Reimagining Self-Adaptation in the Age of Large Language Models

Raghav Donakanti, Prakhar Jain, Shubham Kulkarni, Karthik Vaidhyanathan
Software Engineering Research Center, *IIIT Hyderabad*, India
raghav.donakanti@students.iiit.ac.in, prakhar.jain@research.iiit.ac.in, shubham.kulkarni@research.iiit.ac.in,
karthik.vaidhyanathan@iiit.ac.in

2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C) | 979-8-3503-6625-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICSA-C63560.2024.00036

***Abstract*—Modern software systems are subjected to various types of uncertainties arising from context, environment, etc. To this end, self-adaptation techniques have been sought out as potential solutions. Although recent advances in self-adaptation through the use of ML techniques have demonstrated promising results, the capabilities are limited by constraints imposed by the ML techniques, such as the need for training samples, the ability to generalize, etc. Recent advancements in Generative AI (GenAI) open up new possibilities as it is trained on massive amounts of data, potentially enabling the interpretation of uncertainties and synthesis of adaptation strategies. In this context, this paper presents a vision for using GenAI, particularly Large Language Models (LLMs), to enhance the effectiveness and efficiency of architectural adaptation. Drawing parallels with human operators, we propose that LLMs can autonomously generate similar, context-sensitive adaptation strategies through its advanced natural language processing capabilities. This method allows software systems to understand their operational state and implement adaptations that align with their architectural requirements and environmental changes. By integrating LLMs into the self-adaptive system architecture, we facilitate nuanced decision-making that mirrors human-like adaptive reasoning. A case study with the SWIM exemplar system provides promising results, indicating that LLMs can potentially handle different adaptation scenarios. Our findings suggest that GenAI has significant potential to improve software systems' dynamic adaptability and resilience.**

***Index Terms*—Self-Adaptation, Software Architecture, Generative AI, LLM, Software Engineering**

I. Introduction

Software systems face uncertainties, such as component failures and varying workloads, affecting their Quality of Service (QoS). The concept of autonomic computing, as proposed by Kephart and Chess [1], sought to enhance the autonomy of software systems and mitigate these uncertainties through various strategies. Over the years, self-adaptation has emerged as a potential solution to mitigate these problems [2].

As of late, with advancements in computing and AI in general, Machine Learning (ML) has emerged as a key solution for self-adaptation [3], utilizing large datasets to identify patterns and predict future states, thus enabling dynamic structural/behavioural adjustments. Despite its benefits, ML's application in self-adaptation requires extensive data and a thorough understanding of ML principles. The rise of Generative AI (GenAI), primarily through Large Language Models (LLMs), presents new avenues in this field. These models have revolutionized how machines understand and generate human-like text, transcending simple language tasks, thus demonstrating potential applications in diverse areas, especially for complex decision-making given adequate context [4], [5].

Given that software and systems are fundamentally defined and controlled through code, APIs, JSON, XML, etc. and also generate reports, logs, metrics, etc, forms of language, LLMs offer an intriguing solution to the challenge of self-adaptation. By leveraging their capacity to understand and generate various language formats, LLMs could potentially interact with and adapt software systems in real-world dynamic environments.

To this end, this paper introduces an innovative framework, MSE-K (Monitor, Synthesize, Execute) that integrates Large Language Models (LLMs) into the self-adaptation process, enabling software systems to autonomously generate and implement contextually relevant and actionable architectural adaptation strategies aligned with their operational goals. This approach seeks to overcome the limitations of current self-adaptation mechanisms, paving the way for more efficient and intelligent software systems [6]. As an initial validation, we performed evaluations of our approach on SWIM [7], an exemplar for simulating web infrastructure through well-defined interfaces for monitoring runtime metrics and executing adaptation strategies. Our initial results indicate that incorporating LLMs into self-adaptive systems has immense potential to improve the self-adaptability of software systems and thereby guarantee performance in the event of uncertainties.

II. Related Works

The integration of machine learning into self-adaptive systems has marked a significant advance in the field of software engineering. Gheibi et al. [3] have highlighted the use and key challenges of engineering ML-based adaptive systems. Approaches based on reinforcement learning for planning and adapting software systems were introduced in [8] and [9]. [10] combines ML for adaptation pattern selection and quantitative verification for decision feasibility. [6] emphasizes the bidirectional relationship between self-adaptation and AI to manage unanticipated changes. [11] proposes a novel approach to self-adaptation by integrating a lifelong ML layer into self-adaptive systems that use ML techniques, enabling dynamic tracking and updating of their learning models. This paper builds on existing research on integrating AI into self-adaptive systems by exploring the transformative potential of Generative AI,

2768-4288/24/$31.00 ©2024 IEEE
DOI 10.1109/ICSA-C63560.2024.00036

especially Large Language Models (LLMs), for managing uncertainties and devising adaptive strategies. It demonstrates LLMs' potential to improve decision-making in architectural adaptation.

III. LLM-Based Architecture for Self-Adaptation

The approach, as represented in Fig. 1, conforms to the conceptual model of a self-adaptive system. It consists of two subsystems, namely the *Managed System* and *Managing System*. *The Managed System* represents the operational software system equipped with probes for ongoing QoS monitoring and effectors for implementing adaptations.

The Managing System is responsible for handling the adaptations. It consists of different components that implement the adaptation control loop, similar to MAPE-K [1]. However, in a traditional MAPE-K-based control loop of adaptation, the *analyze* phase is responsible for constantly analyzing various aspects of the system to identify any possible deviations from the adaptation goals. Such deviation triggers the *plan* phase, which generates adaptation decisions. In our approach, the analyze and plan phases are abstracted by the *Synthesize* phase, which leverages LLM to interpret and generate adaptation decisions. This stems from the fact that modern-day LLMs have the ability to interpret information and further generate adaptation decisions. Hence, we envision an MSE-K loop as represented in Fig. 1. The rest of the section details the different components of the managing system.

Monitor: The monitor as in the case of the traditional MAPE-K approach is responsible for continuously monitoring the running software system. In our approach, the monitor activity is exploited to monitor and collect the system logs as well as the QoS metrics of the software system. To this end, it consists of two components *logs collector*, which is responsible for collecting the system logs and a*metric collector*, which collects different QoS metrics like response time, CPU utilization, etc. These data together form the context data, C, that represents the running state of the system at a given instant of time t. Further, these data are ingested into the Knowledge base for further processing and future use.

For instance, with respect to our case study on SWIM, the context data, C is a set, such that $C = \{D_t, AS_t, U_t, R_t, AR_t, t\}$ where D_t, AS_t, U_t, R_t *and* AR_t represent dimmer value, active servers, server utilization. average response time and arrival rate at time t respectively.

Knowledge/Memory: The Knowledge acts as a central storage for different types of knowledge required by different components of the managing system for performing the adaptations. It stores four types of information: *i) Conversation History* which comprises tuples of past context data (C) and adaptation decision (AD) pairs. Hence the conversation history at timestep t, $H_t = [(C_1, AD_1), (C_2, AD_2) \dots (C_t, AD_t)]$. The conversation history is updated with C_t and AD_t at every execution of the *Synthesize* step. *ii) Fine-tuned models* consist of a set of LLMS that can be used for different tasks by leveraging the notion of multi-agents [5], where each LLM can be tasked with a role and responsibility. For instance, one

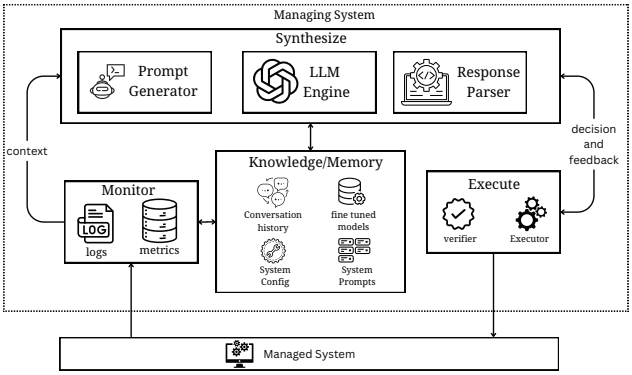

Fig. 1. Approach

LLM agent can be responsible for gathering insights from logs; another LLM agent can be responsible for interpreting and inferring information from system metrics and so on. These agents, as and when needed, can be leveraged by the *LLM Engine* in the Synthesize component; *iii) System Config* where the different configurations of the system are stored, such as the number of compute resources that are available, other hardware information, etc; *iv) Prompts* Consists of different system prompts that provide the LLM with instructions on how to interpret the various data and further how to generate decisions along with the objective, O. They act as guidelines to ensure that the LLMs generate decisions that are relevant to the given context of a system and its objectives. Further, it also consists of a set of few-shot prompts which provide the LLM with a few examples of adaptation scenarios and the corresponding actions that need to be taken in such scenarios. These prompts further allow the LLM to learn how to generate an adaptation decision given a context. This also ensures that the LLM is more consistent when generating decisions.

For instance, in the case of SWIM, we utilize *Prompts* and *Conversation History*. Our *Prompts*, P_{SWIM} (refer Figure 3), consist of a description of what an adaptation manager is and what actions the LLM will be capable of performing, which contextualizes the LLM in our setting. This is paired with the objective the LLM needs to optimize for, followed by a brief description of various metrics in C (refer Figure 2). Finally, a few-shot examples are provided.

Synthesize: The Synthesize activity encompasses the Analyze and Plan phase of the traditional MAPE-K loop by interpreting the output of the *Prompt Generator*, P to generate the adaptation decision which is sent to the execute phase. It consists of three major components:

i) Prompt Generator: It is responsible for integrating the context, C into the conversation history, H. It then compiles H and other long-term data stored in *Knowledge* into a single prompt, P. Overall the prompt generator pulls together everything needed for the LLM to generate a decision, $P = \{C, H, \textit{long term data}\}$ Sometimes P can lead to the LLMs context length being exceeded. This occurs because of how the transformer architecture can only process a finite

C and AD

- C
  Status:
  - D_t: 0.8
  - AS_t: 2.0
  - U_t: 0.89261
  - R_t: 0.35951825050096935
  - AR_t: 42.9667
  - t: 3000
  - max_servers: 3.0

  Actions you can take:
  1. Set Dimmer (A_1)
  2. Add Server (A_2)
  3. Remove Server (A_3)
  4. Do Nothing (A_4)

- AD: 1 0.6

Fig. 2. Example of C and AD in SWIM

P_{SWIM}

You are an adaptation manager for a server system handling user requests. Self-adaptive systems are...O...You being the adaptation manager are responsible for modifying the runtime behavior of the system...decide if adaption is required and what type of adaptation is needed. *Terminologies* Here are a few examples on how you are to interact with the system...*Few Shot*

- O: Here the primary system objective is to keep the average response time as low as possible. Secondary objective is to keep dimmer as high as possible. Tertiary objective is to keep servers used as less as possible. These objectives are given in decreasing order of priority.
- *Terminologies*: Brief description of metrics in C. Metrics are dimmer,active servers, max servers, utilization, average response time, arrival rate, and time elapsed.
- *Few Shot*: $\{(C_{s1}, AD_{s1})...\}$

Fig. 3. Example of P_{SWIM}

number of tokens at once. To this, the *Prompt Generator* may use strategies such as summarizing past decisions or truncating older context. It is then passed to the *LLM Engine*.

ii) LLM Engine: Understands the prompt P passed into it and generates adaptation decisions. The conversation history, H gives information pertaining to how previous AD_t and C_t variated through timesteps. The objective, O, helps in choosing adaptation decisions that build towards it. Using this data, the LLM will be able to generate an adaptation decision AD (refer Figure 2) which is then pushed into H and passed to *Response Parser*. Each AD may in turn consist of multiple actions.

iii) Response Parser Parses the AD given and formats it. This formatting is necessary since the adaptation needs to comply with the communication format of the managed system. It also extracts arguments that the managed system may need from AD.

For example, the P that was provided at a timestep was of the form $P = \{C, H, P_{SWIM}\}$. The LLM generated response *1 0.8*, which was further parsed by the *Response Parser* to produce 1 as the adaptation action and 0.8 as the argument (dimmer value) for the action.

Execute is responsible for executing the adaptations on the running system. It further consists of two components. *i) Verifier :* which verifies the given AD by leveraging formal

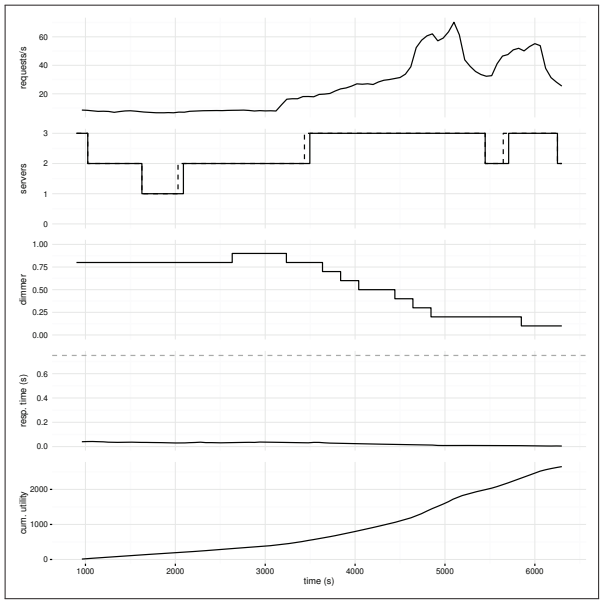

Fig. 4. Results from running the GPT-4 based adaptation manager on the world cup trace in SWIM. We observe that the average response time remains stable throughout the simulation.

verification techniques to check quantitative properties on the system to determine if the proposed AD aligns with our objective. If so the *Executor* will execute the given AD otherwise *Synthesizer* will generate an alternative adaptation decision. *ii) Executor :* Once the AD is verified, the *Executor* is responsible for the execution of the AD within the managed system by modifying the system configuration or resources as specified by the AD.

For instance, if the *LLM Engine* decides to add a server to handle higher traffic, the Verifier first confirms this action A aligns with the objectives O. Upon verification, the Executor would allocate an additional server to accommodate the increased load. We have outlined the *Executor* phase but haven't used the Verifier component in our approach for SWIM yet.

IV. Preliminary Results and Discussion

The SWIM simulation was run in a dockerized environment, on a machine running a distribution of Ubuntu 22.04 LTS: AMD Ryzen 7 PRO 5850U at 4.5GHz and 16GB of RAM. The SWIM simulation was run with the world cup trace, starting with three servers and a dimmer value of 0.9, for the full length of the simulation (105 minutes). The adaptation manager was a python file, consisting of the interfaces to communicate with SWIM and the GPT-4 API (LLM). It interacts with SWIM through a TCP interface. Python version used for running the adaptation manager was 3.10.12. The adaptation manager runs the MSE loop every 200 seconds in our case. In this case, the objective given to the LLM is to keep the response time as low as possible.

Our experiment [1] involving SWIM demonstrates the poten-

[1] https://github.com/Raghav010/llm_selfAdapt

tial of LLMs to enhance self-adaptation in software systems. As illustrated in Figure 4, employing LLMs ensured stable response times below 0.1 seconds, even with varying workloads, achieving a utility score that exceeds a value of 2500. When compared to the traditional reactive adaptation managers in SWIM [7], which although reaching a utility score exceeding a value of 3500 struggled to maintain stable response times with spikes of more than 6 seconds. The LLM-based adaptation manager attained a utility score that is roughly 71% of the reactive adaptation manager. An increase in requests triggered the adaptation manager to efficiently scale server resources and decrease the dimmer value, thus minimizing the response time. These results effectively address our primary objective of reducing response time, highlighting the potential of LLMs to optimize resource allocation and system performance in real-time, as well as manage system behaviour dynamically.

V. Vision for the Future and Further Research

Exploring LLMs in self-adaptation marks the beginning of a promising research area with vast potential. Future efforts should focus on enhancing LLMs' understanding of complex system dynamics. While LLMs effectively grasp the qualitative effects of adaptations on system performance, their grasp of complex equations detailing adaptation impacts is limited. Integrating knowledge infusion techniques could provide LLMs with deeper insights into system operations and effects. Expanding to multiple LLMs or multi-agent LLM architectures [5] offers scalability and improved decision-making for comprehensive software applications, potentially reducing errors and improving adaptation quality. This setup could distribute system components across LLM agents or organize them hierarchically for deeper analysis.

Addressing long context challenges in production environments might benefit from advanced technologies like MemGPT [1] or StreamingLLM [12], focusing on optimizing scenario storage for better adaptation strategies. Fine-tuning LLMs, particularly for domain-specific applications, and employing Reinforcement Learning using System Feedback (RLSF) to adjust model parameters based on the system objectives, represent critical areas for future research. This approach could significantly enhance LLM performance and reliability.

Another challenge is the ever-discussed issue of hallucination which may result in incorrect decisions produced by the LLM. This is where we believe that LLMs combined with formal verification techniques similar to the approach presented by Camara et al. [10] can improve the guarantees on the decisions generated.

Despite the list of aforementioned challenges, this paper underscores the transformative impact of integrating LLMs into self-adaptive systems, paving the way for more resilient, intelligent, and efficient software solutions.

[1] https://github.com/cpacker/MemGPT

VI. Conclusion

In this work, we have presented a novel self-adaptation approach that makes use of an MSE-K loop for dynamic architectural adaptation by leveraging the capabilities of LLMs. Our initial findings through evaluations on an exemplar system highlight LLMs' transformative impact on software engineering, enabling complex, human-like decision-making and strategies for software to autonomously adapt their architecture with reduced human intervention. This research sets a foundation for further exploration into LLMs' capabilities, striving for software that is increasingly adaptive, resilient, and efficient in an ever-evolving technological landscape.

References

[1] J. Kephart and D. Chess, "The vision of autonomic computing," Computer, vol. 36, no. 1, pp. 41–50, Jan 2003.

[2] D. Weyns, An introduction to self-adaptive systems: A contemporary software engineering perspective. John Wiley & Sons, 2020.

[3] O. Gheibi, D. Weyns, and F. Quin, "Applying machine learning in self-adaptive systems: A systematic literature review," ACM Trans. Auton. Adapt. Syst., vol. 15, no. 3, aug 2021. [Online]. Available: https://doi.org/10.1145/3469440

[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," arXiv preprint arXiv:2308.10620, 2023.

[5] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation," Microsoft, Tech. Rep., August 2023.

[6] T. Bureš, "Self-adaptation 2.0," in 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), May 2021, pp. 262–263.

[7] G. A. Moreno, B. Schmerl, and D. Garlan, "Swim: an exemplar for evaluation and comparison of self-adaptation approaches for web applications," in Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems, ser. SEAMS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 137–143. [Online]. Available: https://doi.org/10.1145/3194133.3194163

[8] D. Kim and S. Park, "Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software," in 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, May 2009, pp. 76–85.

[9] H. N. Ho and E. Lee, "Model-based reinforcement learning approach for planning in self-adaptive software system," in Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication, ser. IMCOM '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2701126.2701191

[10] J. Cámara, H. Muccini, and K. Vaidhyanathan, "Quantitative verification-aided machine learning: A tandem approach for architecting self-adaptive iot systems," in 2020 IEEE International Conference on Software Architecture (ICSA), March 2020, pp. 11–22.

[11] O. Gheibi and D. Weyns, "Lifelong self-adaptation: self-adaptation meets lifelong machine learning," in Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems, ser. SEAMS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–12. [Online]. Available: https://doi.org/10.1145/3524844.3528052

[12] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," 2023.