GitChameleon: Breaking the version barrier for code generation models

Anonymous

Reviewed on OpenReview: `https://openreview.net/forum?id=XXXX`

Figure 1: Conventional large language models tailored for code (LLM-$\mathcal{C}$) (Chameleon) are trained with a fixed cutoff date. However, as code environments are dynamic and subject to ongoing changes introduced by developers (young chameleons), these LLM-$\mathcal{C}$ models encounter challenges in adapting to such evolving scenarios. Images generated by DALL.E (Betker et al. (2023))

Abstract

The ever-changing landscape of programming languages poses a significant challenge in the development and training of models designed for code generation. Code, being a dynamic and constantly evolving environment, necessitates a continuous process of adaptation to stay in sync with the rapidly shifting paradigms, frameworks, and methodologies within the software development domain. The inherent variability in coding styles, the emergence of new programming languages, and the continuous evolution of libraries and packages underscore the imperative for an active approach in updating code generation models. In response to this challenge, we introduce **GitChameleon**, an innovative dataset comprising more than 12,000 version-sensitive examples in Python, designed to facilitate research into the adaptation of code generation models to the rapidly changing landscape of programming languages. Furthermore, we assess the performance of state-of-the-art code models and demonstrate their inadequacy in generating version-specific code. For example, the latest CodeLlama-70B (Rozière et al. (2024)) only achieves a 46.76% `exact_string_match` score when evaluated on **GitChameleon**.

Keywords: Large language models, code generation, lifelong learning, prompting, library-level understanding, version-sensitive code

1 Introduction

Large language models (LLMs) require substantial data for effective generalization, especially in coding contexts. The dynamic evolution of libraries complicates the collection of comprehensive data across diverse projects using them. Challenges escalate when libraries undergo significant version changes, resulting in LLMs generating code specific to one version, but not necessarily the current one. Companies dealing with these challenges may need to adhere to a particular library version, even if the model excels with a different one. Recognizing this, there is a push to establish benchmarks highlighting the constrained capabilities of large language models customized for code (LLM-$\mathcal{C}$), emphasizing both library-specific nuances and version-specific considerations.

Code is inherently dynamic, especially with the growing global community of developers and the rise of open-source software (OSS). GitHub reported a significant 27% year-over-year growth in project creation, with 4.5 billion contributions in 2023[1]. As shown in Fig. 2, prominent machine and deep learning libraries have experienced multiple version updates, reflecting a consistent upward trend in user downloads.

With the rise of coding and software development jobs, more individuals are turning to AI-assisted coding tools for learning and collaborative programming. A Stack Overflow survey[2] found that 70% of participants are either using or planning to integrate AI coding tools such as GitHub Copilot[3]. Among the respondents, 33% cited increased productivity as the main motivation for incorporating AI-based coding tools into their workflows. While various studies have explored the impact of AI-assisted coding in education (Becker et al. (2022); Kazemitabaar et al. (2023)) and the professional realm (Dakhel et al. (2023); Ziegler et al. (2022); Peng et al. (2023); Perry et al. (2023)), the consensus strongly supports the growing significance of integrating such tools into the coding workflow.

1. https://github.blog/2023-11-08-the-state-of-open-source-and-ai/
2. https://stackoverflow.co/labs/developer-sentiment-ai-ml/
3. https://github.com/features/copilot

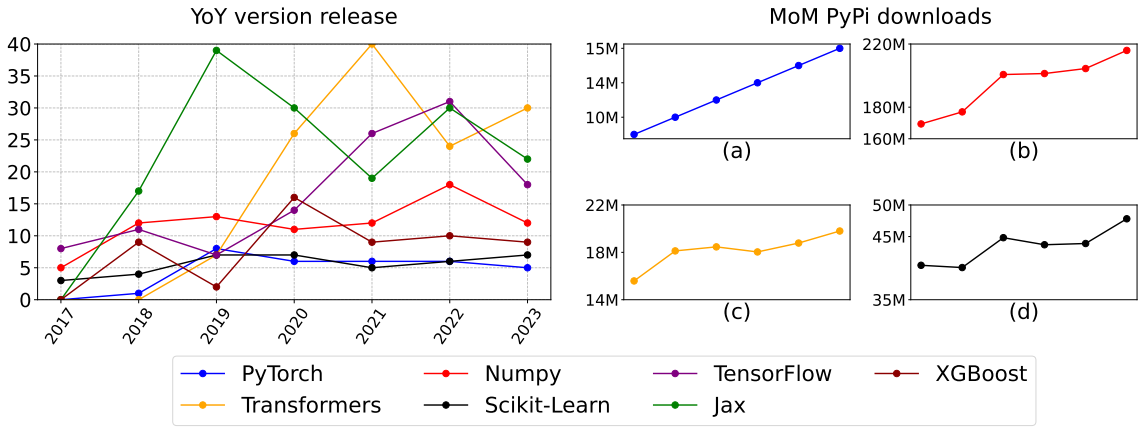


Figure 2: From left to right: (i) Year over year trends in number of version releases for popular machine learning/ deep learning stack libraries. (ii) Subplots (a), (b), (c), (d) represent the month over month PyPI downloads for PyTorch (Paszke et al. (2019)), Numpy (Harris et al. (2020)), Transformers (Wolf et al. (2020)) and Scikit-Learn (Buitinck et al. (2013)) respectively.

Due to the high demand, several recent LLM-$\mathcal{C}$ models have been proposed that include StarCoder (Li et al. (2023)), WizardCoder (Luo et al. (2023)), CodeGen v1 and v2 (Nijkamp et al. (2023b,a)), the CodeLLama family (Rozière et al. (2023)), and others. Further details on LLM-$\mathcal{C}$ can be found in Section 2.1. However, these models are typically trained with a static dataset that has a cut-off date, limiting the model's knowledge base. This limitation raises concerns about their ability to adapt to code changes, especially with the introduction of new library versions post their cut-off date.

Recognizing this significant limitation has been inadequately addressed and researched, primarily due to the inherent absence of evaluation protocols and benchmarks designed to highlight this aspect. In response, we have developed a new dataset and benchmark named **GitChameleon**. This benchmark comprises 12,584 samples of version-sensitive code, with the objective of assessing the capabilities of LLM-$\mathcal{C}$ to generate version-specific code and subsequently adapt to new versions or changes resulting from version upgrades.

Hence, our primary contributions can be succinctly summarized as follows.

1. We construct a novel and comprehensive Python-based dataset, named **GitChameleon**, specifically designed to be version-sensitive.

2. Our analysis across diverse LLM and LLM-$\mathcal{C}$ models provides initial evidence of their failure in generating version-specific code on our proposed dataset.

3. We empirically demonstrate the influence of model scaling, particularly within the CodeLLama family (Rozière et al. (2023)), in addressing our benchmark challenges.

4. In light of these findings, we propose a set of open research avenues utilizing our dataset **GitChameleon**, followed by a discussion on potential frameworks focused on fast adaptation.

Table 1: Popular Datasets for Code Generation

Dataset	Problems	Data Source	Library Specific	Version Specific
HumanEval (Chen et al. (2021))	164	Hand-Written	No	No
MBPP (Austin et al. (2021))	974	Hand-Written	No	No
MTPB (Nijkamp et al. (2022))	115	Hand-Written	No	No
APPS (Hendrycks et al. (2021))	10000	Competitions	No	No
CodeContests (Li et al. (2022))	117	Competitions	No	No
JulCe (Agashe et al. (2019))	1518049	Notebooks	No	No
DSP (Chandel et al. (2022))	1119	Notebooks	Yes	No
CoNaLa (Yin et al. (2018))	2879	StackOverflow	Yes	No
DS-1000 (Lai et al. (2023))	1000	StackOverflow	Yes	No
GitChameleon (Ours)	12584	Github	Yes	Yes

2 Related work

2.1 LLM-$\mathcal{C}$ training and evaluation protocols

LLM-$\mathcal{C}$ evaluations mainly revolve around code completion (Zhang et al. (2023a); van Dam et al. (2023); Lu et al. (2021)). Existing benchmarks emphasize generic code completion, yet a recognized limitation is the inability of LLM-$\mathcal{C}$ to comprehend and link library and project-level knowledge (Xu and Zhu, 2022), vital for real-world software applications.

Recent initiatives address repository-level code understanding by LLMs (Bairi et al. (2023); Shrivastava et al. (2023a,b); Liu et al. (2023); Guo et al. (2024)). Attempts at library-level code generation (Zan et al. (2022)) and consideration of dependencies between files (Guo et al. (2024)) have been made. However, these efforts do not directly address the challenge of accommodating library version-sensitive changes, adding complexity.

The core issue arises from models being trained on library code without explicit knowledge of library versions or their functional changes. Consequently, when tasked with generating code specifically compatible with a particular library version, these models often encounter failures.

2.2 Datasets

Existing datasets like HumanEval (Chen et al. (2021)), MBPP (Austin et al. (2021)), and MTPB (Nijkamp et al. (2022)) provide sets of handwritten prompts and test cases to evaluate code generated by LLM-$\mathcal{C}$. However, these datasets are relatively small and lack context regarding a model's comprehension of repositories. APPS (Hendrycks et al. (2021)) and CodeContest (Li et al. (2022)) offer challenging datasets with coding competition questions, providing insights into a model's performance on difficult problems but without a focus on library-specific challenges. DSP (Chandel et al. (2022)) and DS-1000 (Lai et al. (2023)) concentrate on the top data science libraries in Python, while JulCe (Agashe et al. (2019)) uses Jupyter Notebooks for training and evaluation, but these notebooks do not necessarily need to be repository-specific. CoNaLa (Yin et al. (2018)) contains problems collected from StackOverflow across multiple programming languages, including both library-specific questions and non-library-specific code. Although these datasets offer valuable resources for

training and evaluating models in various contexts, our contribution of the GitChameleon dataset opens up new avenues for research into version-conditioned generation by LLM-$\mathcal{C}$.

2.3 Implications for Lifelong Learning

Continual/lifelong learning in code generation models is in its early stages (Yadav et al., 2023; Weyssow et al., 2024; Wu et al., 2024; Gao et al., 2023). However, current efforts often focus on artificial sequential tasks rather than utilizing the natural distribution shift in the chronological evolution of code. Notably, continual learning mainly targets mitigating catastrophic forgetting and balancing forward- and backward-transfer on a data stream, which may not align optimally with coding environment demands.

In coding environments, obsolete or legacy libraries may prompt selective forgetting of irrelevant knowledge, particularly at the library/package level. Previous research, such as Caccia et al. (2021), serves as a foundation for developing continual learning in Large Language Models for code (LLM-$\mathcal{C}$). These approaches aim to prevent forgetting pre-training knowledge while adapting to new libraries and versions.

3 Dataset

3.1 Definitions

To ensure uniformity throughout the paper, our objective is to establish clear definitions for specific terminologies that are frequently used in reference to the proposed dataset and its corresponding construction framework.

1. **Project**: A project is characterized as any repository that utilizes components (classes, functions, helper code) from or is precisely a *dependent* of a library.

2. **Library**: A library is defined as any package containing reusable functions or classes that dependent projects invoke in their subsequent code. Libraries are assumed to be more commonly used and often undergo multiple version updates over time.

3. **Version Change**: A version change is described as any substantial modification marked in a new release for a specific library. For example, the transition from PyTorch (Paszke et al., 2019) 1.6 to 1.7 constitutes a specific version change for the PyTorch library.

3.2 Collection Framework

We curated a selection of the top 364 libraries[4], each having multiple versions/releases. Additionally, we compile a list of the top 700 projects based on their star count[5], each featuring only one release. Subsequently, we extracted the requirements for all these projects, with only 380 of them having accompanying `requirements.txt` files. Among these, 310 projects

4. Libraries were selected by filtering for those with a minimum of 1 GitHub star, extending up to a maximum of 16102 stars.
5. Projects were selected by filtering for those with a minimum of 5 GitHub stars, extending up to a maximum of 2500 stars.

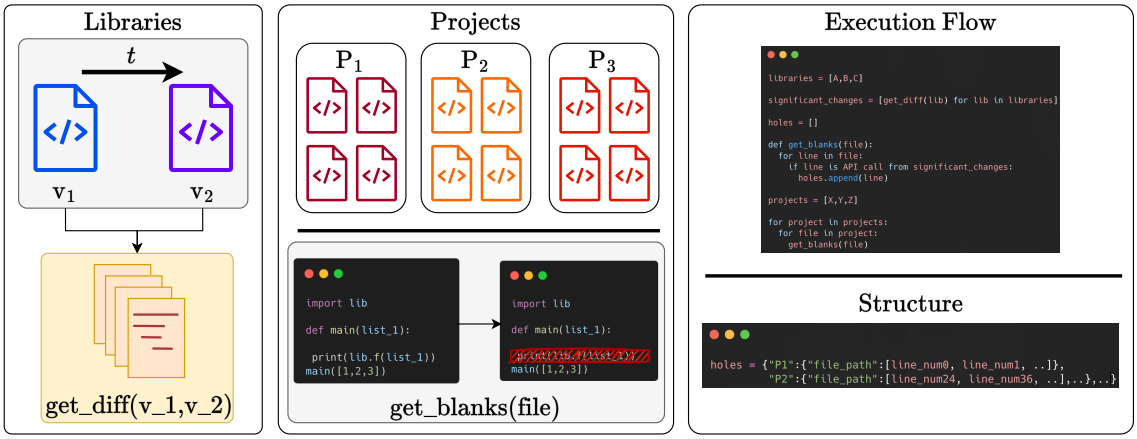


Figure 3: **GitChameleon construction framework**: Initiated with a collection of libraries, we identify version differences for each library using `get_diff()`. Following this, we gather a set of projects employing the specified libraries and identify lines in each project file where a version-sensitive function from the package is invoked, utilizing `get_blanks()`. Subsequently, we annotate these lines for later use in the version-specific code completion task.

had a requirement that matched our set of libraries. We proceeded to use the versions and checked for the existence of a GitHub release that corresponded to the version specified in the `requirements.txt` file from the projects. By doing so, we obtained that version and a version preceding it, resulting in a total of 47 libraries.

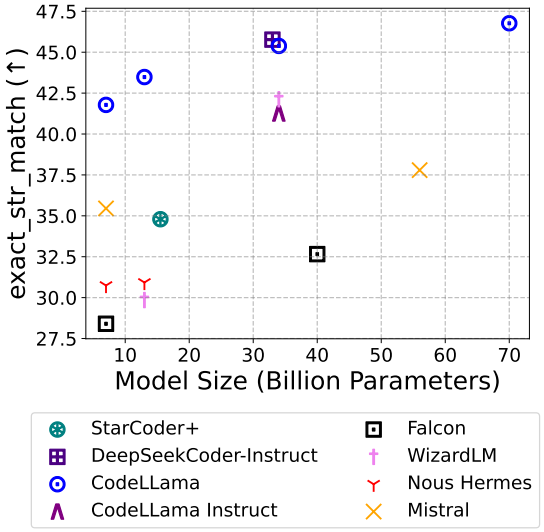
Subsequently, we identified the projects that had a requirement from the set of 47 libraries and proceeded to download these projects. Our download process successfully retrieved 32 projects.

Filtering: We filter out projects older than 2023-04-01[6]. We filter out projects that have more than one version. We filter out libraries which have only one version. We use the `requirements.txt` file from projects and keep only the projects that use a specific version from a library. We then use the set of versions for each library and download all versions for each library.

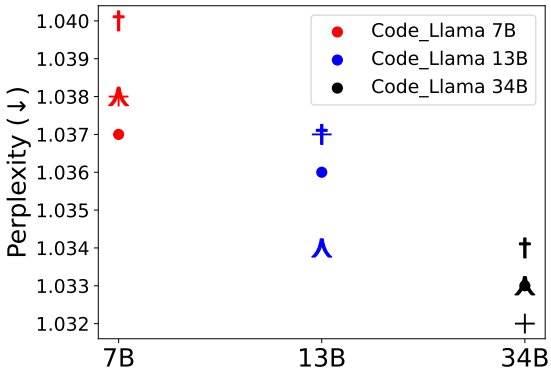
`get_diff(v_1,v_2)`**:** To identify version-specific method invocations in projects, we employ an AST parser to extract all class and method names from each file. Our analysis compares two versions of a library, identifying significant changes based on criteria that may introduce breaking changes for users. These include the presence or absence of function or class names, discrepancies in non-optional arguments, and changes in the order of arguments for functions. For classes with the same name in both versions, differences in attributes are considered. This approach aims to pinpoint modifications that affect the compatibility or functionality of the library.

`get_blanks(file)`**:** To pinpoint sections requiring updates between versions, we systematically go through the files of each library. Using `get_diff` for files present in both

6. This date corresponds to the training cut-off date for the StarCoder model (Li et al., 2023)



(a) `exact_str_match` results on different scales of LLM and LLM-$\mathcal{C}$ families. For Mistral (Jiang et al. (2023)), the 56 billion parameter variant on the plot represents the Mixtral 8x7B model (Jiang et al. (2024)).



(b) **Scaling behaviour**: Impact of increasing the model capacity of the Code-LLama family on **perplexity**. ·,+,⅄,† represent perplexities when prompted via templates 1,2,3 and 4 respectively as described in Table 2.

Figure 4: **GitChameleon evaluation benchmark**

versions, we identify spots necessitating modifications due to library updates. Subsequently, we scrutinize each line of the project files to identify calls to library functions that have undergone substantial changes between versions, marking these areas as blanks. This automated approach aids in identifying lines in projects that may lead to compatibility issues.

We provide further details on the dataset structure in the Appendix A.

4 Evaluation

4.1 Task and prompts

In this study, we delve into the task of version-specific code completion. Specifically, for lines where version-sensitive code was identified through `get_blanks()`, we mask those lines and instruct the LLM/LLM-$\mathcal{C}$ to generate the version-specific code using the prompt templates outlined in Table 2.

template 1	"# Using library-version: {}"
template 2	"# Given the following code uses library-version: {}"
template 3	"# Finalize the provided code using library-version: {}"
template 4	"# Strictly use library-version: {} to complete the following code"

Table 2: Prompt templates for version specific code completion.

We discuss other potential tasks on GitChameleon in the Appendix B.

4.2 Results

In Fig. 4a, we investigate various LLM and LLM-$\mathcal{C}$ models, including WizardLM (13B, 34B) (Luo et al., 2023), DeepSeekCoder (33B Instruct) (Guo et al., 2024), Nous Hermes (7B, 13B) (Teknium, 2023), Mistral-7B (Jiang et al., 2023), Mixtral 8x7B (Jiang et al., 2024), CodeLLama (7B, 13B, 34B, 70B, Instruct 34B) (Rozière et al., 2023, 2024), Falcon (7B, 40B) (Almazrouei et al., 2023), and StarCoder+[7] (15.5 B) (Li et al., 2023). Our assessment focused on the version-specific code completion task outlined in Sec. 4.1. The summary of our findings is as follows:

1. **Scaling Trends.** The results presented in Fig. 4a and 4b indicate a consistent scaling behavior. As observed, models with higher parametric capacity tend to exhibit elevated `exact_string_match` scores and reduced perplexity, both within the same model class and across different model families. For example, Falcon 7B achieves an approximate `exact_string_match` score of 29.85%, while its larger counterpart, Falcon 40B, achieves around 42.1% `exact_string_match` score. Regarding the perplexity analysis, as highlighted in Fig. 4b, our focus is specifically on the CodeLLama model family. Across all prompt templates, perplexity consistently decreases with increasing parametric capacity of the model.

2. **Inability to perform the task.** It is noteworthy that even the most advanced state-of-the-art (SoTA) LLM-$\mathcal{C}$ models struggle to accomplish the task. For example, the recent CodeLlama-70B (Rozière et al. (2024)) achieves only a 46.76% `exact_string_match` when assessed on the GitChameleon dataset.

Note: In Fig. 4a, we report the score for each model based on the best `exact_string_match` score obtained across all used prompt templates. Details on the metrics used are provided in the Appendix D.

5 Conclusion

Recognizing the crucial need for LLM-$\mathcal{C}$ adaptation to evolving code environments, particularly in widely used libraries, we introduce a novel and extensive Python-based version-specific benchmark named **GitChameleon**. By effectively leveraging GitChameleon, we expose the shortcomings of existing state-of-the-art (SoTA) LLM and LLM-$\mathcal{C}$ models in producing version-specific code, representing an inaugural effort to draw attention to this challenge. While our work exposes this shortcoming, we acknowledge the dataset's limitations, such as the filtered number of projects and the confinement to the Python language. In future endeavors, we aim to enhance the dataset's comprehensiveness across various programming languages and frameworks. Additionally, we plan to introduce new tasks that can benefit research on LLM-$\mathcal{C}$ models using GitChameleon.

Finally, in Appendices B, and C, we outline potential implications of this benchmark, present research questions, identify open problems, and suggest different avenues of solutions that can support foundational LLM-$\mathcal{C}$ models in addressing this task.

7. StarCoder+ model card: `https://huggingface.co/bigcode/starcoderplus`

Broader Impact Statement

With GitChameleon, researchers and developers gain a valuable resource to improve LLM-$\mathcal{C}$ models, aligning them more closely with the dynamic landscape of programming languages, frameworks, and libraries. This potential improvement could significantly boost developer efficiency and mitigate version-related errors in AI-assisted code generation. Industries often face challenges in updating dependencies or crafting version-specific code to address vulnerabilities arising from older versions. By investigating how models generate version-specific code, we anticipate advancements in addressing these issues. The project's open-access approach, providing both dataset and task to the public, fosters collaborative innovation, enabling the wider research community to contribute to the evolution of code generation technologies. This inclusive strategy encourages cooperation, accelerates progress, and democratizes access to cutting-edge tools.

While our dataset represents a crucial initial step in evaluating LLM-$\mathcal{C}$ model capabilities for version-specific code generation, we acknowledge its limitations. The dataset focuses exclusively on Python code and is restricted to public projects, limiting the scope of sectors for model evaluation.

References

R. Agashe, S. Iyer, and L. Zettlemoyer. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *arXiv preprint arXiv:1910.02216*, 2019.

E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo. The falcon series of open language models, 2023.

J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

R. Bairi, A. Sonwane, A. Kanade, V. D. C, A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, and S. Shet. CodePlan: Repository-level Coding using LLMs and Planning, Sept. 2023. URL `http://arxiv.org/abs/2309.12499`. arXiv:2309.12499 [cs].

B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos. Programming is hard – or at least it used to be: Educational opportunities and challenges of ai code generation, 2022.

J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2(3):8, 2023.

L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

M. Caccia, P. Rodriguez, O. Ostapenko, F. Normandin, M. Lin, L. Caccia, I. Laradji, I. Rish, A. Lacoste, D. Vazquez, and L. Charlin. Online Fast Adaptation and Knowledge Accumulation: a New Approach to Continual Learning, Jan. 2021. URL http://arxiv.org/abs/2003.05856. arXiv:2003.05856 [cs].

J. Cao, M. Li, X. Chen, M. Wen, Y. Tian, B. Wu, and S.-C. Cheung. Deepfd: automated fault diagnosis and localization for deep learning programs. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22. ACM, May 2022. doi: 10.1145/3510003.3510099. URL http://dx.doi.org/10.1145/3510003.3510099.

S. Chandel, C. B. Clement, G. Serrato, and N. Sundaresan. Training and evaluating a jupyter notebook data science assistant. *arXiv preprint arXiv:2201.12901*, 2022.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. 2021.

M. Chen, H. Zhang, C. Wan, Z. Wei, Y. Xu, J. Wang, and X. Gu. On the Effectiveness of Large Language Models in Domain-Specific Code Generation, Dec. 2023. URL http://arxiv.org/abs/2312.01639. arXiv:2312.01639 [cs].

A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. Ming, and Jiang. Github copilot ai pair programmer: Asset or liability?, 2023.

D. Emelin, D. Bonadiman, S. Alqahtani, Y. Zhang, and S. Mansour. Injecting Domain Knowledge in Language Models for Task-Oriented Dialogue Systems, Dec. 2022. URL http://arxiv.org/abs/2212.08120. arXiv:2212.08120 [cs].

S. Gao, H. Zhang, C. Gao, and C. Wang. Keeping pace with ever-increasing data: Towards continual learning of code intelligence models, 2023.

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence, Jan. 2024. URL http://arxiv.org/abs/2401.14196. arXiv:2401.14196 [cs].

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mixtral of experts, 2024.

M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. ACM, Apr. 2023. doi: 10.1145/3544548.3580919. URL `http://dx.doi.org/10.1145/3544548.3580919`.

Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR, 2023.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. Starcoder: may the source be with you!, 2023.

Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. Sutherland Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, Dec. 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL `http://dx.doi.org/10.1126/science.abq1158`.

T. Liu, C. Xu, and J. McAuley. RepoBench: Benchmarking Repository-Level Code Auto-Completion Systems, Oct. 2023. URL `http://arxiv.org/abs/2306.03091`. arXiv:2306.03091 [cs].

S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation, 2021.

Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang. Wizardcoder: Empowering code large language models with evol-instruct, 2023.

E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong. A conversational paradigm for program synthesis. *arXiv preprint*, 2022.

E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou. Codegen2: Lessons for training llms on programming and natural languages. *arXiv preprint arXiv:2305.02309*, 2023a.

E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong. Codegen: An open large language model for code with multi-turn program synthesis, 2023b.

A. Nikanjam, H. B. Braiek, M. M. Morovati, and F. Khomh. Automatic fault detection for deep learning programs using graph transformations, 2021.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer. The impact of ai on developer productivity: Evidence from github copilot, 2023.

N. Perry, M. Srivastava, D. Kumar, and D. Boneh. Do users write more insecure code with ai assistants? In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23. ACM, Nov. 2023. doi: 10.1145/3576915.3623157. URL `http://dx.doi.org/10.1145/3576915.3623157`.

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code Llama: Open Foundation Models for Code, Aug. 2023. URL `http://arxiv.org/abs/2308.12950`. arXiv:2308.12950 [cs].

B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. C. Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code, 2024.

D. Shrivastava, D. Kocetkov, H. de Vries, D. Bahdanau, and T. Scholak. RepoFusion: Training Code Models to Understand Your Repository, June 2023a. URL `http://arxiv.org/abs/2306.10998`. arXiv:2306.10998 [cs].

D. Shrivastava, H. Larochelle, and D. Tarlow. Repository-Level Prompt Generation for Large Language Models of Code, June 2023b. URL http://arxiv.org/abs/2206.12839. arXiv:2206.12839 [cs].

Teknium. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants, 2023. URL https://huggingface.co/datasets/teknium/OpenHermes-2.5.

T. van Dam, M. Izadi, and A. van Deursen. Enriching source code with contextual data for code completion models: An empirical study, 2023.

M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. Sahraoui. Exploring parameter-efficient fine-tuning techniques for code generation with large language models, 2024.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

T. Wu, L. Luo, Y.-F. Li, S. Pan, T.-T. Vu, and G. Haffari. Continual learning for large language models: A survey, 2024.

Y. Xu and Y. Zhu. A Survey on Pretrained Language Models for Neural Code Intelligence, Dec. 2022. URL http://arxiv.org/abs/2212.10079. arXiv:2212.10079 [cs].

P. Yadav, Q. Sun, H. Ding, X. Li, D. Zhang, M. Tan, X. Ma, P. Bhatia, R. Nallapati, M. K. Ramanathan, M. Bansal, and B. Xiang. Exploring Continual Learning for Code Generation Models, July 2023. URL http://arxiv.org/abs/2307.02435. arXiv:2307.02435 [cs].

P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig. Learning to mine aligned code and natural language pairs from stack overflow. In *Proceedings of the 15th international conference on mining software repositories*, pages 476–486, 2018.

D. Zan, B. Chen, D. Yang, Z. Lin, M. Kim, B. Guan, Y. Wang, W. Chen, and J.-G. Lou. CERT: Continual Pre-Training on Sketches for Library-Oriented Code Generation, June 2022. URL http://arxiv.org/abs/2206.06888. arXiv:2206.06888 [cs].

F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation, Oct. 2023a. URL http://arxiv.org/abs/2303.12570. arXiv:2303.12570 [cs].

K. Zhang, H. Zhang, G. Li, J. Li, Z. Li, and Z. Jin. ToolCoder: Teach Code Generation Models to use API search tools, Sept. 2023b. URL http://arxiv.org/abs/2305.04032. arXiv:2305.04032 [cs].

Z. Zhang, Z. Zeng, Y. Lin, H. Wang, D. Ye, C. Xiao, X. Han, Z. Liu, P. Li, M. Sun, and J. Zhou. Plug-and-Play Knowledge Injection for Pre-trained Language Models, Dec. 2023c. URL http://arxiv.org/abs/2305.17691. arXiv:2305.17691 [cs].

S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang, and G. Neubig. DocPrompting: Generating Code by Retrieving the Docs, Feb. 2023. URL http://arxiv.org/abs/2207.05987. arXiv:2207.05987 [cs].

A. Ziegler, E. Kalliamvakou, S. Simister, G. Sittampalam, A. Li, A. Rice, D. Rifkin, and E. Aftandilian. Productivity assessment of neural code completion, 2022.

Appendix A. GitChameleon structure and statistics

Following the filtration process outlined in Sec. 3.2, we now have 47 distinct libraries and 32 individual projects. The complete list of all projects and libraries part of the GitChameleon dataset is provided in Sec E. *nightly* and *pre-release* versions while are used to differentiate between libraries and projects, we only compute `get_diff()` on the main releases.

```
blanks_lines_by_library ->
"jinja": {
       "Projects/THUDM/WebGLM/model/retriever/extracting/html2text.py": [ -> this
           is the file
           [
               [
                   "jinja-3.1.2",
                   "jinja-3.0.2"
               ],
               849,
               "escape"
           ],
           [
               [
                   "jinja-3.1.2",
                   "jinja-3.0.2"
                    |
                    -> these are the versions we compared for getting the
                        difference
               ],
               849, -> this is the line
               "escape" -> this is the function
           ]
       ]
```

We introduce a dataset designed to assess the effectiveness of an LLM in generating version-specific code. The dataset has a distinctive structure that centers on variations in function calls across different versions of widely used libraries. To exemplify its organization, we present a section of the dataset called `blanks_lines_by_library`.

The dataset is organized as a nested dictionary, with the outer layer keyed by library names. For example, under the "jinja" key, there is an associative array mapping file paths to specific instances of version-dependent code. Each file path leads to an array of items, each representing a unique instance where a function call differs between versions of the library.

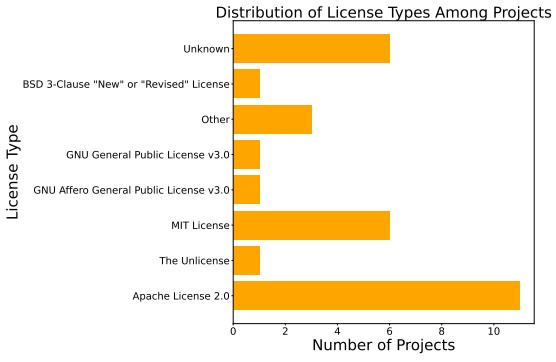


Figure 5: License distributions of the projects contained within the GitChameleon dataset.

An item in this array is a triple, where the first element consists of version identifiers (e.g., "jinja-3.1.2" and "jinja-3.0.2"), denoting the specific releases compared to identify functional differences. The second element is an integer that indicates the line number in the file where the difference occurs. Finally, the third element is a string naming the function involved in the version-specific change.

This structured approach facilitates a thorough examination of how functions evolve across versions within a library. The dataset's analysis of these version-specific changes provides a valuable resource for evaluating the LLM's capability to generate precise and version-aware code.

Regarding the statistics of the data set, all the projects used in crafting GitChameleon are publicly accessible and predominantly carry open-source licenses, with the Apache License 2.0 and the MIT License being the primary licenses, as illustrated in Fig. 5. The complete dataset contains a total of *3363723* lines of code.

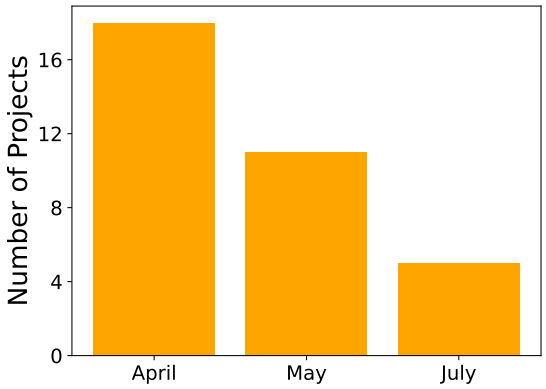


Figure 6: Repository creation date in 2023 of the projects scraped as part of the GitChameleon dataset.

As shown in Fig. 6, the majority of projects included in the current version of GitChameleon, as presented in this study, were created in the months of April, May, and June 2023. We recognize this as a limitation of our current dataset construction process and, as a result, plan to enhance it in the next iteration of the benchmark.

One crucial criterion for filtering projects involved adhering to the StarCoder (Li et al., 2023) cut-off date of 2023-04-01. This measure aimed to prevent exposure of StarCoder to the projects included in the dataset during its training, mitigating the risk of data leakage. However, it is important to note that this safeguard cannot be guaranteed for the other models examined in this paper. Initially, it is essential to acknowledge that some of the models assessed in this study are closed-source or lack detailed information about their training data, introducing a potential risk of data leakage for those models. In particular, the CodeLLama model (Rozière et al., 2023, 2024) has a cut-off date of September 2022, ensuring that the best-performing open-source model on our benchmark avoids data leakage.

A prospective avenue for exploration involves examining the performance of each model on versions within and outside their respective training cut-off dates. Assessing versions outside the training cut-off window serves as a proxy for out-of-distribution generalization, presenting an intriguing challenge for further investigation of state-of-the-art LLM-$\mathcal{C}$ models.

Appendix B. Future work

As discussed in previous sections, our ongoing efforts to enhance the GitChameleon benchmark and introduce novel tasks involve exploring the following key areas:

1. **More projects:** While GitChameleon offers a substantial dataset, we recognize the need for greater comprehensiveness. Expanding to include projects created beyond

April, May, and June 2023 is vital. This expansion aims to cover repositories associated with new frameworks, diverse research areas, and the latest package versions.

2. **Exploration of software stacks:** The current library set explicitly focuses on the machine / deep learning software stack. While serving as a strong baseline, it is essential to assess how observations transfer to other software stacks. Therefore, expanding to encompass other software stacks is a critical future exploration.

3. **Analysing performance based on model cut-off date:** As outlined in Section A, we plan to analyze model performance on partitions of the GitChameleon dataset conditioned on their training cut-off date.

4. **Expanding to other programming languages:** While Python is a popular and well-documented language for the machine/deep learning stack, our aim is to extend our dataset to include other languages such as C++, Rust, and Java.

5. **Exploring new tasks on GitChameleon:** Our work, initially focused on code completion, has limitations. To address this, we intend to propose novel tasks that leverage the dataset's structure to discover fundamental properties of the LLM-$\mathcal{C}$ models. One potential task is inspired by automated fault diagnosis (Cao et al., 2022; Nikanjam et al., 2021), which involves the model in identifying or annotating code lines in a project file that could be version sensitive.

6. **Synthetic test set generation:** All experiments in this paper relied on zero-shot prompting for inference with the LLM and LLM-$\mathcal{C}$ models. However, GitChameleon can be employed for the rapid adaptation of the LLM and LLM-$\mathcal{C}$ models. Consequently, a test set is needed to evaluate the effectiveness of GitChameleon when integrated into any training framework.

Appendix C. Potential Solutions to Library-version specific adaptation

RAG+LLM-$\mathcal{C}$. Retrieval Augmented Generation (RAG) is an approach that incorporates a retriever model with access to non-parametric memory, such as the entire Wikipedia index Lewis et al. (2020). The use of RAG has recently been explored as a solution to generate code at the repository level Zhou et al. (2023); Zhang et al. (2023b). Consequently, there is an intuitive direction toward integrating LLM-$\mathcal{C}$ models with RAG. In this setup, the RAG's knowledge base is continuously updated with new version releases of libraries. When presented with a version-specific code generation task, the RAG can retrieve the relevant code or references from the library documentation corresponding to that specific version.

Knowledge injection: Another recent strategy to integrate new information into an LLM is known as knowledge injection (Zhang et al., 2023c; Emelin et al., 2022). Previous research has investigated the efficacy of knowledge injection specifically for library version-specific code (Chen et al., 2023). Although this method can be a viable means of continually updating LLM-$\mathcal{C}$, it is essential to note that these approaches may incur significant computational expenses.

Appendix D. Metrics

The results presented in Sec 4.2 utilized two distinct metrics for evaluation: (i) `exact_str_match`, and, (ii) Perplexity.

`exact_str_match` simply computes whether the generated output from the LLM/ LLM-$\mathcal{C}$ model is equal to the target/ ground-truth. Higher `exact_str_match` values indicate better model performance.

Perplexity can be mathematically defined as:

$$\text{Perplexity} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_1,w_2,...,w_{i-1})} \tag{1}$$

- N is the total number of words in the sequence.
- w_i is the actual word at position i.
- $P(w_i|w_1, w_2, ..., w_{i-1})$ is the probability assigned by the language model to the word w_i given the preceding context $w_1, w_2, ..., w_{i-1}$
- The sum runs over all words in the sequence.

This formula calculates the average perplexity over the entire sequence of words, measuring how well the language model predicts the next word in a sequence. Lower perplexity values indicate better model performance.

Appendix E. List of projects and libraries in GitChameleon

Table 3: Projects

peterw/Chat-with-Github-Repo	muellerberndt/mini-agi
facebookresearch/llama-recipes	sail-sg/EditAnything
SCUTlihaoyu/open-chat-video-editor	Lightning-AI/lit-gpt
yanqiangmiffy/Chinese-LangChain	IntelligenzaArtificiale/Free-Auto-GPT
baichuan-inc/Baichuan-13B	VideoCrafter/VideoCrafter
thomas-yanxin/LangChain-ChatGLM-Webui	Jittor/JittorLLMs
TigerResearch/TigerBot	jxxghp/MoviePilot
yangjianxin1/Firefly	turboderp/exllama
THUDM/CodeGeeX2	agiresearch/OpenAGI
ttengwang/Caption-Anything	NotJoeMartinez/yt-fts
X-PLUG/mPLUG-Owl	Docta-ai/docta
liltom-eth/llama2-webui	AlanChen4/Summer-2024-SWE-Internships
linyiLYi/snake-ai	liucongg/ChatGLM-Finetuning
open-mmlab/Multimodal-GPT	ZrrSkywalker/Personalize-SAM
lyuchenyang/Macaw-LLM	wenge-research/YaYi
mishalhossin/Discord-AI-Chatbot	a16z-infra/llama2-chatbot
peterw/Chat-with-Github-Repo	

Table 4: Libraries

openai/openai-python	pandas-dev/pandas	scikit-learn/scikit-learn
theskumar/python-dotenv	mozillazg/python-pinyin	psf/black
pytorch/pytorch	psf/requests	huggingface/datasets
google/python-fire	Textualize/rich	albumentations-team/albumentations
huggingface/transformers	pytorch/vision	Lightning-AI/lightning
numpy/numpy	tqdm/tqdm	streamlit/streamlit
gradio-app/gradio	microsoft/DeepSpeed	pallets/click
pallets/jinja	DLR-RM/stable-baselines3	tiangolo/fastapi
huggingface/diffusers	openai/gym	pallets/flask
fxsjy/jieba	jiaaro/pydub	huggingface/pytorch-image-models
encode/starlette	langchain-ai/langchain	joblib/joblib
matplotlib/matplotlib	pydantic/pydantic	librosa/librosa
PaddlePaddle/PaddleNLP	openai/tiktoken	Delgan/loguru
		python-jsonschema/jsonschema

Appendix F. ML reproducibility checklist

1. For all authors...
   a. Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   b. Did you describe the limitations of your work? [Yes]
   c. Did you discuss any potential negative societal impacts of your work? [N/A]
   d. Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   a. Did you state the full set of assumptions of all theoretical results? [N/A]
   b. Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments (e.g. for benchmarks)...
   a. Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The dataset GitChameleon proposed in the paper is publicly available at `https://shorturl.at/dvJ45` and will be de-anonymized and hosted on HuggingFace Hub after acceptance of the paper. All models benchmarked in the paper are accessible on `https://together.xyz/`.
   b. Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   c. Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   d. Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Considering the paper provides only inference-based zero-shot results, only CPU-based compute resources were utilized for the project. Acknowledgments of the computational resources utilized shall be disclosed upon the acceptance of the paper. For the evaluation of the models, we used the Inference Engine of `https://together.xyz/` which resulted in a total amount of USD$ 46.6.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   a. If your work uses existing assets, did you cite the creators? [Yes]
   b. Did you mention the license of the assets? [Yes]
   c. Did you include any new assets either in the supplemental material or as a URL? [Yes]
   d. Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

e. Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   a. Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   b. Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   c. Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix G. Datasheet

Motivation

For what purpose was the dataset created? Was there a specific task in mind? Was there a specific gap that needed to be filled? Please provide a description.

GitChameleon was created to demonstrate the current SoTA LLM and LLM for code models' shortcomings in generating version-specific code.

Who created this dataset (e.g., which team, research group) and on behalf of which entity (e.g., company, institution, organization)?

Who funded the creation of the dataset? If there is an associated grant, please provide the name of the grantor and the grant name and number.

No explicit funding was utilized in the creation of the GitChameleon dataset. Acknowledgment of the computational resources utilized shall be disclosed upon the acceptance of the paper.

Any other comments?

Composition

What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)? Are there multiple types of instances (e.g., movies, users, and ratings; people and interactions between them; nodes and edges)? Please provide a description.

The dataset comprises publicly available Python code repositories sourced from GitHub. It includes libraries with multiple versions and associated projects that utilize at least one version from those libraries.

How many instances are there in total (of each type, if appropriate)?

There is a total of 47 unique libraries (with multiple versions) and 32 unique projects.

Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

The dataset is a subset of the most widely used libraries and projects. Popularity is determined by the number of stars each repository receives. We selected the top X libraries and the top Y projects on the basis of their star count. Libraries had to have multiple versions, while projects were limited to a single version. Additionally, the projects were required to use at least one specific version of a library, confirmed by inspection of the `requirements.txt` file in the library's repository.

What data does each instance consist of? "Raw" data (e.g., unprocessed text or images) or features? In either case, please provide a description.

Each repository's data comprises the cloned repository itself, containing code files, text, and other relevant materials associated with the repository.

Is there a label or target associated with each instance? If so, please provide a description.

Each repository has a specific name as defined in the repository name on GitHub.

Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text.

There was no removal of data from each repository; we solely downloaded the versions specifically utilized by the projects.

Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit.

The connection between libraries and projects is established by examining the `requirements.txt` file in each project. Each library is represented by a folder and within it there are subfolders containing cloned repositories for each distinct version.

Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

This dataset exclusively serves as testing data, specifically designed for zero-shot inference, and thus, it does not include any splits.

Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description.

The data is in its raw form, and there may be redundancies since no deduplication preprocessing has been applied.

Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate.

The repositories are downloaded and zipped. The libraries are guaranteed to remain constant, since they were downloaded for specific versions. However, projects may change. Licensing varies for each repository, as illustrated in Fig. 5.

Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctor-patient confidentiality, data that includes the content of individuals non-public communications)? If so, please provide a description.

The data was gathered from public repositories, ensuring that there is no confidential information involved.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why.

While we did not scrutinize the contents of the repositories, their popularity implies that they likely contain valuable code, and we anticipate that they do not include any content that could be considered offensive, insulting, threatening, or otherwise anxiety-inducing.

Does the dataset relate to people? If not, you may skip the remaining questions in this section.

No.

Does the dataset identify any subpopulations (e.g., by age, gender)? If so, please describe how these subpopulations

are identified and provide a description of their respective distributions within the dataset.

Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset? If so, please describe how.

Does the dataset contain data that might be considered sensitive in any way (e.g., data that reveals racial or ethnic origins, sexual orientations, religious beliefs, political opinions or union memberships, or locations; financial or health data; biometric or genetic data; forms of government identification, such as social security numbers; criminal history)? If so, please provide a description.

Any other comments?

Collection Process

How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how.

The data is directly accessible since it comprises public GitHub repositories.

What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated?

Projects were filtered to have only one release version, while libraries were required to have multiple versions. The selection ensured that all the projects utilized a specific version of a library.

If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)?

We collected the top 700 projects with stars ranging from 5 to 2500 on GitHub, filtering based on their requirements and release count. Additionally, we gathered the top 364 libraries with stars ranging from 1 to 16102, ensuring that they had multiple release versions.

Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)?

The data collection process was carried out exclusively by individuals involved in this project, with no external contributors.

Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created.

Data collection and download occurred from July 31st, 2023, to August 16th, 2023.

Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation.

No ethical review processes were conducted.

Does the dataset relate to people? If not, you may skip the remaining questions in this section.
No.

Did you collect the data from the individuals in question directly, or obtain it via third parties or other sources (e.g., websites)?

Were the individuals in question notified about the data collection? If so, please describe (or show with screenshots or other information) how notice was provided, and provide a link or other access point to, or otherwise reproduce, the exact language of the notification itself.

Did the individuals in question consent to the collection and use of their data? If so, please describe (or show with screenshots or other information) how consent was requested and provided, and provide a link or other access point to, or otherwise reproduce, the exact language to which the individuals consented.

If consent was obtained, were the consenting individuals provided with a mechanism to revoke their consent in the future or for certain uses? If so, please provide a description, as well as a link or other access point to the mechanism (if appropriate).

Has an analysis of the potential impact of the dataset and its use on data subjects (e.g., a data protection impact analysis) been conducted? If so, please provide a description of this analysis, including the outcomes, as well as a link or other access point to any supporting documentation.

Any other comments?

Preprocessing/cleaning/labeling

Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.
We exclude projects created before 2023-04-01 and select projects that utilize a particular version from a library based on the `requirements.txt` file.

Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.
The raw dataset of repositories and the dataset suitable for code generation tasks are both accessible.

Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.
No software was used.

Any other comments?

Uses

Has the dataset been used for any tasks already? If so, please provide a description.
No.

Is there a repository that links to any or all papers or systems that use the dataset? If so, please provide a link or other access point.
N/A.

What (other) tasks could the dataset be used for?

We leverage the raw dataset to formulate a task for evaluating the ability of a language model to generate version-specific code. Additionally, our dataset can be used for other tasks focusing on the assessment of repository-specific attributes and their various versions.

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses? For example, is there anything that a future user might need to know to avoid uses that could result in unfair treatment of individuals or groups (e.g., stereotyping, quality of service issues) or other undesirable harms (e.g., financial harms, legal risks) If so, please provide a description. Is there anything a future user could do to mitigate these undesirable harms?

No.

Are there tasks for which the dataset should not be used? If so, please provide a description.

No.

Any other comments?

Distribution

Will the dataset be distributed to third parties outside of the entity (e.g., company, institution, organization) on behalf of which the dataset was created? If so, please provide a description.

No.

How will the dataset will be distributed (e.g., tarball on website, API, GitHub) Does the dataset have a digital object identifier (DOI)?

The dataset is distributed as an open-source dataset hosted on the HuggingFace Hub. Currently, the dataset can be accessed at `https://shorturl.at/dvJ45`

When will the dataset be distributed?

The dataset will be distributed with open-access after the acceptance of the paper.

Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)? If so, please describe this license and/or ToU, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms or ToU, as well as any fees associated with these restrictions.

The licensing is dependent on each file within the dataset.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any relevant licensing terms, as well as any fees associated with these restrictions.

No.

Do any export controls or other regulatory restrictions apply to the dataset or to individual instances? If so, please describe these restrictions, and provide a link or other access point to, or otherwise reproduce, any supporting documentation.

No.

Any other comments?

Maintenance

Who will be supporting/hosting/maintaining the dataset?

The authors of the paper will be collectively responsible in maintaining, supporting and hosting the dataset.

How can the owner/curator/manager of the dataset be contacted (e.g., email address)?

The authors of the dataset can be contacted via email which shall be publicly available post acceptance of the paper.

Is there an erratum? If so, please provide a link or other access point.

N/A

Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)? If so, please describe how often, by whom, and how updates will be communicated to users (e.g., mailing list, GitHub)?

Any updates will be noted on the HuggingFace hub.

If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were individuals in question told that their data would be retained for a fixed period of time and then deleted)? If so, please describe these limits and explain how they will be enforced.

N/A

Will older versions of the dataset continue to be supported/hosted/maintained? If so, please describe how. If not, please describe how its obsolescence will be communicated to users.

Yes, older versions of the dataset shall be hosted and maintained on the HuggingFace hub.

If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so? If so, please provide a description. Will these contributions be validated/verified? If so, please describe how. If not, why not? Is there a process for communicating/distributing these contributions to other users? If so, please provide a description.

Not for the moment. If there is, it will be noted on the HuggingFace hub.

Any other comments?