



**HAL**  
open science

# A Demonstration of End-User Code Customization Using Generative AI

Mathieu Acher

► **To cite this version:**

Mathieu Acher. A Demonstration of End-User Code Customization Using Generative AI. VAMOS 2024 - 18th International Working Conference on Variability Modelling of Software-Intensive Systems, Feb 2024, Bern, Switzerland. pp.1-6, 10.1145/3634713.3634732 . hal-04312909

**HAL Id: hal-04312909**

**<https://hal.science/hal-04312909v1>**

Submitted on 28 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# A Demonstration of End-User Code Customization Using Generative AI

Mathieu Acher

Univ Rennes, IRISA, Inria, CNRS, IUF

Rennes, France

mathieu.acher@irisa.fr

## ABSTRACT

Producing a variant of code is highly challenging, particularly for individuals unfamiliar with programming. This demonstration introduces a novel use of generative AI to aid end-users in customizing code. We first describe how generative AI can be used to customize code through prompts and instructions, and further demonstrate its potential in building end-user tools for configuring code. We showcase how to transform an undocumented, technical, low-level TikZ into a user-friendly, configurable, Web-based customization tool written in Python, HTML, CSS, and JavaScript and itself configurable. We discuss how generative AI can support this transformation process and traditional variability engineering tasks, such as identification and implementation of features, synthesis of a template code generator, and development of end-user configurators. We believe it is a first step towards democratizing variability programming, opening a path for end-users to adapt code to their needs.

## CCS CONCEPTS

• **Software and its engineering** → **Software product lines; Context specific languages**; • **Computing methodologies** → **Machine learning approaches**.

## KEYWORDS

variability, end-user programming, software product lines, generator, generative AI, LLM, code synthesis, customization

## 1 INTRODUCTION

Customizing code is highly challenging, particularly for individuals unfamiliar with programming. The customization of code involves modifying or extending an existing piece of software code to create a new, different variant that has, e.g., different features or functionalities. The process involves different activities. There is first the need to understand existing code. A code can be more or less documented, written by someone else, or simply challenging to comprehend. There is also the need to master different programming languages, technologies, frameworks, or libraries, each with a specific syntax. A person familiar with one language may struggle to read, let alone modify, code written in a different language. Furthermore, each modification can potentially affect other parts of the code. In general, there is a gap between the intention and the technical realization. Programming per se is already a difficult task and variability further increases this complexity since multiple features, code variations, and an exponential number of possible variants should be implemented [4, 27].

In this demonstration paper, we defend the idea that generative AI and *large language models (LLMs)* can be leveraged to support

end-users in the customization of code. We first show how generative AI can be used to customize code through prompts and instructions. In a sense, LLMs act as a new variability tool capable of transforming a high-level specification (prompt) into variable code or features. We then demonstrate LLMs' potential in *building* end-user tools for configuring code. Such dedicated tools for end-users do not operate over technical code and provide simplified interfaces for customization.

Throughout the demonstration, we aim to show how to transform an undocumented, technical, low-level TikZ code into a user-friendly, configurable version via a Web-based customization tool. This transformation process involves identifying and implementing explicit features, leading to a template code generator accessible to the end-user. Moreover, the Web application, written in Python, HTML, CSS, and JavaScript, is itself customizable, thanks to end-user interactions with generative AI to control the layout, the image border or size. We believe it is a first step towards democratizing variability programming, opening a path for end-users to adapt code to their needs.

### 1.1 Related work

LLMs such as BERT [9], GPT-2 [34], GPT-3 [5], GPT-4 [33], PaLM [8], to name a few, are scaling to support a variety of tasks such as text generation, question-answering, arithmetic on numbers, etc. [10, 15, 21, 36, 39]. In software engineering [12, 18], code assistants have been proposed like Alphacode, CodeParrot or Codex [7, 24] and are now deployed at scale, such as GitHub Copilot [16].

The application of machine learning in Software Product Line (SPL) and variability engineering encompass various use cases [3, 17, 20, 22, 30–32, 37, 38, 40–43, 45]. These use cases include performance prediction, optimization, specialization, debugging, detection of configuration-related errors, etc. In the current study, we focus on the potential of generative artificial intelligence and LLMs that rely on transformers' technology, an advanced deep learning architecture initially developed for Natural Language Processing (NLP) applications. The adoption of NLP in SPL engineering has drawn several interests, predominantly for domain analysis and requirements engineering [6, 13, 19, 28, 29, 35]. The usage of LLMs into variability has recently started to catch attention. Acher and Martinez [2] explored how SPL can be synthesized out of variants. Galindo et al. [14] evaluated the ability of LLMs to synthesize valid instances of feature models. Acher et al. [1] introduced the use of LLMs to assist developers in implementing variability in different programming languages and mechanisms. In this demonstration paper, (1) we specifically target end-users' scenarios; (2) we span different kinds of artifacts (not only TikZ) and replicate on different codes, engineering different generators and configurators; (3)

we give more details about the process (feature location, template synthesis, etc.).

End-user software engineering [23, 25], a branch of software engineering, focuses on empowering individuals who are not professional programmers to design, implement, and maintain software applications. This paper specifically considers how end-users can implement variability with the assistance of generative AI. While traditional works in end-user programming have typically focused on providing dedicated tools and notations, this research proposes two novel directions: first, relying on LLMs to construct these specialized tools, and second, considering LLMs themselves as a tool and an interface for end-users. Additionally, our approach is tailored to assist users in dealing with "unfamiliar" software-intensive artifacts, aiming to enable or simplify their customization.

## 1.2 Audience

We believe a demonstration of generative AI technology can be of interest for the variability community, especially when specifically targeting end-users and variability. We are targeting two kinds of users: (1) end-users that want to synthesize variants of a piece of (unfamiliar) code, automatically generate and configure variants with dedicated tools; (2) developers that want to build generators, configurators, or SPLs. Both can benefit from generative AI for some variability tasks (feature location, template synthesis, variation implementation, automated derivation of variants). The approach is also applicable to different kinds of technical artefacts and code: TikZ, HTML, CSS, JavaScript, SVG, SCAD, etc.

We share sessions (prompts used, sequences of interactions, etc.) and resulting codes:

<https://github.com/acherm/progvary-withgpt/>

The repository contains additional examples that go beyond the end-to-end demonstration we plan to conduct. During the demonstration, the sessions will be presented, reviewed and discussed. We believe our prompts can be reused and revisited on top of different LLMs and generative AI technologies.

## 2 FROM UNFAMILIAR, NON-CUSTOMIZABLE CODE TO END-USER CUSTOMIZATION WITH LLM

We are tasked with the challenge of personalizing a squid image, developed in TikZ, a commonly utilized package within  $\LaTeX$  for creating illustrations. A visual of the squid is rendered in the bottom right of Figure 2. Due to the lack of space, we refer to Listing 1 that gives a rough idea of the TikZ code. It should be noted that the original TikZ code is *not subject to variation and here to produce a unique squid*. The 40 lines of code are arguably hard to understand: there is no comment for describing the different parts of the squid while TikZ code combines different low-level primitives like fill, ellipse, etc. The code was found in a public forum [44], created by a TikZ expert, without many explanations.

Hence, the challenge is as follows: transforming the unfamiliar, non-variable code into something configurable by end-users.

### Listing 1: Generated template TikZ code (excerpt)

```
squid/.pic={
\begin{scope}[x=3cm/480,y=3cm/480]
```

```
\useasboundingbox (-256, -256) (256, 256);
\reflect[split={DeepOrange500 and DeepOrange700}]{
(% if aggressive_shading %)
\fill [left color=fill color, right color=fill color
!40!black, middle color=fill color!70!black]
(0,96) ellipse [x radius=96, y radius=128];
(% else %)
\fill [fill color] (0,96) ellipse [x radius=96, y
radius=128];
(% endif %)
% \fill [fill color] (0,96) ellipse [x radius=96, y
radius=128];
% \fill [fill color] (0,96) ellipse [x radius=96, y
radius=128];
(% if crazy_tentacles %)
(% if aggressive_shading %)
\path [left color=fill color, right color=fill color
!40!black, middle color=fill color!70!black,
shading angle=90]
(% else %)
...
(% endif %)
(% if original_tentacles %) %%% original tentacles
(% if aggressive_shading %)
\path [left color=fill color, right color=fill color
!40!black, middle color=fill color!70!black,
shading angle=90]
(% else %)
\fill [fill color]
(% endif %)
(72,-16)
.. controls ++(270:128) and ++(270:160) .. (0,
-16);
(% endif %)
\fill [fill color] (64, 16) circle [radius=48];
\fill [(( outer_eye_color ))] (64, 16) circle [radius
=32];
\fill [(( inner_eye_color ))] (64, 16) circle [radius
=12];
\fill [fill color] (-4, -20)
.. controls ++(270:256) and ++(225: 64) .. (128,-192)
}
(% if prominent_mouth %)
\fill [DeepOrange900] (-40,-40) arc (150:390:40 and 20)
arc (390:150:40 and 50);
(% else %)
\fill [DeepOrange900]
(-40,-40) arc (180:360:40 and 16) arc (360:180:40 and
40);
(% endif %)
% \fill [DeepOrange300]
(% for spot in spots %)
(\fill [ (( spot.color )) ] ( (( spot.x )) , (( spot.y ))
) ) circle [radius=(( spot.radius ))] ;
(% endfor %)
\end{scope}
},
```

### 2.1 Customizing code with LLM

We now detail how LLMs assisted us in customizing the TikZ code. We have used<sup>1</sup> ChatGPT-4 [33], using the Web chat interface of OpenAI. Using "prompts" (referred to as

(User) green boxes

<sup>1</sup>Recordings of the sessions and interactions will be made available for sharing.

), which are natural language instructions that can incorporate existing code, LLMs automatically generates programs and potential explanations (see

(LLM) red boxes

hereafter). We start asking about the TikZ code.

(User) What's this TikZ code is doing?

(LLM) This TikZ code defines a custom drawing or picture called "squid" using the ".pic" syntax. The code specifies the coordinates, shapes, and colors of various elements to create the image of a squid. The code consists of several parts:

```
\begin{scope}[x=3cm/480,y=3cm/480]: This line begins a
  scope environment and scales the x and y coordinates
  to create a proportional drawing.
...
\reflect[split={DeepOrange500 and DeepOrange700}]{...}:
  This block of code defines the symmetrical parts of
  the squid, such as the body and tentacles. It uses a
  custom reflect command that is not part of the
  standard TikZ package (you will need the
  corresponding macro definition for this to work).
  The split option specifies the colors for the left
  and right halves of the squid.
...

```

A line-by-line technical explanation of the TikZ code is not that helpful, especially for end-users. The level of details is too low, and the explanations are not really actionable, especially for relating code elements to visual aspects.

Instead, we can directly ask what kinds of customizations we want.

(User) can you vary the eye of the squid with the blue color? tell me what varies in the TikZ code

(LLM) To change the color of the squid's eye to blue, you'll need to modify the fill commands that define the eye. The lines that define the eye are:

```
\fill [Grey50] (64, 16) circle [radius=32];
\fill [BlueGrey900] (64, 16) circle [radius=12];
To change the eye color to blue, you can replace the colors
...
\fill [blue] (64, 16) circle [radius=32];
\fill [blue!60!white] (64, 16) circle [radius=12];
...

```

The answer is correct. The visual and high-level description ("eyes") has been mapped to a precise line of code. The proposed variation is syntactically correct and only focuses on code parts that need to be customized. End-users can locate and implement other features related to the number of spots, the tentacles, etc. A series of prompts can be fed to an LLM that then returns the piece of code to modify and the suggested modification corresponding to a feature.



Figure 1: Three variants of a squid: original configuration, manual configuration, random configuration

## 2.2 Building customizable tools with LLM

So far, end-users can customize parts of the code with LLM. However, there are some limitations:

- end-users have to manually integrate the code and follow technical instructions;
- end-users can apply individual customizations but not their combinations;
- interactions of LLMs can change from time to time, even with the same prompt. LLMs are by construction non-determinist, and so are corresponding suggestions of customization.

An answer to these limitations is to engineer customization tools and build an actual configurable system composed of different features. It has several benefits: (1) the integration of customized transformations can be automated; (2) features can be combined to synthesize variants; (3) the engineering of tools matures the customization knowledge and makes it reliable. Our idea is to build customization tools with LLM.

(User) would it be possible to vary the original code using a templating engine in Python in such a way it is possible to have either eye of the squid with the blue color or the original eye (as in the original TikZ code)? ... can you parameterize the spots with a random color? if not random mode, then it's the default color

Listing 2 provides an excerpt of the Python generator in charge of deriving variants out of template instantiations of Listing 1.

```
from jinja2 import Environment, FileSystemLoader
import random
def random_color():
    color_options = ['Blue500', 'Yellow50', 'Lime900', '
                    DeepOrange900']
    return random.choice(color_options)
...
# Define the random_mode flag (True for random colors,
# False for the default color)
random_mode = True
# Define the spots
o_spots = [
    ...
]

# Render the template with the chosen eye colors/spots
crazy_tentacles = False # True
original_tentacles = False # True
prominent_mouth = True
agressive_shading = True

```

```
# Render the template with the chosen values
with open('squid_output.tex', 'w') as output_file:
    output_file.write(template.render(eye_colors, spots=
        extra_spots, crazy_tentacles=crazy_tentacles,
        prominent_mouth=prominent_mouth,
        aggressive_shading=aggressive_shading,
        original_tentacles=original_tentacles))
```

### Listing 2: Generated Python generator

Once individual features have been implemented, there is then the need to combine all variations together. As TikZ has no obvious built-in mechanisms to support variation, we can implement a template-based code generator with LLM. Listing 1 shows an excerpt of the template. It should be noted that the prompt comes after the series of prompts for locating and implementing features (9 features were identified at the time of the session).

## 2.3 Further customization

The Python generator can then be used to synthesize a gallery of diverse squids (e.g., with random colors and scales). The generator can also be controlled through a Web application (see Figure 2). The client side offers to users the control to configure different features of a squid, while the server side internally calls the template-based generator and renders the images in SVG. During the demonstration, we will describe (1) the use of the Web configurator; (2) the effort needed to engineer it. Moreover, end-users can also customize other *aspects* of the Website. In the top, right-hand side of Figure 2, there is a configuration panel that controls different features: (1) layout: vertical, horizontal, or hybrid, having an effect on how the image is located w.r.t. the squid configurator panel; (2) whether there is a border on top of the image. This configuration perspective operates over the HTML and CSS. It has been typically obtained through high-level interactions describing some refinements of the Website.

In the bottom, right-hand side of Figure 2, there is another configuration panel that can customize the image (1) its relative position and the ability to move from left to right the squid; (2) its size; (3) its padding; (4) the ability to flip or mirror the image; (5) its rotation. This configuration aspect of the squid internally operates over an SVG artefact. For each variation point, high-level instructions have been formulated to specify the intent. The variations have then been implemented by the LLM through the synthesis of JavaScript functions that manipulate the SVG. Finally, at the center of Figure 2 and below the SVG image, the variant of the TikZ code is depicted. More technical users can benefit from this code, and the idea is that end-users never interact with it. Overall, all these features and variations, including the underlying Python, HTML, CSS, and JavaScript have been synthesized through interactions with generative AI.

To summarize, Figure 2 demonstrates the final result of end-user customization of TikZ code, but also with Python, HTML, CSS, and JavaScript. We also show that the approach is applicable to other technical artefacts like SVG.

## 3 THE RISE AND FALL OF LLMS

LLMs can support end-users or developers in the following variability activities, with pros and cons.

**Code understanding.** Gaining a thorough comprehension of a technical artifact is important before customization to ensure any

edits will have the expected effect. Unfortunately, the explanations, though correct, are often too low-level and may not resonate with the intended audience (e.g., users not familiar with TikZ).

**Feature location.** Instead of directly editing the code, users formulate prompts and LLMs can map high-level descriptions into lines of code. The promise is to raise the level of abstraction and gives to end-users the means to express their intent onto a technical artefact. The counterpart is that LLMs can be inaccurate and provide a wrong mapping, with unexpected results. In our experience, several interactions might be needed to guide or correct the LLM. A challenge, and a possible barrier for end-users, is that some interactions require a technical expertise (e.g., to report on compilation errors).

**Feature and variation implementation.** LLMs can synthesize variants of a piece of code, aiding end-users who do not have to write code themselves. For developers, this generative approach not only boosts productivity, but is also advantageous for those unfamiliar with a particular technological domain. A risk is that the generated code can be of bad quality, leading to compilation errors or wrong implementation of a feature, and calling to debug and further interact. Another challenge arises from feature interactions: as the code base expands with added features, managing and understanding the complex interplay between them can become increasingly difficult, especially for end-users that are not supposed to deal with this kind of technical complexity.

**Combination of features and automated derivation.** LLMs can synthesize template-based generators or configurators (including user interfaces) that provide the means to *combine* different features. This task falls beyond the skill set of end-users, whereas the boilerplate code can significantly enhance developers' experience. A counterpart is that LLMs might improperly handle the underlying logics, such as commonalities and dependencies between features. In our end-user scenarios, we did not use LLMs for specifying constraints across features, though some works have emerged to synthesize feature models [14].

**Code integration, back and forth.** Incorporating LLM outcomes require a back-and-forth with the code base. It is an obstacle for fully automating the involvement of end-users. This iterative process often requires continuous switching and adjustments, adding layers of complexity and potential frustration for developers. The emergence of AI-based code environments can mitigate this issue. We envision specialized development environments, geared towards end-users, and capable of parsing, integrating, and executing the suggested modifications.

**Prompts.** Specifying effective, tailored prompts for each intermediate task, ranging from code understanding to variation implementation, is essential. A research direction is to thoroughly evaluate LLMs for their robustness and sensitivity to prompts in realizing variability tasks. Existing studies suggests that prompt variations can have an impact on the quality of the generated code [11, 12, 18, 26]. Our goal is to maintain the possibility of informal, high-level formulations, embracing the spirit of end-user programming. However, it might well be the case that more structure and detail are needed to make an LLM effective. We envision having a systematic prompting strategy that could integrate the informal directives of end-users while offering a structured framework for more technical people.



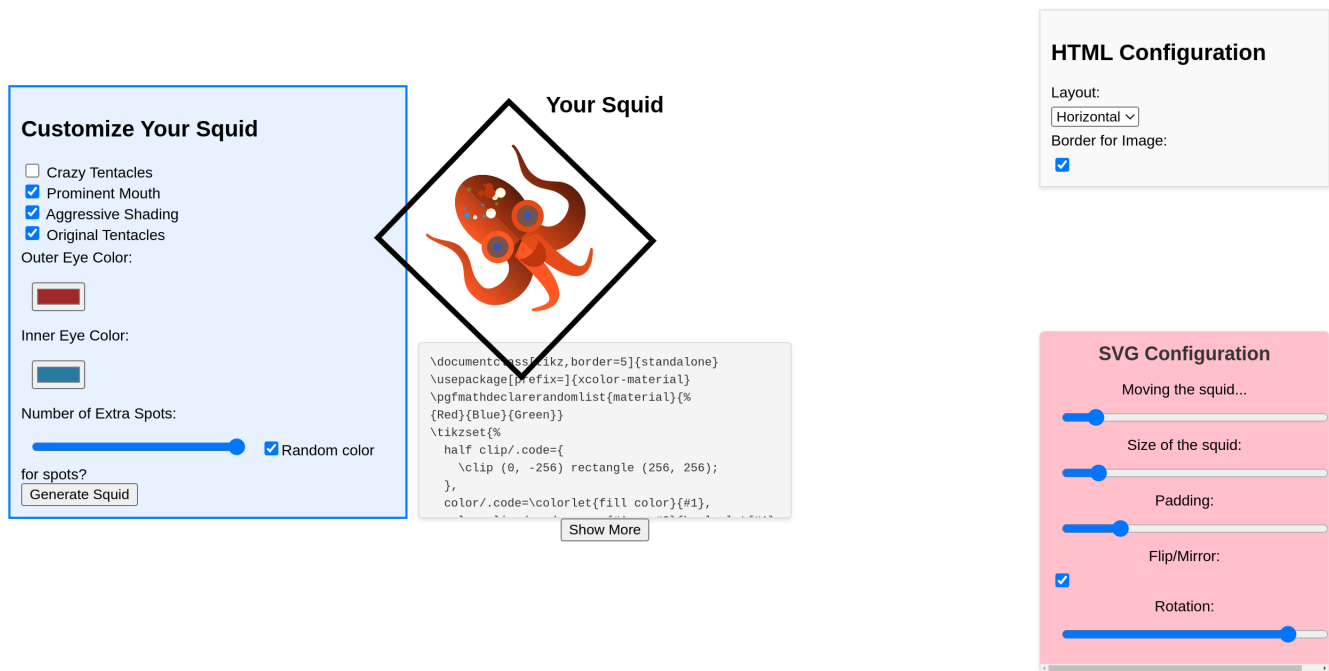


Figure 2: End-user customization of (1) TikZ code (squid) (2) HTML/CSS (Website layout) (3) SVG (post-process over the resulting image). The Python template-based generator produces a variant of the TikZ code, which is displayed, but end-users do not interact with it.

## 4 CONCLUSION

In this demonstration, we plan to showcase how LLMs can be leveraged to support end-users in the customization of code and technical artefacts. We aim to describe how to transform an undocumented, technical, low-level TikZ into a user-friendly, configurable, Web-based customization tool written in Python, HTML, CSS, and JavaScript and itself customizable. Other demonstrations over SVG and OpenSCAD artefacts are also planned. Spanning topics from code understanding to implementing features and offering end-user configurators, the presentations will provide an opportunity to review the benefits and drawbacks of generative AI for supporting variability tasks.

**Acknowledgements.** We thank reviewers for their valuable suggestions for future work.

## REFERENCES

- [1] Mathieu Acher, Jean-Marc Jézéquel, and José A. Galindo. 2023. On Programming Variability with Large Language Model-based Assistant. In *Proceedings of the 27th International Systems and Software Product Line Conference (SPLC 2023) - Research Track (SPLC '23)*. ACM, New York, NY, USA.
- [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, SPLC 2023, Tokyo, Japan, 28 August 2023- 1 September 2023*, Paolo Arcaini, Maurice H. ter Beek, Gilles Perrouin, Iris Reinhartz-Berger, Ivan Machado, Silvia Regina Vergilio, Rick Rabiser, Tao Yue, Xavier Devroey, Mónica Pinto, and Hironori Washizaki (Eds.). ACM, 57–66. <https://doi.org/10.1145/3579028.3609016>
- [3] Juliana Alves Pereira, Hugo Martin, Mathieu Acher, Jean-Marc Jézéquel, Goetz Botterweck, and Anthony Ventresque. 2021. Learning Software Configuration Spaces: A Systematic Literature Review. *Journal of Systems and Software (JSS)* (June 2021). <https://doi.org/10.1145/nnnnnnn.nnnnnnn>
- [4] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer-Verlag.
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [6] Jessie Carbonnel. 2018. *L'analyse formelle de concepts: un cadre structurel pour l'étude de la variabilité de familles de logiciels*. Ph.D. Dissertation. Université Montpellier.
- [7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. *Evaluating Large Language Models Trained on Code*. Technical Report arXiv:2107.03374. arXiv. <https://doi.org/10.48550/arXiv.2107.03374> arXiv:2107.03374 [cs] type: article.
- [8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [10] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathy Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2021. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts. arXiv:2112.06905 [cs.CL] <https://arxiv.org/abs/2112.06905>

- [11] Jean-Baptiste Döderlein, Mathieu Acher, Djamel Eddine Khelladi, and Benoit Combemale. 2023. Piloting Copilot and Codex: Hot Temperature, Cold Prompts, or Black Magic? [arXiv:2210.14699](https://arxiv.org/abs/2210.14699) [cs.SE]
- [12] 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](https://arxiv.org/abs/2310.03533) (2023).
- [13] Alessio Ferrari, Giorgio Oronzio Spagnolo, and Felice dell'Orletta. 2013. Mining commonalities and variabilities from natural language documents. In *SPLC*.
- [14] 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, SPLC 2023, Tokyo, Japan, 28 August 2023- 1 September 2023*, Paolo Arcaini, Maurice H. ter Beek, Gilles Perrouin, Iris Reinhartz-Berger, Miguel R. Luaces, Christa Schwanninger, Shaoukat Ali, Mahsa Varshosaz, Angelo Gargantini, Stefania Gnesi, Malte Lochau, Laura Semini, and Hironori Washizaki (Eds.). ACM, 15–26. <https://doi.org/10.1145/3579027.3608973>
- [15] Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting Numerical Reasoning Skills into Language Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 946–958. <https://doi.org/10.18653/v1/2020.acl-main.89>
- [16] Github. 2021. *GitHub Copilot - Your AI pair programmer*. <https://copilot.github.com>
- [17] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *ASE*.
- [18] 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 preprint arXiv:2308.10620](https://arxiv.org/abs/2308.10620) (2023).
- [19] Nili Itzik and Iris Reinhartz-Berger. 2014. SOVA - A Tool for Semantic and Ontological Variability Analysis. In *Joint Proceedings of the CAISE 2014 Forum and CAISE 2014 Doctoral Consortium*. 177–184. <http://ceur-ws.org/Vol-1164/PaperDemo06.pdf>
- [20] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Akshay Patel, and Yuvraj Agarwal. 2017. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 497–508.
- [21] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How Can We Know What Language Models Know? *Transactions of the Association for Computational Linguistics* 8 (2020), 423–438. [https://doi.org/10.1162/tacl\\_a\\_00324](https://doi.org/10.1162/tacl_a_00324) [arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl\\_a\\_00324/1923867/tacl\\_a\\_00324.pdf](https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00324/1923867/tacl_a_00324.pdf)
- [22] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-Based Sampling of Software Configuration Spaces. In *Proceedings of the International Conference on Software Engineering (ICSE)*.
- [23] Amy J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-User Software Engineering. *ACM Comput. Surv.* 43, 3, Article 21 (apr 2011), 44 pages. <https://doi.org/10.1145/1922649.1922658>
- [24] Thomas Wolf, Lewis Tunstall, Leandro von Werra. 2022. *Natural Language Processing with Transformers, Revised Edition [Book]*. <https://www.oreilly.com/library/view/natural-language-processing/9781098136789/> ISBN: 9781098136796.
- [25] Wendy E. Mackay. 1990. Patterns of Sharing Customizable Software. In *Proceedings of the 1990 ACM Conference on Computer-Supported Cooperative Work (Los Angeles, California, USA) (CSCW '90)*. Association for Computing Machinery, New York, NY, USA, 209–221. <https://doi.org/10.1145/99332.99356>
- [26] Antonio Mastropaolo, Luca Pascarella, Emanuela Guglielmi, Matteo Ciniselli, Simone Scalabrino, Rocco Oliveto, and Gabriele Bavota. 2023. On the robustness of code generation techniques: An empirical study on github copilot. [arXiv preprint arXiv:2302.00438](https://arxiv.org/abs/2302.00438) (2023).
- [27] Gabriela K Michelon, Wesley KG Assunção, David Obermann, Lukas Linsbauer, Paul Grünbacher, and Alexander Egyed. 2021. The life cycle of features in highly-configurable software systems evolving in space and time. In *Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. 2–15.
- [28] Daniel-Jesus Munoz, Dilian Gurov, Mónica Pinto, and Lidia Fuentes. 2021. Category Theory Framework for Variability Models with Non-functional Requirements. In *Advanced Information Systems Engineering - 33rd International Conference, CAISE 2021, Melbourne, VIC, Australia, June 28 - July 2, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12751)*, Marcello La Rosa, Shazia W. Sadiq, and Ernest Teniente (Eds.). Springer, 397–413. [https://doi.org/10.1007/978-3-030-79382-1\\_24](https://doi.org/10.1007/978-3-030-79382-1_24)
- [29] Alexandr Murashkin, Michal Antkiewicz, Derek Rayside, and Krzysztof Czarnecki. 2013. Visualization and exploration of optimal variants in product line engineering. In *17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26 - 30, 2013*, Tomoji Kishi, Stan Jarzabek, and Stefania Gnesi (Eds.). ACM, 111–115. <https://doi.org/10.1145/2491627.2491647>
- [30] I Made Murwantara, Behzad Bordbar, and Leandro L. Minku. 2014. Measuring Energy Consumption for Web Service Product Configuration. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services (Hanoi, Viet Nam) (iiWAS)*. ACM, New York, NY, USA, 224–228.
- [31] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using Bad Learners to Find Good Configurations. In *Proceedings of the European Software Engineering Conference/Foundations of Software Engineering (ESEC/FSE)*. 257–267.
- [32] Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2018. Finding Faster Configurations Using Flash. *IEEE Transact. on Software Engineering* (2018).
- [33] OpenAI. 2023. GPT-4 Technical Report. [arXiv:2303.08774](https://arxiv.org/abs/2303.08774) [cs.CL]
- [34] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
- [35] Iris Reinhartz-Berger, Arnon Sturm, and Yair Wand. 2013. Comparing functionality of software systems: An ontological approach. *Data Knowl. Eng.* 87 (2013), 320–338. <https://doi.org/10.1016/j.datak.2012.09.005>
- [36] Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, 379–389. <https://doi.org/10.18653/v1/D15-1044>
- [37] A. Sarkar, Jianmei Guo, N. Siegmund, S. Apel, and K. Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems (T). In *ASE'15*.
- [38] Norbert Siegmund, Alexander Grebhahn, Christian Kästner, and Sven Apel. 2015. Performance-Influence Models for Highly Configurable Systems. In *ESEC/FSE'15*.
- [39] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adria Garriga-Alonso, et al. 2022. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. [arXiv preprint arXiv:2206.04615](https://arxiv.org/abs/2206.04615) (2022).
- [40] Shaghayegh Tavassoli, Carlos Diego Nascimento Damasceno, Ramtin Khosravi, and Mohammad Reza Mousavi. 2022. Adaptive behavioral model learning for software product lines. In *SPLC '22: 26th ACM International Systems and Software Product Line Conference, Graz, Austria, September 12 - 16, 2022, Volume A, Alexander Felternig, Lidia Fuentes, Jane Cleland-Huang, Wesley K. G. Assunção, Andreas A. Falkner, Maider Azanza, Miguel Á. Rodríguez Luaces, Megha Bhushan, Laura Semini, Xavier Devroey, Cláudia Maria Lima Werner, Christoph Seidl, Viet-Man Le, and José Miguel Horcas (Eds.)*. ACM, 142–153. <https://doi.org/10.1145/3546932.3546991>
- [41] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, and Olivier Barais. 2017. Learning-Contextual Variability Models. *IEEE Software* 34, 6 (Nov. 2017), 64–70. <https://hal.inria.fr/hal-01659137>
- [42] Paul Temple, José Angel Galindo Duarte, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using Machine Learning to Infer Constraints for Product Lines. In *Software Product Line Conference (SPLC'16)*. Beijing, China. <https://doi.org/10.1145/2934466.2934472>
- [43] Maurice H. Ter Beek, Axel Legay, Alberto Lluch Lafuente, and Andrea Vandin. 2020. A Framework for Quantitative Modeling and Analysis of Highly (Re)configurable Systems. *IEEE Transactions on Software Engineering* 46, 3 (2020), 321–345. <https://doi.org/10.1109/TSE.2018.2853726>
- [44] Stackexchange thread about TikZ. 2023. <https://tex.stackexchange.com/questions/387047/the-duck-pond-showcase-of-tikz-drawn-animals-ducks>.
- [45] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. 2015. Empirical Comparison of Regression Methods for Variability-Aware Performance Prediction. In *SPLC'15*.