The proof-of-concept was implemented using *LlamaIndex* with *GPT-4o-mini* as the underlying LLM[1].

Our design agents take as input a textual requirement for exploring architecture alternatives and a brief description of the system related to that requirement. The user-provided inputs are processed through an input prompt. Each agent can plan multiple *ReAct* trajectories, each resulting in a design solution for the requirement. Fig. 4 shows examples of design trajectories for a requirement and the corresponding patterns. The *ReAct* agent computes separate, sequential trajectories, while the *LATS* agent computes all its trajectories as part of a tree. A reflection evaluates the quality of each solution and assigns it a score. Once the agent search is finished, the solutions are summarized, ranked and presented to the user.

We built the tools from the prompts used by the `ArchMind` agents. Furthermore, we tailored the *ReAct* and *LATS* prompts

[1]Additional details including the artifacts used for the case study can be found in the companion repository: https://github.com/tommantonela/ReArch.

to generate candidate thoughts and reflections with a focus on architectural considerations. The current tools work as follows:

1. `find_patterns`. Identifies candidate patterns relevant to an input requirement, returning a list of pattern names. This tool aims at retrieving a broad set of design solutions.

2. `compare_patterns`. Evaluates the pros and cons of each pattern in an input list with respect to a requirement. The patterns are ranked (and filtered out) based on their suitability.

3. `make_decision`. Selects a suitable pattern from a ranking and instantiates it into a concrete design decision.

4. `assess_decision`. Conducts an in-depth evaluation of a decision, analyzing risks, tradeoffs and related decisions.

Given a requirement, the agent should ideally execute its tools in a logical order, although the LLM may adjust the order based on the current trajectory. The candidate generation prompt produces one or more thoughts depending on the last tool used in the trajectory. The best opportunities for branching occur after using the `find_patterns` and `compare_patterns` tools, as these enable the agent to explore alternative decisions, assessments, and comparisons by invoking the remaining tools. The reflection prompt evaluates the trajectory's steps (i.e., decisions) against three criteria: correctness, completeness, and degree of requirement satisfaction. These criteria relied on an *LLM-as-a-judge* mechanism.

IV. Initial Evaluation

We conducted an evaluation using the `CampusBike` case study [6], which develops a software mobility application. This case study includes functional requirements (e.g., 'user registration', 'view available bikes', 'reserve a bike', 'make payments', and 'view usage reports') as well as quality attributes (e.g., usability, performance, security, localization, availability, scalability, compatibility, integration, accessibility, reliability). The solution [6] defines a microservices architecture, in which design patterns are applied for each microservice. We refined the base requirements to include quality-attribute aspects and feed them into our two agents to obtain alternative decisions. Our goal was to compare the alignment or overlap between the original architecture [6] and the patterns generated by the *ReAct* and *LATS* frameworks.

Table I summarizes the results of the experiment[2]. The two rows at the bottom show the patterns included in the original solution. We can see that both agents generated many valid patterns related either to microservices or object-oriented design. As expected, *LATS* offered richer design alternatives than *ReAct* and more points of view in its analysis. In the trajectories, the *ReAct* agent was prone to execute the four tools in the prescribed order, while the *LATS* agent often alternated between the tools for pattern comparison and decision making, disregarding the decision assessment tool. On the downside, *LATS* sometimes returned duplicate patterns, which resulted in a higher inference work for the LLM. In the decision ranking, we also noticed that some options were not patterns or tactics from the literature (i.e., hallucinations).

[2]The agents provide more detailed decisions than the pattern names, but those are omitted due to space constraints.

TABLE I
COMPARISON OF DESIGN DECISIONS GENERATED BY REARCH FOR EACH REQUIREMENT. IN **BOLD** THE PATTERNS INCLUDED IN THE OBJECT-ORIENTED DESIGN, AND UNDERLINED THE PATTERNS RELATED TO THE APPLICATION MICROSERVICES

	Patterns *ReAct*	Patterns *LATS*
R1. User registration	User registration. End-to-end encryption.	Secure communication. Authentication and authorization.
R2. Verification code	Timeout management.	**Event-driven.** Timeout management.
R3. User login	Authentication. Biometric Authentication	Authentication and authorization. Biometric Authentication. Session management. MFA. **Singleton**.
R4. Automatic logout	Idle session timeout. Session management.	Session management. Automatic logout. **Singleton**. **Observer**.
R5. Bike proximity search	Location-based services.	Location-based services. Geofencing. **Event-driven**.
R6. Navigation and bike registration	Facade.	Command.
R7. Bike availability updates	Event sourcing.	**Strategy**. **Observer**. Visitor. Command. State.
R8. Payment options	Payment gateway.	Payment gateway. **Event-driven**.
R9. Rental time notifications	**Observer**.	**Observer**. Publish-Subscribe. State management. Payment management.
R10. Rental durations	**Strategy**.	**Strategy**. State management.
R11. Usage history	CQRS.	CQRS. Caching. Load-balancing. **Observer**. Access control.
R12. Bike overview for administrators	CQRS.	Observer. CQRS. Publish-subscribe. Real-time data streaming.
R13. Usage analytics for administrators	**Observer**.	Application metrics. Data aggregation. Monitoring and logging. Distributed tracing. Analytics and reporting.
Microservices architecture: Authentication, User management, Bike inventory, Payment gateway, Location, Notification, and Reporting, CQRS (Command Query Responsibility Segregation)		
Design patterns: Singleton, Factory Method, Observer, Strategy, Decorator, Event-driven.		

Discussion: Our experiments revealed both opportunities and challenges in applying LLMs to software architecture. Agent features such as multi-step reasoning and reflection enable a structured and rich analysis of alternative decisions, enhancing their rationale for the target requirement or system. The natural language inputs and outputs also makes architecture tools accessible to a wider range of practitioners. To reap the benefits of agent-generated recommendations, however, the tools must be grounded in appropriate architectural knowledge. Dealing with issues like hallucinations and overly general recommendations is stil challenging. Further validation through real-world experiments and metrics is crucial to strengthen the reliability and usefulness of the recommended decisions.

V. Related Work

Although the role of LLMs to deal with software engineering problems has been extensively discussed, few approaches have explored their application in solving design problems

and managing architectural knowledge. In this regard, Dhar et al. [14] and Eisenreich et al. [15] focused on using simple LLM prompting strategies to assist architects by suggesting, evaluating and documenting design alternatives. Ozkaya [16] focused on the support of generative AI for architecture development and refinement. Ahmad et al. [6] used *ChatGPT* for mimicking architect roles in collaborative design activities, including analysis, synthesis, and evaluation, to generate a UML design. Donakanti et al. [17] proposed using LLMs to generate context-based strategies for self-adaptive systems.

In summary, these works reinforce the potential of LLMs for addressing design challenges. Nonetheless, in general, these approaches rely on standalone LLMs for specific tasks, and do not yet take advantage of agent techniques. Our approach, in turn, proposes a holistic perspective by integrating LLMs and agentic worflows into the architectural reasoning process.

VI. Future Plans

Based on our preliminary findings, we envision several lines of work for the `ReArch` approach. First, we plan to incorporate points for user interventions, so that an agent can run autonomously for a while but request designer's clarifications when necessary. An iterative collaboration between agents and the architect can promote reflective practices [10] to critique a current design and lead to quality decisions. A second idea is to consider multiple LLM-based agents, each one tackling particular requirements and coordinating their trajectories to achieve different tradeoffs. Along this line, we will investigate how the agent architecture can support ATAM-like evaluations. Third, we will evolve the *LATS*/*MCTS* framework to better capture the activities, inputs and outputs of existing design methods (e.g., Attribute-Driven Design), using an agentic workflow and different knowledge sources as part of an automation process. In this process, the LLM-based mechanisms can be integrated via tools with existing search engines and analytical solvers for software architectures. At last, to reduce hallucinations and enable a better grounding on architectural concepts (e.g., decisions, quality attributes, patterns), we will explore advanced RAG (Retrieval Augmented Generation) techniques for the agent tools and memory.

VII. Conclusion

Although using LLM-based agents in the software architecture field is still in early stages, we argue that this technology has the potential to improve the quality, rationale and consistency of humans' decisions. Agents can support expert architects by amplifying their exploration of (complex) design spaces, or help junior architects enhance their design skills.

In this work, we investigate a combination of agents, prompting frameworks and tools to simulate an automated exploration of design alternatives and also incorporate reflection in the decision-making process. Our experience with `ReArch` shows that, despite its technical challenges, the approach seems to generate reasonable results. Experiments to validate these findings against architectural knowledge sources and test the agents with human users are underway.

Acknowledgements. J. A. Diaz-Pace and A. Tommasel were supported by project PICT-2021-00757, Argentina.

References

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software architecture optimization methods: A systematic literature review," *IEEE Trans. on Soft. Eng.*, vol. 39, no. 5, pp. 658–683, 2013.

[2] D. Arcelli, V. Cortellessa, M. D'Emidio, and D. Di Pompeo, "Easier: An evolutionary approach for multi-objective software architecture refactoring," in *2018 IEEE Int. Conf. on Software Architecture (ICSA)*, 2018, pp. 105–115.

[3] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, "A survey on large language model based autonomous agents," *Fronts. of Comp. Science*, vol. 18, no. 6, p. 186345, Mar 2024.

[4] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024.

[5] M. A. Sami, M. Waseem, Z. Zhang, Z. Rasheed, K. Systä, and P. Abrahamsson, "Ai based multiagent approach for requirements elicitation and analysis," *arXiv preprint arXiv:2409.00038*, 2024.

[6] 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.

[7] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*. Cham: Springer Nature Switzerland, 2024, pp. 71–108.

[8] J. A. Díaz-Pace, A. Tommasel, and R. Capilla, "Helping novice architects to make quality design decisions using an llm-based assistant," in *European Conference on Software Architecture*. Springer, 2024, pp. 324–332.

[9] L. Bass, "Generate and test as a software architecture design approach," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, 2009, pp. 309–312.

[10] M. Razavian, A. Tang, R. Capilla, and P. Lago, "Reflective approach for software design decision making," in *2016 QRASA*. IEEE, 2016, pp. 19–26.

[11] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[12] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang, "Language agent tree search unifies reasoning acting and planning in language models," *arXiv preprint arXiv:2310.04406*, 2023.

[13] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *NeurIPS*, vol. 36, 2024.

[14] R. Dhar, K. Vaidhyanathan, and V. Varma, "Can llms generate architectural design decisions?-an exploratory empirical study," *arXiv preprint arXiv:2403.01709*, 2024.

[15] T. Eisenreich, S. Speth, and S. Wagner, "From requirements to architecture: an ai-based journey to semi-automatically generate software architectures," in *Proceedings of the 1st International Workshop on Designing Software*, 2024, pp. 52–55.

[16] I. Ozkaya, "Can architecture knowledge guide software development with generative ai?" *IEEE Software*, vol. 40, no. 5, 2023.

[17] R. Donakanti, P. Jain, S. Kulkarni, and K. Vaidhyanathan, "Reimagining self-adaptation in the age of large language models," *arXiv preprint arXiv:2404.09866*, 2024.