Towards AI for Software Systems

Nafise Eskandani
ABB AG Corporate Research Center
Germany

Guido Salvaneschi
University of St. Gallen
Switzerland

ABSTRACT

Generative Artificial Intelligence (GenAI) is being adopted for a number of Software Engineering activities, mostly centering around coding, such as code generation, code comprehension, code reviews, test generation, and bug fixing. Other phases in the Software Engineering process have been less explored. In this paper, we argue that more investigation is needed on the support that GenAI can provide to the design, and operation of *software systems*, i.e., a number of crucial activities, beyond coding, that are necessary to successfully deliver and maintain software services. These include reasoning about architectural choices and dealing with third-party platforms.

We discuss crucial aspects of AI for software systems. taking as a use case Function as a Service (FaaS). We present several challenges, including cold start delays, stateless functions, debugging complexities, and vendor lock-in and explore the potential of GenAI tools to mitigate FaaS challenges. Finally, we outline future research into the application of GenAI tools for the development and deployment of software systems.

CCS CONCEPTS

• **Software and its engineering** → **Software design engineering**.

KEYWORDS

Generative AI, Serverless Computing, FaaS

ACM Reference Format:
Nafise Eskandani and Guido Salvaneschi. 2024. Towards AI for Software Systems. In *Proceedings of 1st ACM International Conference on AI-powered Software Proceedings (AIware '24).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3664646.3664767

1 INTRODUCTION

Generative Artificial Intelligence (GenAI) is gaining significant attention in software development. GenAI tools such as OpenAI's ChatGPT [1] and GitHub Copilot [2] are at the forefront of this revolution, offering developers suggestions and assisting them in generating code snippets. These tools have shown impressive speed gains for many common developer tasks. The adoption of GenAI is expected to grow, with a Gartner report predicting that by 2026, more than 80% of enterprises will have used GenAI APIs or deployed GenAI-Enabled Applications [3]. While the use of AI for software *code* has been explored in depth (e.g., code generation, comprehension, and fixing), comparatively less effort has been devoted to the benefits of AI for *software systems* as a whole, i.e. including architectural design, platforms and operation. We provide a first exploration of this potential in the concrete context of Function as a Service (FaaS) which has rapidly gained popularity over the last years. In the FaaS model, programmers upload the code of one or more functions to a cloud platform which respond to specific trigger events, such as REST requests, file uploads, or database events. Once the code is uploaded, the cloud provider takes on the responsibility of deploying and managing the underlying resources. This approach eliminates the need for over-provisioning to handle peak resource demands, as resources automatically scale based on actual usage. Consequently, developers are billed only for the resources actively consumed by their applications.

Despite its several advantages, FaaS faces challenges that can impact its development and usage. These challenges include cold start delays, the stateless nature of functions, complexities in debugging and monitoring, and perhaps most notably, vendor lock-in [13, 20, 34, 42]. A variety of studies have tackled these issues using different methodologies and frameworks, with the goal of improving FaaS performance and easing the development process for FaaS developers [7, 25]. These solutions involve a wide range of approaches, from pre-warming and scheduling techniques [26, 43, 44] to the implementation of stateful functions [2, 3, 45], and the provision of troubleshooting and monitoring services [5, 19, 31]. Additionally, some have proposed cloud-agnostic models to mitigate the risk of vendor lock-in [32, 39, 56]. Nevertheless, challenges persist in both deployment and utilization of FaaS solutions [6, 23, 52].

In this paper, we explore the potential of GenAI for software systems, showing how it can address these challenges for FaaS. We propose directions for future investigations into the application of GenAI for the development and deployment of serverless applications. Our research indicates the versatility and potential of the GenAI tools in various phases of serverless application development. The capabilities of GenAI tools spans a wide range, from assisting in the identification of the optimal architectural design tailored to specific application requirements, to speeding up the implementation of core functionalities, to providing detailed instructions for deployments that are aligned with the specifications of FaaS platforms. We believe that developers, by leveraging GenAI, can navigate the complexities of serverless application development.

[1] https://openai.com/chatgpt
[2] https://docs.github.com/en/copilot

AIware '24, July 15–16, 2024, Porto de Galinhas, Brazil
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0685-1/24/07…$15.00
https://doi.org/10.1145/3664646.3664767

[3] https://shorturl.at/qBIQU

2 BACKGROUND AND RELATED WORK

2.1 GenAI for Software Development

GenAI has seen significant evolution with the introduction of the Prompt paradigm and Reinforcement Learning from Human Feedback (RLHF), in combination with pre-trained Large Language Models (LLMs) [16]. These techniques have enabled GenAI to transition from task-specific to gradually adopting a more general pattern [15]. RLHF, in particular, has enabled GenAI to adapt to the specific needs and preferences of individuals.

One of the most notable examples of GenAI tools – OpenAI's ChatGPT – is capable of generating code, crafting narratives, facilitating machine translation and conducting semantic analysis [55]. Derived from the generative pre-training transformer (GPT), ChatGPT belongs to the family of transformer-based LLMs [14, 49]. Users engage with GenAI tool like ChatGPT via "prompts" [27] that (1) provide the LLM with a context for its operation, and (2) guide its generation of responses. The construction of these prompts is a critical aspect of GenAI, as users can tailor them to achieve specific outcomes.

In software development, GenAI has become increasingly popular, providing assistance to numerous managerial and technical project activities. Tools like GitHub Copilot, for instance, leverage the power of GenAI to automate and improve various aspects of software development, from code generation to project management [9]. These tools can learn from past data and developer feedback to make recommendations, further improving the efficiency of software development processes [33].

An increasing number of research studies have focused on leveraging GenAI for software development tasks, including requirements engineering, design, and testing [9, 10, 36, 37, 41]. Researchers have explored the use of GenAI tools for retrieving requirements information [40, 60] and generating architectural designs [1, 17]. Additionally, GenAI has been investigated for code generation, program repair, and code summarization [38, 46, 47, 57], as well as its potential to improve software testing process [35]. The results of these studies highlight the effectiveness of GenAI tools in software development. Yet, despite such investigations, the use of GenAI tools across different phases of software development remains a relatively unexplored area [38].

2.2 FaaS in a Nutshell

The key feature of FaaS model is its ability to abstract away the complexities of infrastructure management, allowing developers to focus solely on writing and deploying code. This approach not only facilitates autoscaling, but also reduces operational costs. Applications range from web and mobile backends to real-time data processing, IoT, and even machine learning inference [11]. Its elastic scalability makes it ideal for handling unpredictable workloads, while its pay-as-you-go pricing model aligns well with cost-effective resource allocation strategies. Developing a serverless application involves several phases, that remain consistent across different FaaS platforms, with differences in implementation. In the following, we discuss some of the key phases.

Application Design: This phase involves identifying the requirements and goals of the application. This includes understanding the domain, defining the functionality, and outlining the user experience. The architectural pattern of the application can then be designed to integrate diverse services and functions into a cohesive unit that fulfills these requirements. This architecture further outlines the interconnections among these services and functions to ensure the correct data flow within the application.

The stateless nature of serverless architecture poses challenges in maintaining application state across invocations [4, 12]. This often results in a dependency on external services for managing state, introducing delays and complexity [48]. Therefore, selecting an appropriate architectural pattern is essential to align with the quality attributes specified by the application requirements.

Function Development: Following the architectural design, the focus shifts to implementing the core functionality of the application. This is where the actual programming takes place. Each function is developed to execute a distinct task, adhering to a stateless and event-driven paradigm. The event-driven paradigm of FaaS combined with decentralized structure of serverless applications compound these challenges [24]. Particularly, managing asynchronous events, preserving event sequence, and tracing data flow across multiple functions and services become significantly complicated, consequently increasing debugging and performance optimization efforts [13, 42].

Environment and Resource Configuration: The next step is defining the infrastructure requirements, such as databases, storage, and event source mappings. These settings can be defined either through the cloud console provided by the FaaS platform or via a configuration file using a data serialization language such as JSON or YAML – hence supporting version control and repeatability.

Managing dependencies for each function can be complex [53]. Functions may require different versions of the same library, leading to conflicts. Additionally, each FaaS platform offers unique features and services, such as the maximum execution time for a function, the number of concurrent executions, and the size of the deployment package. These limits can constrain the design of the application [59]. Migration to another provider becomes cumbersome, involving code modifications and potential functionality loss due to platform-specific features.

Application Deployment: This phase involves packaging the functions and their dependencies, and uploading them to the FaaS platform, which takes care of server management, scaling, and capacity planning, allowing the application to be event-responsive and highly available. A well-known challenge of FaaS is cold start: a function invocation incurs additional latency when the function is not already loaded in memory [18]. The impact on the performance can be significant, particularly for latency-sensitive applications [48]. Also, pricing structures and SLAs vary among platforms, complicating cost estimation and performance optimization. The cognitive overhead of navigating the documentation of the platforms further exacerbates these challenges [13].

Application Monitoring and Debugging: The final phase is to set up monitoring and debugging mechanisms. Monitoring involves tracking the performance of the application, including function invocations, execution times, and error rates. This can be achieved using the tools provided by the chosen FaaS platform. Debugging and monitoring of serverless applications involves examining the performance of the application, as well as log outputs for each function invocation to pinpoint the root cause of any issues. Given the distributed nature of serverless applications, traditional debugging methods may not be effective, necessitating the use of specialized serverless debugging tools and practices.

3 GENAI FOR SERVERLESS APPLICATIONS

3.1 Preliminary Experiment

Conducting thorough experiments is essential for evaluating the effectiveness of GenAI tools in software system development. As an initial step toward this goal, we outlined an experiment to assess the impact of these tools on designing and deploying serverless applications. The experiment focuses on three scenarios, each designed to address a common use cases of serverless applications.

- The first scenario is a **web service** handling requests from clients and processing each in a serverless function.
- The second scenario is an **online retail platform** with separate endpoints for user authentication, product catalog management, order processing, and customer support.
- The third scenario focuses on an **image processing application**, dynamically resizing uploaded images and storing processed data in a database.

We created prompts for GPT4 in four formats: (1) plain text, (2) structured text, (2) JSON configuration, and (4) code templates, and requested deployment on Azure Functions.

Among other findings, we observed that the model selects an appropriate architecture based on application requirements. However, the choice of prompt format significantly impacts result correctness and quality. For instance, plain text prompts generate accurate design and code for basic web services, but struggle with more complex scenarios. The granularity of the prompt also impacts the outcome. For instance, a gradual approach tends to be more effective than providing all requirements at once. Additionally, the model lacks emphasis on security best practices; it hard-codes service keys in the suggested code. Such initial findings from a limited setup necessitate further validation and exploration. In the following, we propose potential research directions for these investigations.

3.2 Promising Solutions

GenAI models can leverage extensive domain knowledge enabling them to execute tasks across a diverse array of fields [58]. In software system development, this domain knowledge can help improving the development and deployment process by identifying the appropriate architectural pattern, determining triggers and bindings for serverless functions, as well as configuring the serverless environment. Based on the preliminary results from Section 3.1 we discuss how AI can address some of the challenges associated with FaaS in development phases mentioned in Section 2.2. Table 1 shows concrete examples of such mitigations.

Selection of Architectural Patterns: Popular GenAI models have significant knowledge of architectural patterns, including the strengths, weaknesses, and appropriate use cases for each pattern. Such understanding enables GenAI tools to select the most suitable pattern based on the specific requirements of the application. For example, if an application requires heavy data processing, GenAI may choose the Fan-out and Fan-in pattern which aims to efficiently distribute and aggregate tasks. Furthermore, GenAI tools have the ability to translate these patterns into actual serverless application architectures. This involves defining the functions, events, and resources of the application, and configuring their interactions according to the chosen pattern.

Adapted definitions for platforms: LLMs used in popular GenAI tools are trained on data that is publicly available from all FaaS platforms and can generate platform-specific instructions and code. This capability of GenAI is particularly beneficial in mitigating the cognitive overhead associated with the FaaS model. Crucially, by generating instructions and code tailored to each platform, GenAI can reduce the complexities and dependencies associated with requirements of a specific vendor. Also, GenAI tools can enable developers to rapidly transform the code and deployment instructions for a FaaS platform to another application deployment on a different platform possibly with different requirements. This feature is particularly useful in today's diverse cloud environment, where applications often need deployment across multiple platforms.

Smart Pre-warming: To mitigate the cold start delays of the FaaS model, GenAI tools can predict the demand pattern of serverless functions and pre-warm the function accordingly. Leveraging machine learning algorithms, GenAI tools analyze historical usage data and predict future demand patterns. This ensures that serverless functions are pre-warmed and ready to execute as soon as they are invoked. Furthermore, GenAI models are capable of continuous learning and adaptation to changes in the usage pattern, ensuring optimal function performance over time.

Enhanced Observability: GenAI models can predict bugs of a serverless application and generate corrective measures. For instance, they can create synthetic data that mirrors the behavior of distributed functions, enabling developers to simulate and analyze various scenarios without a live environment. This proactive approach allows for the early detection of anomalies, reducing the time spent on debugging. Furthermore, GenAI can enhance monitoring by generating visualizations and reports based on collected data, providing insights into the application's performance. These capabilities can significantly improve the process of serverless application development and deployment. By automating the selection and implementation of serverless patterns, GenAI can help developers create efficient and cost-effective serverless applications. Yet, the effectiveness of GenAI 's designs ultimately depends on its understanding of the application requirements and its ability to accurately map these requirements to the appropriate pattern.

4 OPEN RESEARCH DIRECTIONS

Prompt engineering is critical in GenAI models. The prompt – the model's input – can significantly influence the output. Prompt engineering involves modifying both the structure and content of the prompt [8]. Structural modifications include changing the prompt length or the arrangement of instances in it. Content modifications is about phrasing, the choice of illustrations, or the directives given. Previous studies [28, 29, 50] indicate that the model's behavior is significantly influenced by the phrasing and sequence of examples.

Prompt formats for GenAI models, include plain text, code templates, and images. This flexibility allows a diverse range of applications and use-cases. As seen in our experiment (Section 3.1) serverless applications can be specified in various ways: requirements can be articulated in text, with code templates provided

Table 1: The potential of GenAI to address FaaS challenges in different development phases

Development Phase	Challenge	GenAI-based Mitigation
Application Design	Stateless functions	Optimal architectural pattern based on application requirements
Function Development	Event-driven paradigm	Code generation according to the chosen architectural pattern
Environment Configuration	Vendor lock-in	Configuration adaptation across divers FaaS platforms
Application Deployment	Cold start	Smart pre-warming by predicting demand patterns
Monitoring and Debugging	Distributed functions	Predicting issues, generating corrective measures, visualizations, and reports

for function implementations. Additionally, configuration for the deployment environment can be supplied in JSON or YAML files.

Prompt Format vs Architectural Complexity: The impact of the prompt format on the quality of the generated application is an interesting ground for research. A crucial research question is: *"How does the prompt format interact with the complexity of the architectural design for generating serverless applications?"* For instance, it may be more appropriate to provide text for developing a simple web service with a single function. Yet, a complex data processing application with several functions and external services may benefit more from a combination of text and an image that is conveying the architecture.

Architectural Details vs Result Accuracy The level of detail in a prompt can also influence the output. For example, prompts can be used in two different ways for serverless applications. (1) They can specify the requirements, asking the LLM to generate code for the serverless function and instructions for deploying the application. This approach enables a high degree of customization, tailoring the generated code and instructions to the specific requirements in the prompt. (2) Prompts can refer to a particular serverless architectural pattern, requesting the LLM to generate code and instructions for it. This is a more general method that relies on predefined patterns rather than specific requirements and can be highly efficient for common use cases, allowing rapid generation of functional code.

Investigating the relationship between the level of detail in a prompt and the result can yield valuable insights. For instance, efficient prompt patterns [54] for common use cases of the FaaS model can be reused to generate applications for similar scenarios, saving time and resources. The research question is: *"Does referring to services from a specific FaaS platform in the prompt influence the accuracy of the generated application when the request is to deploy the application to a different platform?"* This question arises from the observation that the context in the prompt may bias the generation towards the mentioned platform, potentially affecting the correctness of a deployment on a different platform. The investigation can involve a comparative study of application generation with and without mentioning specific FaaS services, followed by an evaluation of the success of deployment.

Prompt Decomposition vs Application Decomposition: GenAI models have demonstrated the ability to perform more complex reasoning tasks when provided with a sequence of reasoning steps. This approach, known as Chains-of-Thought prompting [51], illustrates the effect of breaking down a complex task into simpler steps. This capability of GenAI models can be particularly useful in the development of serverless applications, where complex tasks need to be decomposed into serverless functions. A promising line of research can investigate the optimal number, granularity, and sequence of prompts for designing, developing, and deploying serverless applications. One potential research question can be: *"How does prompt granularity affect the generated output for with multi-function serverless applications?"* This line of research can also explore how different developers respond to prompts with varying granularities, potentially leading to personalized prompt recommendations for individual developer's needs and preferences.

Innovation vs Reliability: A key aspect that affects the quality of GenAI models is "hallucination" [22]. This term is used to describe situations where the AI produces content that is imaginative or fictional in nature. While this can sometimes lead to creative and innovative outputs, it can also result in outputs that are not grounded in reality or factual information. This is a common concern in the field of AI, as it can impact the reliability and trustworthiness of the generated content. In serverless application development, hallucination can have several implications. For example, the model may suggest managed services that have been deprecated or were never available by a serverless platform. Another example can be adding non-existent or unnecessary requirements for the application. The model may suggest implementing certain features or functionalities that are not relevant to the application's purpose or the developer's needs. Despite the extensive research [21, 30, 61] conducted on harmonizing creativity with factual accuracy, the challenge persists and requires further investigations. A potential research question can be: *"What are the impacts of hallucination on serverless application development?"*

Result Correctness vs Software Quality Attributes: Another line of research can explore software quality attributes of the generated serverless applications. Software quality attributes are the features and characteristics that affect a software's ability to meet its objectives. These attributes include performance efficiency, reliability, usability, maintainability, portability, and security. Each of these attributes contributes to the overall quality of the software, and their importance can vary depending on the specific requirements of the software. For example, a research can explore: *"How does the serverless architectural pattern influence the performance efficiency of the generated applications?"*.

5 SUMMARY

AI support for software development has been largely studied for activities related to coding the application logic. Less attention has been devoted to the issues that emerge when engineering *software systems*, including architecture trade offs, platforms, and operations. This paper provides a preliminary discussion and a roadmap for adopting AI in engineering software systems, taking FaaS and serverless application development as a use case and discussing the challenge that emerge in this domain, such as FaaS architectural design, cold start delays, stateless functions, debugging complexities, and vendor lock-in.

REFERENCES

[1] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards human-bot collaborative software architecting with chatgpt. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 279–285.

[2] Adil Akhter, Marios Fragkoulis, and Asterios Katsifodimos. 2019. Stateful functions as a service in action. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1890–1893.

[3] Daniel Barcelona-Pons, Marc Sánchez-Artigas, Gerard París, Pierre Sutra, and Pedro García-López. 2019. On the faas track: Building stateful distributed applications with serverless architectures. In *Proceedings of the 20th international middleware conference*. 41–54.

[4] Daniel Barcelona-Pons, Pierre Sutra, Marc Sánchez-Artigas, Gerard París, and Pedro García-López. 2022. Stateful serverless computing with crucial. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–38.

[5] Maria C Borges, Sebastian Werner, and Ahmet Kilic. 2021. Faaster troubleshooting-evaluating distributed tracing approaches for serverless applications. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 83–90.

[6] Paul Castro, Vatche Isahagian, Vinod Muthusamy, and Aleksander Slominski. 2023. Hybrid serverless computing: Opportunities and challenges. *Serverless Computing: Principles and Paradigms* (2023), 43–77.

[7] Mohak Chadha, Anshul Jindal, and Michael Gerndt. 2021. Architecture-specific performance optimization of compute-intensive faas functions. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 478–483.

[8] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2023. Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735* (2023).

[9] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734.

[10] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv:2304.07590* (2023).

[11] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. A review of serverless use cases and their characteristics. *arXiv preprint arXiv:2008.11110* (2020).

[12] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2021. The state of serverless applications: Collection, characterization, and community consensus. *IEEE Transactions on Software Engineering* 48, 10 (2021), 4152–4166.

[13] Nafise Eskandani and Guido Salvaneschi. 2023. The uphill journey of FaaS in the open-source community. *Journal of Systems and Software* 198 (2023), 111589.

[14] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. *arXiv preprint arXiv:2310.03533* (2023).

[15] Patrick Fernandes, Aman Madaan, Emmy Liu, António Farinhas, Pedro Henrique Martins, Amanda Bertsch, José GC de Souza, Shuyan Zhou, Tongshuang Wu, Graham Neubig, et al. 2023. Bridging the gap: A survey on integrating (human) feedback for natural language generation. *Transactions of the Association for Computational Linguistics* 11 (2023), 1643–1668.

[16] Giorgio Franceschelli and Mirco Musolesi. 2024. Reinforcement Learning for Generative AI: State of the Art, Opportunities and Open Research Challenges. *Journal of Artificial Intelligence Research* 79 (2024), 417–446.

[17] Theodoros Galanos, Antonios Liapis, and Georgios N Yannakakis. 2023. Architext: Language-Driven Generative Architecture Design. *arXiv preprint arXiv:2303.07519* (2023).

[18] Muhammed Golec, Guneet Kaur Walia, Mohit Kumar, Felix Cuadrado, Sukhpal Singh Gill, and Steve Uhlig. 2023. Cold start latency in serverless computing: A systematic review, taxonomy, and future directions. *arXiv preprint arXiv:2310.08437* (2023).

[19] Beatriz Guerreiro, Filipe Freitas, and José Simão. 2023. Monitoring in Function-as-a-Service Platforms. In *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 1–4.

[20] Hassan B Hassan, Saman A Barakat, and Qusay I Sarhan. 2021. Survey on serverless computing. *Journal of Cloud Computing* 10 (2021), 1–29.

[21] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232* (2023).

[22] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[23] Vincent Lannurien, Laurent D'orazio, Olivier Barais, and Jalil Boukhobza. 2023. Serverless Cloud Computing: State of the Art and Challenges. *Serverless Computing: Principles and Paradigms* (2023), 275–316.

[24] Yongkang Li, Yanying Lin, Yang Wang, Kejiang Ye, and Chengzhong Xu. 2022. Serverless computing: state-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing* 16, 2 (2022), 1522–1539.

[25] Changyuan Lin, Nima Mahmoudi, Caixiang Fan, and Hamzeh Khazaei. 2022. Fine-grained performance and cost modeling and optimization for faas applications. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (2022), 180–194.

[26] Ping-Min Lin and Alex Glikson. 2019. Mitigating cold starts in serverless platforms: A pool-based approach. *arXiv preprint arXiv:1903.12221* (2019).

[27] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[28] Yi Liu, Gelei Deng, Zhengzi Xu, Yuekang Li, Yaowen Zheng, Ying Zhang, Lida Zhao, Tianwei Zhang, and Yang Liu. 2023. Jailbreaking chatgpt via prompt engineering: An empirical study. *arXiv preprint arXiv:2305.13860* (2023).

[29] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786* (2021).

[30] Potsawee Manakul, Adian Liusie, and Mark JF Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896* (2023).

[31] Johannes Manner, Stefan Kolb, and Guido Wirtz. 2019. Troubleshooting serverless functions: a combined monitoring and debugging approach. *SICS Software-Intensive Cyber-Physical Systems* 34 (2019), 99–104.

[32] Oliviu Matei, Katarzyna Materka, Paweł Skyscraper, and Rudolf Erdei. 2021. Functionizer-a cloud agnostic platform for serverless computing. In *International Conference on Advanced Information Networking and Applications*. Springer, 541–550.

[33] Anh Nguyen-Duc, Beatriz Cabrero-Daniel, Adam Przybylek, Chetan Arora, Dron Khanna, Tomas Herda, Usman Rafiq, Jorge Melegati, Eduardo Guerra, Kai-Kristian Kemell, et al. 2023. Generative Artificial Intelligence for Software Engineering–A Research Agenda. *arXiv preprint arXiv:2310.18648* (2023).

[34] Bjorn Pijnacker and Jesper van der Zwaag. [n. d.]. Opportunities and Challenges in the Adoption of Function-as-a-Service Serverless Computing. *20th SC@ RUG 2022-2023* ([n. d.]), 65.

[35] Ameya Shastri Pothukuchi, Lakshmi Vasuda Kota, and Vinay Mallikarjunaradhya. 2023. Impact of generative ai on the software development lifecycle (sdlc). *International Journal of Creative Research Thoughts* 11, 8 (2023).

[36] Rohith Pudari and Neil A Ernst. 2023. From copilot to pilot: Towards AI supported software development. *arXiv preprint arXiv:2303.04142* (2023).

[37] Asha Rajbhoj, Akanksha Somase, Piyush Kulkarni, and Vinay Kulkarni. 2024. Accelerating Software Development Using Generative AI: ChatGPT Case Study. In *Proceedings of the 17th Innovations in Software Engineering Conference*. 1–11.

[38] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498* (2022).

[39] Pedro Rodrigues, Filipe Freitas, and José Simão. 2022. Quickfaas: Providing portability and interoperability between faas platforms. *Future Internet* 14, 12 (2022), 360.

[40] Kun Ruan, Xiaohong Chen, and Zhi Jin. 2023. Requirements Modeling Aided by ChatGPT: An Experience in Embedded Systems. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. IEEE, 170–177.

[41] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekki, and David Doermann. 2024. Future of software development with generative AI. *Automated Software Engineering* 31, 1 (2024), 26.

[42] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2022. Serverless computing: a survey of opportunities, challenges, and applications. *Comput. Surveys* 54, 11s (2022), 1–32.

[43] Jiacheng Shen, Tianyi Yang, Yuxin Su, Yangfan Zhou, and Michael R Lyu. 2021. Defuse: A dependency-guided function scheduler to mitigate cold starts on faas platforms. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 194–204.

[44] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. 2020. Prebaking functions to warm the serverless cold start. In *Proceedings of the 21st International Middleware Conference*. 1–13.

[45] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Jose M Faleiro, Joseph E Gonzalez, Joseph M Hellerstein, and Alexey Tumanov. 2020. Cloudburst: Stateful functions-as-a-service. *arXiv preprint arXiv:2001.04592* (2020).

[46] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F Bissyandé. 2023. Is ChatGPT the ultimate programming assistant–how far is it? *arXiv preprint arXiv:2304.11938* (2023).

[47] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems*

extended abstracts. 1–7.

[48] Erwin van Eyk and Alexandru Iosup. 2018. Addressing performance challenges in serverless computing. *Proc. ICT. Open* (2018), 6–7.

[49] Yuntao Wang, Yanghe Pan, Miao Yan, Zhou Su, and Tom H Luan. 2023. A survey on ChatGPT: AI-generated contents, challenges, and solutions. *IEEE Open Journal of the Computer Society* (2023).

[50] Albert Webson and Ellie Pavlick. 2021. Do prompt-based models really understand the meaning of their prompts? *arXiv preprint arXiv:2109.01247* (2021).

[51] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[52] Jinfeng Wen, Zhenpeng Chen, Xin Jin, and Xuanzhe Liu. 2023. Rise of the planet of serverless computing: A systematic review. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–61.

[53] Jinfeng Wen, Zhenpeng Chen, Yi Liu, Yiling Lou, Yun Ma, Gang Huang, Xin Jin, and Xuanzhe Liu. 2021. An empirical study on challenges of application development in serverless computing. In *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 416–428.

[54] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *arXiv preprint arXiv:2303.07839* (2023).

[55] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. 2023. A brief overview of ChatGPT: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica* 10, 5 (2023), 1122–1136.

[56] Michael Wurster, Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Vladimir Yussupov. 2018. Modeling and automated deployment of serverless applications using tosca. In *2018 IEEE 11th conference on service-oriented computing and applications (SOCA)*. IEEE, 73–80.

[57] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th international conference on predictive models and data analytics in software engineering*. 62–71.

[58] Ilker Yildirim and LA Paul. 2023. From task structures to world models: what do LLMs know? *Trends in Cognitive Sciences* (2023).

[59] Vladimir Yussupov, Uwe Breitenbücher, Frank Leymann, and Christian Müller. 2019. Facing the unplanned migration of serverless applications: A study on portability problems, solutions, and dead ends. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 273–283.

[60] Jianzhang Zhang, Yiyang Chen, Nan Niu, and Chuang Liu. 2023. A preliminary evaluation of chatgpt in requirements information retrieval. *arXiv preprint arXiv:2304.12562* (2023).

[61] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the AI ocean: a survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).