AthenaLLM: Supporting Experiments with Large Language Models in Software Development

Benedito de Oliveira
Federal University of Pernambuco
Recife, Brazil

Fernando Castor
Federal University of Pernambuco
Recife, Brazil
University of Twente
Enschede, The Netherlands

ABSTRACT

Existing studies on the use of Large Language Models (LLMs) in software development leverage methodologies that limit their scalability and require intensive manual data collection and analysis, for example, due to the use of video data or think-aloud protocols. We propose the use of a specialized tool capable of automatically collecting fine-grained, relevant data during experiments and case studies. It enables researchers to understand for example how often participants accept or reject suggestions made by LLMs and what kinds of prompts are more likely to trigger accepted suggestions, even in studies targeting a large number of participants. We implement this idea as a Visual Studio Code plugin named AthenaLLM [1]. It mimics the functionalities of GitHub Copilot and offers seamless integration with OpenAI API models like GPT-4 and GPT-3.5, and compatibility with other models providing an OpenAI-compatible API, e.g., Vicuna [6]. It automatically collects data at a fine level of granularity and covers both the interactions of developers with their IDE, e.g., all changes made in the code, and the products of such interactions, e.g., the generated code, when accepted. Thus, the proposed approach also reduces bias that the experimental process itself may introduce, e.g., due to the need for participants to verbalize their thoughts. In this paper we discuss how AthenaLLM could enable researchers to go both broader (in terms of number of participants) and deeper (in terms of the kinds of research questions that can be tackled).

KEYWORDS

Experimentation, Large Language Models, Software Development, Empirical Software Engineering.

ACM Reference Format:
Benedito de Oliveira and Fernando Castor. 2024. AthenaLLM: Supporting Experiments with Large Language Models in Software Development. In *32nd IEEE/ACM International Conference on Program Comprehension (ICPC '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643916.3644437

[1]Extension available at https://github.com/nandooliveira/athena_llm_extension

1 INTRODUCTION

The advent of Large Language Models (LLMs) [23, 20, 21, 4, 7], such as GPT-4 [16], introduces a novel paradigm in software development. These models hold the promise of facilitating software development by generating large portions of source code automatically, based on prompts provided explicitly or implicitly by developers. Due to the potential of LLMs to significantly impact how software is built, recent studies [5, 11, 18, 3, 19, 12, 2, 22] have attempted to evaluate attributes such as the security [18], reliability [3], and readability [1] of the generated code, as well as the perceived performance of developers using them [24] and the perceived quality of the suggestions they make [14, 13].

To conduct these studies, researchers rely on a number of approaches. Some studies employ subjective assessments [2, 10, 22]. Finally, a few leverage custom-built tools that are not publicly available and are limited to a specific LLM [24].

Previous research has employed various strategies to evaluate the use of Large Language Models (LLMs) in software development. However, these approaches often encounter limitations, notably the challenge of scalability owing to the reliance on manual data collection and analysis. Additionally, they sometimes require the use of specific devices that demand the active involvement and presence of researchers, such as eye-tracking cameras or electrocardiogram (ECG) devices. These requirements can impose significant constraints on the feasibility and expansiveness of the research.

In this paper, we present a tool that enable researchers to thoroughly assess the utilization of resources such as LLMs in supporting software development. The tool, named AthenaLLM, is a Visual Studio Code plugin that seeks to address the limitations of previous data collection approaches by streamlining and automating the process of gathering data during the use of LLMs. First, it collects fine-grained data during the process of interacting with the LLM, e.g., suggestions provided by the model and which ones are accepted or rejected. Second, it also records the evolution of the code as it is modified both manually, by the developer, and as a consequence of suggestions made by the model. Third, it is compatible with LLMs that use an OpenAI-compatible API. As a consequence, it can be used with ChatGPT but also with models that can run locally, like Vicuna [6]. Fourth, albeit still under development, its current prototype is available as open source software to be used by the research community. AthenaLLM aims to facilitate more efficient evaluations by automating data collection and conducting preliminary data analysis, thereby reducing the manual effort required and enabling researchers to scale up studies on the effectiveness of LLMs and how developers use them.

2 RELATED WORK

LLMs, trained on public Internet data, may learn from low-quality code [3, 19, 18, 12], potentially leading to hard-to-detect bugs and additional developer effort. While proficient in common problems, they may struggle with less familiar cases, producing incomplete or irrelevant results.

Recent studies have investigated developers' use of LLM-based tools, focusing on the reliability and security of the generated code [3, 12, 19, 18]. These studies, however, often involve small cohorts and lack detailed data collection, mainly examining code generation but overlooking other potential uses like documentation and debugging. Our project aims to provide a comprehensive infrastructure for in-depth LLM research in software development.

Barke et al. [2] explored user interactions with GitHub Copilot, facing challenges in capturing real-time data. Similarly, Jiang et al. [10] studied GenLine, a natural language code synthesis tool, relying on transcribed video records for data analysis. Another study [22] also faced data analysis challenges, using audio and screen captures to evaluate code generation tool usability.

In contrast, Perry et al. [19] found that AI-assisted development might lead to less secure code. Their study divided participants into groups with and without AI tools, collecting data on code assistant interactions and final responses. Their approach, similar to ours, was conducted as a web-based exercise, whereas our methodology integrates directly with VSCode, enabling more precise tracking of developer modifications without screen recordings, thus offering deeper insights into developer behavior and potential issues.

3 PROPOSAL

Our objective is to enable researchers to conduct their studies on a larger scale by alleviating the burden of manual data collection. This approach is designed to facilitate more extensive research with a greater number of subjects, significantly enhancing the scope and impact of their work. In our initial approach, we aimed to develop an extension capable of interacting with Github Copilot, enabling researchers to collect data from its usage. However, we encountered a challenge in the form of security restrictions imposed by the VSCode extension API. These restrictions prevent one extension from listening to events generated by another extension, making it difficult to implement this functionality. Due to those restrictions, the tool we propose is an extension for VSCode, designed to mimic the functionalities of GitHub Copilot. Figure 1 presents an example of a completion suggested by the tool in a simple Python program.

AthenaLLM works by continuously monitoring the code in a source file and, based on the current location of the cursor. When the user stops typing for more than 2 seconds, it sends a sequence of tokens as a prompt to the configured LLM, obtains suggestions, and presents them to the user as possible completions. The way AthenaLLM selects the sequence of tokens to be sent as a prompt can be configured by the researcher conducting an experiment. This is further discussed later in this section. The proposed tool not only replicates key features of Github Copilot but also automates the collection of usage data. It tracks various metrics, such as the acceptance or rejection of suggestions by developers, and modifications made by developers to suggestions originally generated by the tool. All these actions trigger events within VSCode that AthenaLLM monitors. Figure 2 illustrates some events captured by AthenaLLM, including "didChange" for file modifications and "suggestionIgnored" when developers overlook suggested completions. It details the location of changes, modified code snippets, and file identifiers, showcasing the tool's ability to track and analyze developer interactions effectively.

The tool is configured to utilize OpenAI API models, such as GPT-4 and GPT-3.5, by default. This design feature empowers researchers with the flexibility to seamlessly switch between these models as per their research requirements. It ensures that the tool remains adaptable and relevant for a range of studies, catering to different research objectives and scenarios within the realm of software development using LLMs. Additionally, it offers compatibility with models that provide an OpenAI-compatible API. This flexibility allows researchers to conduct their studies without being restricted to proprietary models, thereby broadening the scope of their research and enabling a more diverse range of empirical studies in the realm of software development using LLMs.

The primary functionalities of the proposed tool encompass:

Automated Data Collection: This feature enables AthenaLLM to autonomously gather pertinent data during the developer's interaction with the tool. Data collection is triggered by various events, including:

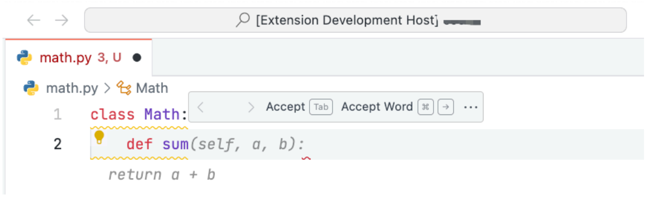
- Modifications made to the document.
- Acceptance of a suggestion by the developer.
- Ignoring of a suggestion by the developer.
- Requests for new suggestions from VSCode to OpenAI or alternative models.
- Receipt of suggestions from the model by VSCode including all the suggestions.
- Initiation and conclusion of an analysis session.

All collected data is stored in a MongoDB [15] database, which can be stored either locally or remotely using a REST API provided by this study, alongside the web application interface. This data is readily accessible to researchers, granting them the flexibility to conduct any analysis they deem necessary.

Depending on the event that is stored we can store different related data.

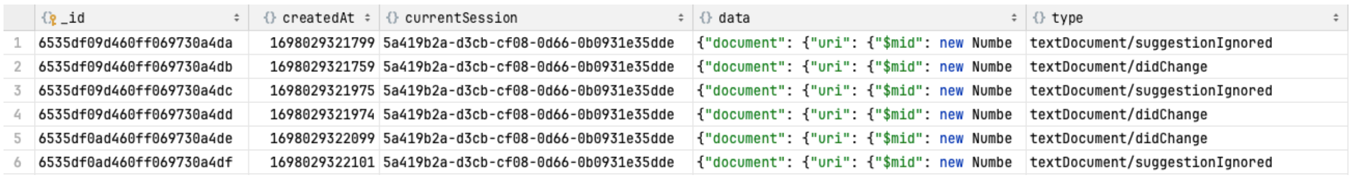
Preliminary Analysis Capability: A web-based tool is provided, which offers functionalities for initial data analysis, allowing for an early assessment of the collected information. In our future iterations, we intend to incorporate robust analysis capabilities.

Customization Options for Researchers: This feature offers flexibility for researchers to tailor the tool to their specific needs. Customization options include:



```
class Math:
    def sum(self, a, b):
        return a + b
```

Figure 1: Example of AthenaLLM showing code completion suggestions to developer.



	_id	createdAt	currentSession	data	type
1	6535df09d460ff069730a4da	1698029321799	5a419b2a-d3cb-cf08-0d66-0b0931e35dde	{"document": {"uri": {"$mid": new Numbe	textDocument/suggestionIgnored
2	6535df09d460ff069730a4db	1698029321759	5a419b2a-d3cb-cf08-0d66-0b0931e35dde	{"document": {"uri": {"$mid": new Numbe	textDocument/didChange
3	6535df09d460ff069730a4dc	1698029321975	5a419b2a-d3cb-cf08-0d66-0b0931e35dde	{"document": {"uri": {"$mid": new Numbe	textDocument/suggestionIgnored
4	6535df09d460ff069730a4dd	1698029321974	5a419b2a-d3cb-cf08-0d66-0b0931e35dde	{"document": {"uri": {"$mid": new Numbe	textDocument/didChange
5	6535df0ad460ff069730a4de	1698029322099	5a419b2a-d3cb-cf08-0d66-0b0931e35dde	{"document": {"uri": {"$mid": new Numbe	textDocument/didChange
6	6535df0ad460ff069730a4df	1698029322101	5a419b2a-d3cb-cf08-0d66-0b0931e35dde	{"document": {"uri": {"$mid": new Numbe	textDocument/suggestionIgnored

Figure 2: Example of events collected by AthenaLLM.

- Inform an API key to be used when making requests to the LLMs APIs.
- Selection from various models available on OpenAI or other models.
- Integration with other LLMs by providing a URL to an API, such as Vicuna [6], hosted on their own server.
- Maximum number of tokens allowed for suggestion responses.
- Model's temperature setting.
- Desired number of suggestions to be received from the model.
- Specified number of lines of code to be sent to the model for contextualizing the prompt.
- The server address where the collected data will be sent for the application.

These settings are crucial for formatting the request for suggestions and defining the prompt structure. They empower researchers to explore various aspects of the LLMs and conduct experiments with more flexibility.

4 USAGE

To utilize the extension, researchers must first install it within Visual Studio Code and configure the settings to suit their specific requirements. Following this, it is necessary to deploy and initialize the REST API application designed to receive and process the data collected by the extension. This REST API can be deployed on any machine that supports HTTP requests, including but not limited to Virtual Private Servers (VPS) offered by various cloud computing service providers.

4.1 Backend Setup

In order to optimize the installation and deployment procedures for both the REST API and the Web Interface, a detailed configuration for Docker [8] containers has been developed. This methodology guarantees a smooth and effective setup process. However, if Docker not align with specific requirements, the application is designed with the capability to be executed directly on a multitude of operating systems. The API is built upon the Flask framework [17], a web framework written in Python, renowned for its straightforward configuration and initialization processes. This inherent flexibility of Flask facilitates the deployment of the API across any Python-compatible environment, thereby offering a range of deployment options to the user.

The first step to setup the REST API is clone the git repository [2] as shown in Listing 1.

```
git clone git@github.com:nandooliveira/athena_llm_backend
    .git
```

Listing 1: Clone REST API repository

To execute the REST API and Web Interface applications using Docker, two primary prerequisites are required: Docker [8] and Docker Compose [9]. Once both Docker and Docker Compose are installed, researchers can initiate the entire necessary environment by executing the command `docker compose up` (see Listing 2). This process streamlines the setup, ensuring a consistent and reproducible environment across various systems.

```
cd /repository/path
docker compose up
```

Listing 2: Initialize the REST API and Web Interface

This process initiates all necessary application servers, including the REST API server, the Web Application server, and the MongoDB database in Docker containers. Once these components are operational, we can start utilization of the extension.

4.2 Extension Installation

To install the AthenaLLM extension, the initial step involves cloning the extension's code repository [3] as shown in Listing 3.

```
git clone git@github.com:nandooliveira/
    athena_llm_extension.git
```

Listing 3: Clone AthenaLLM extension code repository

Subsequently, to install the extension in VSCode, follow these steps:

(1) Launch VSCode and access the Extensions view by clicking on the Extensions icon located in the Activity Bar on the left side of the window, or use the shortcut `Ctrl+Shift+X`.
(2) In the Extensions pane, click on the `...` (More Actions) button at the top-right corner and select `Install from VSIX...`.
(3) Navigate to the directory containing the cloned AthenaLLM repository, locate the `.vsix` file, and select it for installation.

Upon selection, VSCode will proceed to install the AthenaLLM extension from the provided `.vsix` file.

4.3 Extension Setup

To modify the configuration settings of the AthenaLLM extension in Visual Studio Code, open the VSCode settings interface. The relevant configuration options are detailed in Listing 4. Adjust these settings as needed to customize the extension's behavior to suit your requirements.

[2]Backend repository https://github.com/nandooliveira/athena_llm_backend

[3]Extension repository https://github.com/nandooliveira/athena_llm_extension

```
{
    "Athena LLM.apiAddress": "https://api.openai.com/v1/
     chat/completions",
    "Athena LLM.apiKey": "***",
    "Athena LLM.contextSize": 3,
    "Athena LLM.enableTelemetry": true,
    "Athena LLM.maxTokens": 1000,
    "Athena LLM.model": "gpt-4",
    "Athena LLM.numberOfSuggestions": 3,
    "Athena LLM.serverAddress": "http://localhost:8000",
    "Athena LLM.temperature": 0.9
}
```

Listing 4: AthenaLLM customizable configurations

4.4 Experiment Setup

At this stage, the extension is fully operational, allowing for the commencement of the experiment. The developer just need to start using it and the extension will behave like Github copilot and collect the usage data. Currently, the initial version facilitates data segregation solely based on coding sessions. However, future iterations are planned to incorporate more sophisticated data aggregation methods, including categorization by individual experiments, specific tasks, and other relevant criteria.

To initiate a new session, we have integrated a command within Visual Studio Code as shown in Figure 3.

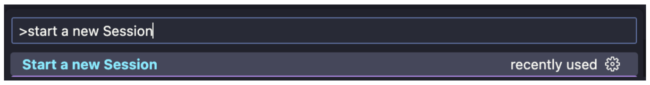
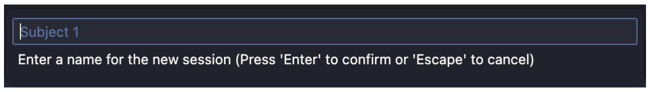


Figure 3: VSCode command to start a new coding session in AthenaLLM.

This feature allows researchers the option to assign a custom name to the session for easier identification, see Figure 4.



Enter a name for the new session (Press 'Enter' to confirm or 'Escape' to cancel)

Figure 4: VSCode command for naming a new coding session in AthenaLLM.

Alternatively, if a specific name is not provided, the extension will automatically generate a unique identifier using a random UUID (Universally Unique Identifier) as the session identifier. Currently, each session remains active until a new one is initiated.

4.5 Application Scenario

To illustrate AthenaLLM's use in research, consider its application in evaluating a new code-generating Large Language Model (LLM). This setup enables the detailed tracking of user interactions with the model, offering insights into its performance through acceptance and rejection metrics. As AthenaLLM supports multiple models, it facilitates comparative studies and offers quantitative (e.g., number of accepted and rejected suggestions) as well as qualitative (e.g., analysis of the characteristics of accepted and rejected suggestions) analysis tools. This aids in understanding user preferences and model effectiveness, offering a comprehensive assessment of LLM technologies in code generation. A visual workflow in Figure 5 showcases AthenaLLM's role from setup to analysis, highlighting its capabilities in facilitating research on automated code generation by benchmarking model performance and analyzing user interaction patterns.

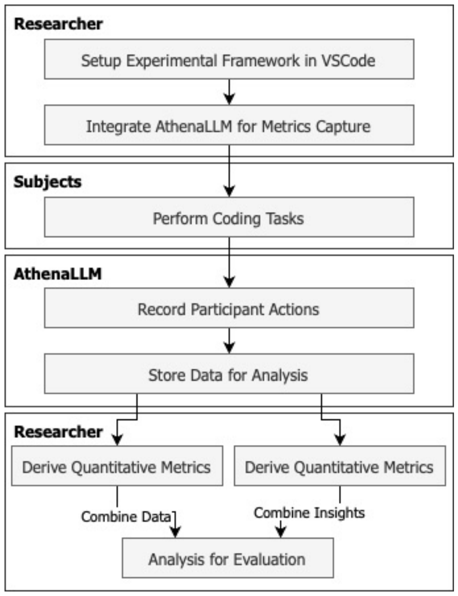

Researcher
Setup Experimental Framework in VSCode
Integrate AthenaLLM for Metrics Capture
Subjects
Perform Coding Tasks
AthenaLLM
Record Participant Actions
Store Data for Analysis
Researcher
Derive Quantitative Metrics
Derive Quantitative Metrics
Combine Data
Combine Insights
Analysis for Evaluation

Figure 5: Experiment Flow Using AthenaLLM

5 CONCLUSION

In conclusion, this research presents a significant advancement in the field of software engineering research, particularly in the study of Large Language Models (LLMs) in software development. The proposed VSCode extension addresses the critical challenges of scalability and manual data collection that have hindered previous research efforts. By automating data collection and providing preliminary analysis features, the tool significantly eases the research process, allowing for more extensive and in-depth studies with a larger participant base.

Furthermore, the tool's flexibility in integrating with various LLMs, including but not limited to OpenAI's GPT-4 and GPT-3.5, opens new avenues for comprehensive and versatile research. This adaptability is further enhanced by its compatibility with other models that provide an OpenAI-compatible API, broadening the scope of potential research applications.

The practical implications of this tool are vast. It not only streamlines the research process but also enriches the quality of insights obtained, contributing substantially to the understanding and advancement of LLMs in software development.

Acknowledgements. We would like to thank the anonymous reviewers who helped to improve this paper. This research was partially supported by INES 2.0 (FACEPE PRONEX APQ 0388-1.03/14 and APQ-0399-1.03/17, CNPq 465614/2014-0).

REFERENCES

[1] Naser Al Madi. 2022. How readable is model-generated code? examining readability and visual inspection of github copilot. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (ASE '22). ACM, (Oct. 2022). DOI: 10.1145/3551349.3560438.

[2] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2022. Grounded copilot: how programmers interact with code-generating models. (2022). arXiv: 2206.15000 [cs.HC].

[3] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking flight with copilot: early insights and opportunities of ai-powered pair-programming tools. *Queue*, 20, 6, (Jan. 2023), 35–57. DOI: 10.1145/3582083.

[4] Tom B Brown et al. 2020. Language models are few-shot learners.

[5] Mark Chen et al. 2021. Evaluating large language models trained on code. (2021). arXiv: 2107.03374 [cs.LG].

[6] Wei-Lin Chiang et al. 2023. Vicuna: an open-source chatbot impressing gpt-4 with 90%* chatgpt quality. (Mar. 2023). https://lmsys.org/blog/2023-03-30-vicuna/.

[7] 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)*. Association for Computational Linguistics, 4171–4186.

[8] Docker Inc. 2019. Docker: empowering app development for developers. Accessed: 2023-12-01. (2019). https://www.docker.com.

[9] Docker, Inc. 2023. Docker compose. https://docs.docker.com/compose/. Accessed: 2023-12-01. (2023).

[10] Aaron Michael Donsbach, Alejandra Molina, Carrie Jun Cai, Claire Kayacik, Edwin Toh, Ellen Jiang, Kristen Clare Olson, and Michael Terry. 2022. Discovering the syntax and strategies of natural language programming with generative language models. In.

[11] Zhangyin Feng et al. 2020. Codebert: a pre-trained model for programming and natural languages. (2020). arXiv: 2002.08155 [cs.CL].

[12] Kevin Jesse, Toufique Ahmed, Premkumar T. Devanbu, and Emily Morgan. 2023. Large language models and simple, stupid bugs. (2023). arXiv: 2303.11455 [cs.SE].

[13] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. (2023). arXiv: 2305.01210 [cs.SE].

[14] Zhijie Liu, Yutian Tang, Xiapu Luo, Yuming Zhou, and Liang Feng Zhang. 2023. No need to lift a finger anymore? assessing the quality of code generation by chatgpt. (2023). arXiv: 2308.04838 [cs.SE].

[15] MongoDB, Inc. 2023. Mongodb documentation. Accessed: 2023-12-01. (2023). https://www.mongodb.com/docs/.

[16] OpenAI. 2023. Gpt-4 technical report. (2023). arXiv: 2303.08774 [cs.CL].

[17] Pallets Projects. 2019. Flask documentation. Accessed: 2023-12-01. (2019). https://flask.palletsprojects.com/.

[18] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. Asleep at the keyboard? assessing the security of github copilot's code contributions. (2021). arXiv: 2108.09293 [cs.CR].

[19] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. 2022. Do users write more insecure code with ai assistants? (2022). arXiv: 2211.03622 [cs.CR].

[20] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

[21] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

[22] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs.nbsp;experience: evaluating the usability of code generation tools powered by large language models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (CHI EA '22) Article 332. Association for Computing Machinery, New Orleans, LA, USA, 7 pages. ISBN: 9781450391566. DOI: 10.1145/3491101.3519665.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. (2023). arXiv: 1706.03762 [cs.CL].

[24] Albert Ziegler, Eirini Kalliamvakou, Shawn Simister, Ganesh Sittampalam, Alice Li, Andrew Rice, Devon Rifkin, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. (2022). arXiv: 2205.06537 [cs.SE].