Generative AI And Software Variability — A Research Vision

Sandra Greiner
University of Bern, Switzerland
sandra.greiner@unibe.ch

Klaus Schmid
University of Hildesheim, Germany
schmid@sse.uni-hildesheim.de

Thorsten Berger
Ruhr University Bochum
thorsten.berger@rub.de

Sebastian Krieter
University of Ulm
sebastian.krieter@uni-ulm.de

Kristof Meixner
TU Wien, CDL-SQI, Austria
kristof.meixner@tuwien.ac.at

ABSTRACT

Generative Artificial Intelligence (GAI) promises groundbreaking automation technology - a potential which may raise the management of variability-intensive software systems to a new level of automation. Several activities in maintaining variability-intensive software systems, such as extracting feature traces to updating features consistently, are repetitive and performed mainly manually or semi-automatically. Exploiting the potentials of GAI in maintaining variability-intensive software systems opens a fundamentally new research perspective, where GAI shall solve repetitive and hard-to-automate tasks. In this vision paper, we propose and discuss increasing levels of maintaining variability-intensive software systems automatically enabled through the support of GAI. We sketch actions necessary to reach the next levels of automation while discussing the current state-of-the-art.

KEYWORDS

Variability-intensive software systems, product lines, generative AI

ACM Reference Format:
Sandra Greiner, Klaus Schmid, Thorsten Berger, Sebastian Krieter, and Kristof Meixner. 2024. Generative AI And Software Variability — A Research Vision . In *18th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2024), February 7–9, 2024, Bern, Switzerland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3634713.3634722

1 INTRODUCTION

Generative Artificial Intelligence (**GAI**) [14, 23] and particularly the free access to pretrained transformer models [13, 35] (types of *large language models* (**LLMs**)) [43] offers groundbreaking automation for many software engineering tasks. Due to the unsupervised learning on billions of (textual) data, these models can generalize their knowledge and perform tasks, such as code explanation [29] or feature detection [2] rather accurately without being explicitly trained for them [14]. This ability offers the benefit of employing LLMs for software engineering tasks where automation is hardly reached or requires costly language-specific solutions.

VaMoS 2024, February 7–9, 2024, Bern, Switzerland

ACM ISBN 979-8-4007-0877-0/24/02.
https://doi.org/10.1145/3634713.3634722

Maintaining variability-intensive software systems is laborious and error-prone once they comprise a certain number of configuration options [3] (a.k.a., features [24]). *Software product line engineering* (**SPLE**) aims to increase productivity and decrease cost when developing and maintaining variability-intensive software [4, 12]. By separating domain from application engineering, SPLE adopts specific processes as well as implementation, analysis, and testing strategies for maintaining variability-intensive software systems.

While SPLE advanced significantly over the past decades, various practical SPLE activities cannot be performed automatically today. For instance, integrating or discontinuing features [24] in legacy software or migrating related products into product lines [28]) still requires substantial manual efforts to check the heuristic results and to assess the impact of features and introduced interactions.

Recent research explores the potential of GAI to migrate products into a product line [2], implement 150 % representations [1], and generate meaningful feature configurations [18]. While single files can be handled easily, all explored LLMs are limited by the size of the input and output they can process. Furthermore, LLMs may produce relatively good results for source code tasks in programming languages they were pre-trained on, but they currently suffer from their non-deterministic nature and may not be applicable for programming languages used in legacy code or specific DSLs [17, 41]. The required significant level of human guidance [1, 2] contradicts the promised boost in automation and autonomy.

Nonetheless, exploiting the potential of GAI in managing variability-intensive software systems at all levels opens a groundbreaking research direction. Currently, GAI is able to perform any text-related analytical, generative, or even adaptive task, but without explicit awareness of variability. In this vision paper, we propose levels of GAI-support for managing variability-intensive software systems. We classify these levels based on dimensions required to support SPLE through GAI and outline research activities to increase the level of each dimension.

2 BACKGROUND

This section presents background knowledge on GAI and approaches where GAI has recently been applied to support S(PL)E tasks.

2.1 Generative AI

GAI is grounded in deep learning, an AI field that relies on neural networks. In GAI, layers of neurons are trained based on labeled and unlabeled data [6]. GAI learns patterns from the data and performs analytical, generative, and adaptive tasks, such as the analysis of language, code, or image and their description, creation,

and adaptation. LLMs, as one form of GAI, are trained on billions of textual data, such as documents or source code [38].

Transformer models [39], represent the state-of-the-art architecture used in natural language processing (**NLP**) tasks, such as text translation or summarization. They perform sequence-to-sequence tasks, where a set of input tokens is *encoded* into a vector space and *decoded* into tokens after being processed. While encoder-only and decoder-only models exist, they perform worse in generative and analytical tasks, respectively [30].

Current LLMs can successfully analyze or create and modify pieces of source code, but are limited by the amount of data they can process [15]. The so-called *context window* summarizes the size of the allowed in- and output tokens. Consequently, LLMs process methods and small software units easily but cannot handle large (legacy) projects or pieces of software. Furthermore, tuning parameters (e.g., temperature and repetition penalty) increase the likelihood of output variation [15]. Due to this non-deterministic behavior of GAI, there is no guarantee that the same input creates the same result again. Moreover, while GAI may generate text reasonably well, it fails, for instance, in computing complex arithmetic operations [19, 46]. To circumvent these limitations, it is possible to outsource computations to state-of-the-art programs (e.g., Python scripts) [19] and, thus, to use already established and tested tooling for tasks which the GAI is not optimized for.

Besides these technical challenges and potential workarounds, assessing the results produced by GAI challenges software engineers. As for NLP tasks, an exact match may not be the decisive metric, other metrics, such as BLEU [33], were introduced to respect the ambiguity of natural language. The same holds when it comes to tasks related to source code [36].

2.2 Employing GAI in SPLE

Employing GAI for software engineering tasks is an active field with rapid progress [10, 15, 22, 30, 38, 46]. However, GAI for supporting and maintaining variability-intensive software systems has received very little attention so far.

Two studies examine the capabilities of ChatGPT, as one representative of an GAI-assistant to migrate variants to a product line [2] and implement platform source code [1]. While the authors thoroughly document the used prompts and results, they cannot automatically verify the results, but must manually check them. In addition, upon repeating exactly the same prompts, the result is likely to be different due to the non-deterministic nature of GAI.

Another work explores GAI to create meaningful feature model configurations [18]. Interestingly, the authors embed the GAI in a pipeline where its results are post-processed to check syntactic and semantic correctness. In fact, since community benchmarks are missing, authors either manually check the results [1, 2] or rely on analysis tools specific for their use case [18].

3 GAI-EMPOWERED SPLE

We present dimensions of GAI support for SPLE in Section 3.1. We organize them into the GAI4SPL model in Section 3.2.

3.1 Dimensions of GAI-Empowered SPLE

This section introduces the following dimensions, which significantly influence the degree of automating GAI-supported tasks and characterize increasing capability levels of a GAI:

Scale: The amount of information the GAI can process in one step.
Task type: The type of SPLE tasks that the GAI supports.
Quality: The quality of the results; i.e., to what extent can one rely on their correctness.
Automation: The level of automation provided by a GAI and, consequently, the manual intervention required by developers.

Scale. Today, most applications of GAI for software engineering focus on relatively small artifacts (or excerpts) that need to be understood and/or generated. One reason is the context window through which a GAI understands its *inputs* and which is used as part of the *output* [15]. Currently, this window ranges from 8000 to up to 32k tokens for large models [9]. In practice, this poses a significant limit: while the context window can contain different artifacts, complete projects exceed the token limit easily and even just entering a few source files will easily exceed the limit. From a practical SPLE perspective, a few artifacts can be processed easily, but complete projects (particularly, entire product-line platforms) are out of scope, let alone multi-project scenarios, such as integrating multiple customer-specific products. Dealing with this limitation is highly actively researched in SE and AI [5, 45].

In the future, we envision that *small projects* (e.g., a variability model and multiple artifacts) may be handled. This will rise to *multiple small projects* (or single larger ones) and eventually reach a state where even multiple large-scale projects can be processed simultaneously – *a potentially unlimited size.* More critically, extractive SPLE [4] entails multiple independent artifacts of significant size to derive a common but variable implementation.

Task Type. GAI can support many different tasks ranging from *analytical*, such as analyzing or reviewing information to *generative*, such as creating new source code. *Adapting* artifacts may be even more challenging as it requires both analysis and generation capabilities. Within analytical, generative, and adaptive tasks several subcategories exist. For instance, combining multiple C-code variants into a single product line using pre-processor statements is an adaptive tasks. We envision future GAI to support a plethora of analytical, generative, and adaptive tasks.

A major challenge when using the GAI for such tasks is not whether it can produce a result. As it can readily hallucinate, GAI can be applied to *any* type of task. The more important question is whether it can address a task in a *productive* way. Thus, the task dimension is strongly related to the *quality* dimension.

It is important to note that the GAI can employ tools[1] [37], meaning it can ensure quality to some extent with tool support. Consequently, existing SPLE tooling may be useful to reduce complexity or to pre- and post-process the input or output of GAI. We expect many tasks to be solved by combining GAI and traditional tools, which emphasizes the importance of also improving traditional approaches.

[1] In the context of ChatGPT and other solutions, these are sometimes also called plugins.

Automation. This dimension describes how autonomously the GAI performs tasks. Of course, this is related to the type of tasks and quality, but it is also a fundamental design decision regarding solutions. Currently, AI needs to be explicitly invoked and guided (*low* automation) or makes autonomous proposals that need to be accepted by the user (e.g., GitHub Copilot [21]). As automated proposals do not require explicit invocations, the level of automation is higher than when requesting it explicitly. We refer to it as *medium*, an automation level which also subsumes other cases where strong user involvement is needed. Ultimately, we envision smart systems that perform major changes to the software with minimal user involvement. A high-level request like "integrate two projects and create a product line with a variability model" may still be required, but without fine-grained guidance. We regard such a system as *fully* automated.

Quality. This dimension refers to the accuracy and reliability of the results produced by the GAI. The dimension also relates with the dimensions *scale* and *task* because complex and larger tasks may increase the error probability. *Low* quality may be acceptable for task types, such as analysis, where identifying potential defects may be beneficial, if the result is significantly better than random. For generative tasks, *high* quality is important, i.e., very few defects may exist in the created output. While not necessarily flawless, high quality should be similar to a junior developer's result. Ultimately, we expect that GAI-empowered tools produce results comparable to very good, i.e., senior developers. We call this *expert* level.

3.2 GAI4SPL Model

While the dimensions, scale, task type, quality, and automation, provide a multi-faceted picture of GAI for SPLE, we organize the automation degrees into levels to guide GAI adoption in SPLE practice in the proposed GAI4SPL Model. A well-known example for such a strategy are the levels for self-driving cars [31]. This model entails increasing capability levels for self-driving cars, ranging from no automation to fully autonomous vehicles. The AI adoption evolution model [8] organizes levels similarly. We follow this lead and introduce the multi-leveled GAI4SPL model, organized in Table 1, as foundation for our research roadmap (cf. Section 4):

Level 0: No GAI. This level represents the baseline where no GAI is available. As tools, such as CoPilot, become widely adopted, the state of practice is shifting towards Level 1.

Description: Engineers develop and maintain variability-intensive software systems without any support of GAI.

GAI Tasks: None.

Manual Tasks: All development activities are done manually or with traditional tools, implemented, and operated by humans.

Challenges: From the perspective of GAI, there are no challenges.

Level 1: GAI support, not specifically for variability. The state of practice shifts towards this level, as solutions, such as Copilot, are increasingly adopted in software engineering.

Description: Limited GAI-support for standard SE-tasks is available, but *without* specialized variability support. Thus, scenarios, such as modifying or analyzing individual features in a product line, are not possible. Due to interactions of multiple variants within the documentation and implementation, the quality is low compared to applications to single-systems.

GAI Tasks: Basic SE tasks, such as code analysis and modification proposals, are available. However, this works mostly within implementation parts in the same feature context. Particularly, we envision code summarization, documentation, proposals, and test derivation (always variability-agnostic) at this level.

Manual Tasks: Humans perform the majority of development tasks. They need to trigger tasks and review results. Everything related to variability is done manually.

Challenges: Challenges are related purely to general software engineering. However, at the same time, the lack of quality (reliability) and the fact that only implementations of rather limited size can be addressed significantly limits the usefulness of GAI.

Level 2: Basic GAI support for variability. While existing GAI-systems may be able to deal with variability, they are not (pre-)trained or adapted to these tasks. We assume an additional focus on variability support is present at this level.

Description: GAI-support is explicitly variability-aware. In particular, this enables systems to summarize multiple, related artifacts from different projects and extract variability from this or allow for integrating a variability model in maintaining artifacts. However, the levels of quality, scalability, and granularity are still low, allowing only limited development support. Due to the restricted size of documents and source code, especially multi-project integration tasks may hardly be supported.

GAI Tasks: Basic SE tasks, even when involving variability, are supported. These tasks may be basic code analysis, generation, or modification proposals [1, 2]. This may also involve multiple related artifacts, such as the variability model plus source code on a small scale. All tasks should be strictly overseen by humans.

Manual Tasks: Most of the development is performed by humans who trigger and guide the tasks and review results.

Challenges: Additional support for variability, including different forms and strategies for its creation and maintenance, must be provided. Dealing with multiple alternative implementations simultaneously and effectively handling variability information poses significant challenges.

Level 3: Advanced GAI support for variability. As capabilities of GAI for general software engineering and SPLE progress, this level establishes more advanced support.

Description: GAI goes beyond the previous levels along multiple dimensions. This includes the scale of the artifacts (or systems) that the tools can deal with, as well as the quality of the produced results. GAI can support a significant breadth of tasks, particularly, evolution and migration tasks at a significant scale. This level of task support might only be achieved if the GAI becomes an experienced user of existing PLE tools.

GAI Tasks: GAI can support most SE and SPLE tasks. Some *basic surveillance* level of review and checking is still required.

Manual Tasks: Humans still provide the basic triggers for development and provide surveillance, but many standard (and advanced) development tasks can be performed by the GAI.

Challenges This transition is highly challenging. It requires an improved scale that goes significantly beyond today's capabilities and high quality, which probably may not be provided by GAI

Level	Automation	Scale	Task Type	Quality
L4: Full GAI support for variability	full	potentially unlimited	any (productive)	expert
L3: Advanced GAI support for variability	medium	limited (multiple small projects)	any	high
L2: Basic GAI support for variability	medium	limited (small projects)	any	low
L1: No GAI support for variability	low	limited (small files)	any	low
L0: No GAI	none	–	–	–

Table 1: Levels of GAI-support in engineering variability-intensive systems.

alone. Support by existing tools or other advances, for instance, agent-based approaches (multiple GAIs), are necessary.

Level 4: Full GAI support for variability. As the most advanced level of GAI support for product lines, we envision that GAI-tools solve SPLE tasks fluently and with competence similar to SPLE experts.

Description: GAI can solve an extensive range of problems, mostly autonomously with expert quality. The GAI addresses problems at a potentially unlimited scale, such as multiple systems.

GAI Tasks: GAI solves any SPLE task like an SPLE expert.

Manual Tasks: Humans still provide basic triggers for development, but GAI can perform development tasks autonomously.

Challenges Challenges arise from perfecting the results' quality and the need for tool usage. Specialized training for the tools may be necessary, too. Another challenge resides in the fact that this level may only be achieved or accepted within specific domains, e.g., either on the application or the technology side.

4 RESEARCH AGENDA

This section sketches strategies to increase the levels in engineering variability-intensive software systems powered by GAI. As the levels in Table 1 are defined in terms of four main dimensions, we discuss how to move forward along these dimensions.

Almost any *task* can be given to a GAI. To achieve satisfactory results, the scale and quality of GAI must be increased. The scale of the processed in- and output may, in turn, influence the quality. To reliably solve variability-related tasks with little human interaction, we discuss how scale and quality can be improved.

4.1 Increasing Scale

The problem of limited scale is not specific to SPLE or SE but relevant for all kinds of GAI applications. Thus, we separately discuss generic solutions and SE-specific ones.

Generic Solutions: To deal with the limited context window, the following general solutions are currently discussed in the GAI-community (beyond growing the model):

- Retrieval-based techniques aim at retrieving relevant parts of information at each step [45].
- Fine-tuning can integrate the relevant knowledge into an LLM. However, fine-tuning always creates a specific GAI variant, threatening its generalization capabilities, and requires a significant amount of resources [11].
- Explicit context management, where the LLM manages the memory access to the context itself [32].

SE Solutions: SE-specific techniques to reduce the input and output size are similar to those software engineers use today:

Abstraction: Abstract (structural or behavioral) models are used to reason on systems while reducing the amount of required information. Due to the reduced size, they may be beneficial compared to full project documentation (including code).

Slicing: Based on specific input tasks, slicing may reduce the total amount of information to a level that can be successfully handled.

SPLE Solutions: Similar to SE-techniques, research on slicing variable software [20, 25] and filtered editing [7, 27] shall be exploited to circumvent missing scale. For instance, given the Linux kernel as context, GAI research should examine how to partition the context needed for the task; e.g., by using slicing and filtering techniques. Consequently, SPLE research needs to examine how much GAI can be trained to optimize the context window size on its own or to use respective tools, e.g., inside a GAI process line.

4.2 Increasing Task

While GAI can handle any task, reaching the productive usage of GAI may require the following research steps:

Tool Integration: As GAI can employ tools, future research should examine how to integrate existing tools and solutions, for instance, feature model analysis, in the solution pipeline for an SPLE task. Particularly, SPLE research needs to examine:

(1) How and in what situations should existing tools be integrated?
(2) How can traceability between tasks computed by the GAI and those computed by classical SPLE engineering be ensured?
(3) How can self-optimization help to assure quality and increase reliability?

4.3 Increasing Quality

To increase quality, three main directions need to be explored: 1) reducing the probability of generating erroneous result through GAI, 2) testing the statistical quality, and 3) testing the concrete result of a GAI.

Pre-Training and Fine-tuning: Pre-training a GAI on variability-specific tasks represents one solution similar to how GAI-models are pre-trained for source code tasks, such as CodeT5 [40]. Fine-tuning may serve to learn one or multiple specific tasks, such as locating or implementing features, but has downsides as it may impede the generalization power of a GAI. Both techniques aim at reducing the probability of false results.

Support Existing Capabilities: Prompt and Model-Based Engineering Prompting techniques may increase the accuracy of the generated result [34, 42]. For instance, chain-of-thought prompting is a current strategy, which requires the GAI to explain how the result came about. Even for domain-specific applications, such as control-code generation [26], the quality of the results improves significantly but the technique still lacks automation [15]. Using

models as abstractions and intermediary steps before implementing source code may increase the accuracy of the generated results. Thus, not only creating feature models and using them as context, but also 150% models, which represent the platform source code may serve as intermediary steps to increase the quality of maintaining variability-intensive software systems.

Data Set (Training) and Metrics (Statistics) for Testing: As GAI is trained on code phrases implemented in 'modern' programming languages, state-of-the-art benchmarks may not be applicable anymore. For instance, the ESPLA catalog contains product variants and their migration to product lines in established programming language (e.g., Java and C++), however, misses subjects implemented in programming languages, such as Rust or Kotlin, which are also used to build software families. Furthermore, as (currently) GAI-models are limited by the context window, real-world benchmarks, such as BusyBox or the Linux kernel, are too large for training and testing the quality of the outcome.

Consequently, establishing new benchmarks is a necessity. Additionally, typically implemented solutions are not unique. Therefore, (new) metrics are required that cannot only assess the quality of generated source code (e.g., as CodeBLEU [36]) but also identify whether features and their interactions are handled correctly.

Testability GAI produces non-deterministic results that must be evaluated [16, 26]. While black-box testing is a common method for software quality assurance, a major issue are missing requirements and specifications towards the results of GAI [16]. Neuron coverage testing adapted to variability-intense networks [44] may be one step to validate results regarding the training input. To achieve a higher level of trust and reliability, either the reasoning behind a solution needs to be clear or certainty values need to be provided alongside the produced result. Consequently, SPLE research needs to develop strategies to explain or compute the probability of generating a correct result through the GAI.

5 CONCLUSION

With this vision paper, we propose a research agenda to enable and increase the support of GAI for multiple tasks within SPLE. To this end, we introduce the four dimensions *scale*, *quality*, *task type*, and *automation*, which influence the adoption of GAI for SPLE tasks. Based on minimum requirements for these dimensions, we define levels for adopting GAI in the GAI4SPL model, and propose an agenda addressing research gaps for each dimension. By empowering SPLE with GAI, we envision developers to implement configurable systems more efficiently and in better quality, and to increase the functionality and reliability of those systems.

REFERENCES

[1] Mathieu Acher, José Galindo Duarte, and Jean-Marc Jézéquel. 2023. On Programming Variability with Large Language Model-Based Assistant. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A* (Tokyo, Japan) *(SPLC '23)*. ACM, New York, NY, USA, 8–14. https://doi.org/10.1145/3579027.3608972

[2] Mathieu Acher and Jabier Martinez. 2023. Generative AI for Reengineering Variants into Software Product Lines: An Experience Report. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B* (Tokyo, Japan) *(SPLC '23)*. ACM, New York, NY, USA, 57–66. https://doi.org/10.1145/3579028.3609016

[3] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, undefinedtefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. 2014. Flexible Product Line Engineering with a Virtual Platform. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 532–535. https://doi.org/10.1145/2591062.2591126

[4] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-642-37521-7

[5] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B. Ashok, and Shashank Shet. 2023. CodePlan: Repository-level Coding using LLMs and Planning.

[6] Christopher M. Bishop. 2007. *Pattern recognition and machine learning, 5th Edition*. Springer.

[7] Paul Maximilian Bittner, Alexander Schultheiß, Sandra Greiner, Benjamin Moosherr, Sebastian Krieter, Christof Tinnes, Timo Kehrer, and Thomas Thüm. 2023. Views on Edits to Variational Software. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A* (Tokyo, Japan) *(SPLC '23)*. ACM, New York, NY, USA, 141–152. https://doi.org/10.1145/3579027.3608985

[8] Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. 2022. Chapter 13 Engineering AI Systems - A Research Agenda. In *Accelerating Digital Transformation - 10 Years of Software Center*, Jan Bosch, Jan Carlson, Helena Holmström Olsson, Kristian Sandahl, and Miroslaw Staron (Eds.). Springer, 407–425. https://doi.org/10.1007/978-3-031-10873-0_18

[9] Eric Boyd. 2023. Introducing GPT-4 in Azure OpenAI Service. https://azure.microsoft.com/en-us/blog/introducing-gpt4-in-azure-openai-service/.

[10] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling* 22, 3 (2023), 781–793. https://doi.org/10.1007/s10270-023-01105-5

[11] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. LongLoRA: Efficient Fine-Tuning Of Long-context Large Language Models. https://doi.org/10.48550/arXiv.2309.12307

[12] Paul Clements and Linda Northrop. 2015. *Software Product Lines: Practices and Patterns* (1 ed.). Pearson Education, USA.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. ACL, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[14] Christof Ebert and Panos Louridas. 2023. Generative AI for Software Practitioners. *IEEE Software* 40, 4 (2023), 30–38. https://doi.org/10.1109/MS.2023.3265877

[15] 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:2310.03533 [cs.SE]

[16] Michael Felderer and Rudolf Ramler. 2021. Quality Assurance for AI-Based Systems: Overview and Challenges (Introduction to Interactive Session). In *Software Quality: Future Perspectives on Software Engineering Quality*. Springer International Publishing, 33–42. https://doi.org/10.1007/978-3-030-65854-0_3

[17] Martin Fowler. 2011. *Domain-Specific Languages*. Addison-Wesley.

[18] José A. Galindo, Antonio J. Dominguez, Jules White, and David Benavides. 2023. Large Language Models to Generate Meaningful Feature Model Instances. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A* (Tokyo, Japan) *(SPLC '23)*. ACM, New York, NY, USA, 15–26. https://doi.org/10.1145/3579027.3608973

[19] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 10764–10799. https://proceedings.mlr.press/v202/gao23f.html

[20] Lea Gerling and Klaus Schmid. 2020. Syntax-Preserving Slicing of C-Based Software Product Lines: An Experience Report. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems* (Magdeburg, Germany) *(VaMoS '20)*. ACM, New York, NY, USA, Article 17, 5 pages. https://doi.org/10.1145/3377024.3377029

[21] Github 2023. Copilot. https://github.com/features/copilot Visited: Oct 18, 2023.

[22] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE]

[23] Mladan Jovanovic and Mark Campbell. 2022. Generative Artificial Intelligence: Trends and Prospects. *Computer* 55, 10 (2022), 107–112. https://doi.org/10.1109/MC.2022.3192720

[24] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Carnegie-Mellon University, Software Engineering Institute.

[25] Frederik Kanning and Sandro Schulze. 2014. Program Slicing in the Presence of Preprocessor Variability. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*. IEEE Computer Society, 501–505. https://doi.org/10.1109/ICSME.2014.82

[26] Heiko Koziolek, Sten Gruener, and Virendra Ashiwal. 2023. ChatGPT for PLC/DCS Control Logic Generation. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1–8. https://doi.org/10.1109/ETFA54631.2023.10275411

[27] Lukas Linsbauer, Felix Schwägerl, Thorsten Berger, and Paul Grünbacher. 2021. Concepts of variation control systems. *J. Syst. Softw.* 171 (2021), 110796. https://doi.org/10.1016/j.jss.2020.110796

[28] Roberto E. Lopez-Herrejon, Jabier Martinez, Wesley Klewerton Guez Assunção, Tewfik Ziadi, Mathieu Acher, and Silvia Regina Vergilio (Eds.). 2023. *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*. Springer International Publishing. https://doi.org/10.1007/978-3-031-11686-5

[29] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, 931–937. https://doi.org/10.1145/3545945.3569785

[30] Changan Niu, Chuanyi Li, Bin Luo, and Vincent Ng. 2022. Deep Learning Meets Software Engineering: A Survey on Pre-Trained Models of Source Code. (7 2022), 5546–5555. https://doi.org/10.24963/ijcai.2022/775 Survey Track.

[31] Society of Automotive Engineers. 2021. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. https://www.sae.org/standards/content/j3016_202104/.

[32] Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2310. MemGPT: Towards LLMs as Operating Systems.

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. ACL, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[34] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True Few-Shot Learning with Language Models. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021 (NeurIPS 2021)*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 11054–11070. https://proceedings.neurips.cc/paper/2021/hash/5c04925674920eb58467fb52ce4ef728-Abstract.html

[35] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21 (2020), 140:1–140:67. http://jmlr.org/papers/v21/20-074.html

[36] Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. *CoRR* abs/2009.10297 (2020). arXiv:2009.10297 https://arxiv.org/abs/2009.10297

[37] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. TPTU: Task Planning and Tool Usage of Large Language Model-based AI Agents.

[38] Rosalia Tufano, Luca Pascarella, and Gabriele Bavota. 2023. Automating Code-Related Tasks Through Transformers: The Impact of Pre-training. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2425–2437. https://doi.org/10.1109/ICSE48619.2023.00203

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[40] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. ACL, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[41] Andrzej Wasowski and Thorsten Berger. 2023. *Domain-Specific Languages: Effective Modeling, Automation, and Reuse*. Springer Int. Publishing. https://doi.org/10.1007/978-3-031-23669-3

[42] Albert Webson and Ellie Pavlick. 2022. Do Prompt-Based Models Really Understand the Meaning of Their Prompts?. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. ACL, Seattle, United States, 2300–2344. https://doi.org/10.18653/v1/2022.naacl-main.167

[43] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research* 2022 (2022). https://openreview.net/forum?id=yzkSU5zdwD

[44] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. 2019. DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 146–157. https://doi.org/10.1145/3293882.3330579

[45] Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Retrieval Meets Long Context Large Language Models.

[46] Zibin Zheng, Kaiwen Ning, Jiachi Chen, Yanlin Wang, Wenqing Chen, Lianghong Guo, and Weicheng Wang. 2023. Towards an Understanding of Large Language Models in Software Engineering Tasks. arXiv:2308.11396 [cs.SE]