Appendix

Anonymous Author(s)

A BACKGROUND

A.1 Software Development Assistant

With the increasing reliance on software systems, the demand for innovative software solutions has surged significantly [11]. However, the process of software development remains complex and challenging for developers who face numerous obstacles throughout the development lifecycle.

One of the primary challenges in software development is the constant evolution of technology [4, 22, 24]. As new technologies emerge and existing ones advance, developers must continuously adapt and assimilate new concepts into their projects. Keeping up with these technological advancements can be overwhelming and time-consuming, often leading to delayed project timelines and increased development costs. Furthermore, the design and implementation of software artifacts require meticulous planning and attention to detail [12, 35]. Developers need to carefully architect the software components, ensuring that they are scalable, maintainable, and aligned with the project objectives. The process of transforming abstract ideas into functional software solutions involves intricate decision making, making the development phase a critical and resource-intensive aspect of the software development lifecycle. Another significant challenge lies in handling exceptions and errors that may arise during the development process [9, 25]. As the complexity of the software increases, the likelihood of encountering bugs and issues also increases. Identifying, debugging, and resolving these problems effectively can be time consuming and can hinder progress if not managed efficiently.

In order to address these challenges, there is an urgent demand for software development assistants [43] that can significantly improve the efficiency and effectiveness of the development process. These assistants, often powered by artificial intelligence and machine learning algorithms, have the potential to revolutionize the way developers work. By providing intelligent and context-aware recommendations, code suggestions, and error analyses, these assistants can enhance developers' abilities, leading to faster development cycles and improved software quality. We are the first to develop a software development assistant based on recently powerful large language models.

A.2 Large Language Models

Large language models (LLMs) have recently emerged as powerful tools in natural language processing (NLP), demonstrating remarkable achievements across a wide spectrum of tasks [3, 37, 44, 47–49]. These models, including GPT-3 [3], BLOOM [44] and LLaMA [37], typically employ a multi-layer Transformer architecture [38] with billions of training parameters. They are trained on massive corpora of unlabeled data, often containing hundreds of billions or even a trillion tokens, enabling them to capture substantial domain knowledge without relying on task-specific training data. Their self-supervised pre-training approach has been a critical factor contributing to their remarkable success. Among these LLMs, LLaMA has gained significant attention as it is a collection of open and efficient LLMs that range from 7B to 65B parameters. Built on the transformer decoder architecture, LLaMA is trained on trillions of tokens and exhibits superior performance in various aspects [37]. Our primary objective is to enable LLaMA to understand developers' intent and generate human-like responses.

A.3 Data Generation with LLMs

Collecting a large-scale dataset comprising human-annotated instructions and corresponding responses can be a time-consuming and labor-intensive endeavor. To overcome this challenge, researchers have turned to alternative approaches that leverage the capabilities of LLMs to generate such data. One notable method is Self-Instruct [40], which proposes a pipeline to utilize existing collections of instructions and a large language model to create more broad-coverage instructions that define diverse tasks, often introducing new ones. Building upon this idea, Alpaca [36] leverages Self-Instruct and Text-Davinci-003 [1] (a powerful LLM) to generate a dataset of 52K instruction-based data. Surprisingly, when fine-tuning LLaMA-7B using this dataset, Alpaca exhibits a remarkable understanding of human intent. Subsequent efforts like codealpaca [5], alpaca-cot [34], GPT4ALL [1], ShareGPT [10], Dolly-v2 [8], BELLE [16], Vicuna [7], Koala [13], Baize [45], Wizardlm [46] and others have further explored data augmentation with LLMs. While previous works have focused on generating general-purpose data, our research aims to generate data for the domain of software engineering.

A.4 Instruction Fine-tuning

The primary objective of instruction fine-tuning is to equip the model with the capability to handle a diverse range of NLP tasks [17, 23, 30, 41, 42]. These models usually convert an amount of NLP tasks into a unified format and are trained with the paradigm of multi-task learning to facilitate cross-task generalization. As a result, they often achieve promising results on new tasks. However, understanding human-written instructions remains challenging for these models [28]. OpenAI addresses this challenge by curating a substantial amount of instruct-based datasets, which encompass human-written instructions and their corresponding desired outputs across a wide array of tasks [28]. Leveraging this dataset and reinforcement learning from human feedback (RLHF) [28, 50], they enable the model to comprehend human instructions and generate human-like responses. This line of development has led to impressive works like ChatGPT [26] and GPT4 [27]. More recent models, such as Alpaca [36] and Baize [45], leverage ChatGPT to generate instruction-based data and fine-tune LLaMA on it, enabling the LLaMA model to align with human intent. Our model's primary goal is to empower LLaMA to understand developers' intent, extending its capabilities to the domain of software engineering.

[1] https://platform.openai.com/docs/models/gpt-3-5

B EXPERIMENTAL DESIGN

B.1 Evaluation Datasets

We primarily focus on verifying the effectiveness of SoTaNa in answering Stack Overflow questions. Additionally, we evaluate its capabilities in code understanding and generation.

Stack Overflow Question Answering: For evaluating the model's ability to answer Stack Overflow questions, we use the SoSum dataset [18], which contains question titles, question bodies, and answer posts with positive scores, along with summarizations of the posts. The dataset was originally intended for evaluating post-summarization models, but we repurpose it to assess question answering (QA) capabilities. Specifically, we feed the question title and body to models, the models are required to generate answers. From the original test set of 506 questions, we exclude 86 questions where large code snippets or images are replaced with *BIGBLOCK*, rendering them incomprehensible. After filtering, we proceed with the evaluation using the remaining 420 questions.

Code Generation: To evaluate the effectiveness of models on code generation, we utilize the widely-used HumanEval [6] dataset, consisting of 164 function-level programming problems in Python. The task requires models to generate the body of a function based on its signature and English description. The evaluation includes test cases to assess the generated code, with an average of 7.7 test cases per problem.

Code Summarization: For evaluating the models' ability to understand code, we use the TL-CodeSum [15] dataset. This dataset is typically used to assess code summarization models. Specifically, given a code snippet, models are required to generate one natural language sentence to describe the semantics of the code. We conduct evaluations on the first 100 examples in the test set to verify the models' code understanding capabilities.

B.2 Baselines

To evaluate the effectiveness of our approach, we compare SoTaNa with two related models, namely LLaMA [37] and Alpaca [36].

LLaMA [37] is a collection of open large pre-trained language models ranging from 7B to 65B parameters. These models are built on the Transformer decoder [38] and pre-trained with approximately 1T tokens from diverse sources such as books, GitHub, Wikipedia, arXiv, and more. Due to the large size, the 65B model cannot be loaded on a single A100 GPU card with 80G memory. Therefore, we focus on the other three sizes (7/13/30B). We denote them as LLaMA-7B, LLaMA-13B, and LLaMA-30B, respectively.

Alpaca [36] is derived from the LLaMA-7B model and fine-tuned with 52K instruction-based data generated by Text-Davinci-003.

B.3 Experimental Settings

Following the previous studies [36, 45], we set the maximum length of the input sequence to 512. The rank r and the constant α in LoRA are set to 8 and 16. To reduce memory usage and speed up the training process, we initialize LLaMA weights with 8-bit integer format. For parameters of LoRA, following the previous work [14], we adopt a random Gaussian initialization for matrix $\mathbf{A}$, while setting matrix $\mathbf{B}$ to zero. This results in the value of $\mathbf{BA}$ being zero at the beginning of training. We inject low-rank decomposition matrices into all linear weights in each layer of LLaMA. The number of LoRA parameters is shown in Table **??**. We utilize the Adam optimizer to update LoRA parameters with a batch size of 512 and learning rates of 1e-4. The dropout rate for LoRA parameters is set to 0.05. LLaMA-7B, LLaMA-13B, and LLaMA-30B are fine-tuned for 5, 5, and 3 epochs, respectively. All experiments are conducted on an NVIDIA A100-80GB GPU. We denote SoTaNa with 7B, 13B, and 30B as SoTaNa-7B, SoTaNa-13B, and SoTaNa-30B, respectively.

B.4 Evaluation Metrics

We evaluate the quality of generated answers for Stack Overflow questions and generated summarization for given code snippets via four metrics: BLEU [29], Meteor [2], Rouge-L [21], and Cider [39]. There are many variants of BLEU being used to measure the generated code summarization [31]. We choose BLEU-DC (a sentence-level BLEU with smoothing function four), which correlates with human perception the most [31]. Additionally, to evaluate code generation models, following previous work [6], we employ the widely-used Pass@1 as the evaluation metric.

C AUTOMATICAL EVALUATION EXAMPLE

Automatical metrics such as BLEU, based on token-based similarity, might not fully reflect the quality of the generated answers. For instance, consider the example in Table 1, where the question is "How to get(extract) a file extension in PHP? " and the corresponding referenced answer is "<code>pathinfo()</code>". Many models (LLaMA-30B, Alpaca-7B, Alpaca-30B, SoTaNa-7B, and SoTaNa-30B) correctly suggest using the *pathinfo()* function to extract a file extension in PHP. However, the answers received low or inconsistent scores in BLEU, Rouge-L, and Cider, highlighting the limitations of these metrics in evaluating answer quality. Specifically, all the answers are scored 0 in terms of BLEU, Rouge-L, and Cider, regardless of whether they are correct or not. While the answers of Alpaca-7B and Alpaca-30B outperform LLaMA-30B by avoiding irrelevant sentences, the Meteor score of LLaMA-30B is higher than that of Alpaca-7B and Alpaca-30B. Therefore, to comprehensively study the effectiveness of our approach SoTaNa, conducting human evaluations is necessary.

D HUMAN EVALUATION CRITERIA

Inspired by previous work [31–33], we conduct a human evaluation to evaluate the effectiveness of SoTaNa. Specifically, we invite 10 volunteers with excellent English abilities and over three years of software development experience. Each volunteer is asked to assign scores from 0 to 4 (with higher scores indicating better quality) to the generated answers based on four aspects: **Alignment** (the degree of understanding questions and providing relevant answers), **Accuracy** (the extent of providing correct information and valid solutions), **Readability** (grammatical correctness, the level of fluency and formatting), and **Confidence** (the degree of confidence in their evaluation). Each pair is evaluated by two volunteers, and the final score (excluding confidence) is the average of their assessments. Detailed scoring criteria, examples, and corresponding explanations are provided in the Table 2.

Table 1: Results generated by different models.

Question Title	How to get(extract) a file extension in PHP?				
Question Body	This is a question you can read everywhere on the web with various answers etc. However, there is always "the best way" and it should be on Stack Overflow.				
Reference	<code>pathinfo()</code>				
Model	Generated Answer	BLEU	Met.	Rou.L	Cid.
LLaMA-7B	Question title: How to get(extract) a file extension in PHP? Question body: This is a question you can read everywhere on the web with various answers: etc. However, there is always "the best way" and it should be on Stack Overflow.	0.0	1.21	0.0	0.0
LLaMA-13B	Answer the question on Stack Overflow. ### Output: Answer the question on Stack Overflow.	0.0	1.11	0.0	0.0
LLaMA-30B	The best way to get a file extension in PHP is to use the `pathinfo()` function. ### Output: The best way to get a file extension in PHP is to use the `pathinfo()` function.	0.0	6.57	0.0	0.0
Alpaca	The best way to get a file extension in PHP is to use the function `pathinfo()`. This function will return an array containing the filename, extension, path, and other information about the file.	0.0	6.13	0.0	0.0
Alpaca-13B	The best way to get (extract) a file extension in PHP is to use the `explode()` function. This function takes a string and a delimiter and returns an array of strings. The delimiter can be used to separate the file extension from the rest of the string. For example, if the string is "myfile.txt", the delimiter can be set to "." and the `explode()` function will return an array with two elements: "myfile" and "txt".	0.0	2.4	0.0	0.0
SoTaNa-7B	The best way to get a file extension in PHP is to use the `pathinfo()` function.	0.0	6.91	0.0	0.0
SoTaNa-13B	The best way to get (extract) a file extension in PHP is to use the `explode()` function. This function takes a string and splits it into an array based on a specified delimiter. In this case, the delimiter is the dot ("."), which is used to separate the file extension from the rest of the file name. The `explode()` function returns an array with the file extension at the end of the array.	0.0	3.6	0.0	0.0
SoTaNa-30B	The best way to get(extract) a file extension in PHP is to use the `pathinfo()` function. This function returns an array containing information about the given path, including the file extension. For example, if the given path is "/path/to/file.txt", the `pathinfo()` function will return an array with the file extension "txt".	0.0	7.34	0.0	0.0

E DISCUSSION

The work most closely related to our research is the StarChat [19] project. They fine-tune a model called StarCoder, which is designed specifically for code, using general-purpose data to make StarCoder [20] capable of handling dialogue. In contrast, our approach centers around using software engineering-related data to fine-tune a general-purpose large language model, with the aim of creating a software development assistant.

F THREATS TO VALIDITY

We identify some threats in this study as follows:

Data Quality. Another potential concern lies in the data generation process using ChatGPT. While we have filtered out some low-quality datasets, such as instructions with less than 3 words, we acknowledge that human checks for the correctness of the generated data were not performed. To improve the quality of generated data, future work can incorporate human checks or propose alternative approaches to ensure data accuracy and reliability.

Evaluation Datasets. The experiments have been conducted on widely-used datasets; however, there are other datasets available for each downstream task. These alternative datasets may differ in construction methods and corpus sizes, potentially leading to variations in model performance. To enhance the robustness of our findings and conclusions, further experiments on a broader range of datasets can be carried out to validate the generalizability of the results.

Evaluation Metrics. We have utilized commonly-used metrics to assess the performance of the models. However, it is crucial to recognize that these metrics may have inherent limitations. For example, metrics like BLEU and METEOR rely on textual similarity and may not fully capture the semantic similarity between two sentences. To address these limitations and obtain a more comprehensive evaluation, we also conducted human evaluations. However, it's worth noting that human evaluations can be labor-intensive and time-consuming. In future research, we will explore new automatic evaluation metrics that are more aligned with human perception.

REFERENCES

[1] Yuvanesh Anand, Zach Nussbaum, Brandon Duderstadt, Benjamin Schmidt, and Andriy Mulyar. 2023. GPT4All: Training an Assistant-style Chatbot with Large Scale Data Distillation from GPT-3.5-Turbo. https://github.com/nomic-ai/gpt4all.

[2] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *IEEvaluation@ACL*.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Lan Cao and Balasubramaniam Ramesh. 2008. Agile requirements engineering practices: An empirical study. *IEEE software* 25, 1 (2008), 60–67.

[5] Sahil Chaudhary. 2023. Code Alpaca: An Instruction-following LLaMA model for code generation. https://github.com/sahil280114/codealpaca.

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[7] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/

Table 2: Scoring criteria. Examples on "How to get(extract) a file extension in PHP?".

Category	Score	Scoring Criteria	Example	Explanation
Alignment	0	The answer is entirely irrelevant, containing content that is unrelated to the question's topic.	Cats are great pets because they are low-maintenance and independent.	The answer is entirely irrelevant because it discusses pets, which have no connection to the topic of extracting file extensions in PHP.
	1	The answer is somewhat related to the topic, but its connection to the question is weak and not directly focused on the problem.	You can determine a file type by looking at the file name.	The answer is somewhat related to the topic as it mentions file type determination, but it doesn't provide a direct solution for extracting a file extension in PHP.
	2	The answer is relevant, displaying an understanding of the question's topic, but it may not encompass all aspects or nuances of the problem.	In PHP, you can find the file extension and name by.	The answer is relevant because it mentions the file extension, but it lacks practical solutions related to "How to".
	3	The answer is highly relevant, demonstrating a deep comprehension of the question's topic and closely connecting to all aspects of the problem.	To find a file extension in PHP, you can split the file name with a delimiter and retrieve the last part.	The answer is highly relevant because it suggests a method for finding file extensions in PHP, although it might not be entirely accurate.
Accuracy	0	The answer is entirely incorrect, providing false information or suggesting an invalid solution.	Use the 'strlen()' function to find the file extension in PHP	The answer is entirely incorrect because the 'strlen()' function is used to find the length of a string, not to extract a file extension..
	1	The answer contains some correct information but also has significant inaccuracies or misconceptions.	Use the pathinfo() function. It returns the extension directly.	The answer is partially correct, as it suggests using 'pathinfo()', but it returns an array rather than the extension.
	2	The answer is mostly accurate, with only minor errors or omissions.	Use pathinfo() in PHP to get file information, including the extension and filedir.	The answer is mostly accurate as it mentions the correct function to get file information. However, it should be 'dirname' instead of 'filedir'.
	3	The answer is completely accurate, providing correct information and a valid solution.	Use the pathinfo() function in PHP to extract the file extension: $extension = pathinfo($filename, PATHINFO_EXTENSION);	The answer is completely accurate, providing a correct PHP function along with an example.
Readability	0	The answer is extremely difficult to understand, with poor grammar, structure, or excessive jargon.	PHP file get extension method apply for find out.	The answer is extremely difficult to understand due to poor grammar and sentence structure.
	1	The answer is somewhat difficult to understand or has some grammatical errors and unclear explanations.	php use pathinfo get file info eg extenion,basenamee,filenme	The answer is somewhat difficult to understand due to the lack of a concrete example and proper grammar.
	2	The answer is clear, well-structured, and has only minor grammatical errors or room for improvement.	=Use the pathinfo() to extract extension: $extension = pathinfo($filename, PATHINFO_EXTENSION);	The answer provides a code example, but the readability is reduced due to the unnecessary symbol "==".
	3	The answer is very clear, well-structured, and free from grammatical errors, making it easy to understand.	Use the pathinfo() function in PHP to extract the file extension: $extension = pathinfo($filename, PATHINFO_EXTENSION)	The answer is very clear, well-structured, and free from grammatical errors, making easy understanding.
Confidence	0	The rater is not at all confident in his evaluation of the answer and feels unsure about the assigned scores.	/	/
	1	The rater has low confidence in their evaluation and may have doubts about the assigned scores.	/	/
	2	The rater is fairly confident in their evaluation, with only minor uncertainties about the assigned scores.	/	/
	3	The rater is highly confident in their evaluation and feels certain about the assigned scores.	/	/

[8] Mike Conover, Matt Hayes, Matt Mathur, Xiangrui Meng, Jianwei Xie, Jun Wan, Ali Ghodsi, Patrick Wendell, and Patrick Zaharia. 2023. Hello Dolly: Democratizing the magic of ChatGPT with open models. https://github.com/databrickslabs/dolly

[9] Chrysanthos Dellarocas and Mark Klein. 2000. A knowledge-based approach for handling exceptions in business processes. *Information Technology and Management* 1 (2000), 155–169.

[10] Domeccleston. 2023. ShareGPT – Share your wildest ChatGPT conversations with one click. https://github.com/domeccleston/sharegpt

[11] DRM Associates. 2002. New Product Development Glossary. Archived from the original on 13 July 2018. http://www.npd-solutions.com/glossary.html

[12] William A Florac and Anita D Carleton. 1999. *Measuring the software process: statistical process control for software process improvement.* Addison-Wesley Professional.

[13] Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A Dialogue Model for Academic Research. Blog post. https://bair.berkeley.edu/blog/2023/04/03/koala/

[14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[15] Xing Hu, Ge Li, Xin Xia, David Lo, Shuai Lu, and Zhi Jin. 2018. Summarizing source code with transferred api knowledge. (2018).

[16] Yunjie Ji, Yong Deng, Yan Gong, Yiping Peng, Qiang Niu, Lei Zhang, Baochang Ma, and Xiangang Li. 2023. Exploring the Impact of Instruction Data Scaling on Large Language Models: An Empirical Study on Real-World Use Cases. *arXiv preprint arXiv:2303.14742* (2023).

[17] Yunjie Ji, Yan Gong, Yong Deng, Yiping Peng, Qiang Niu, Baochang Ma, and Xiangang Li. 2023. Towards Better Instruction Following Language Models for Chinese: Investigating the Impact of Training Data and Evaluation. *arXiv preprint arXiv:2304.07854* (2023).

[18] Bonan Kou, Yifeng Di, Muhao Chen, and Tianyi Zhang. 2022. SOSum: a dataset of stack overflow post summaries. In *MSR*. 247–251.

[19] Tunstall Lewis, Lambert Nathan, Beeching Nazneen, Rajaniand Edward, Le Scao Teven, Han Sheon, Schmid Philipp, von Werra Leandro, and Sasha Rush. 2023. Creating a Coding Assistant with StarCoder. https://huggingface.co/blog/starchat-alpha.

[20] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).

[21] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out.*

[22] Tommi Mikkonen, Casper Lassenius, Tomi Männistö, Markku Oivo, and Janne Järvinen. 2018. Continuous and collaborative technology transfer: Software engineering research with real-time industry impact. *IST* 95 (2018), 34–45.

[23] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2021. Cross-task generalization via natural language crowdsourcing instructions. *arXiv preprint arXiv:2104.08773* (2021).
[24] Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. 2005. Challenges of migrating to agile methodologies. *Commun. ACM* 48, 5 (2005), 72–78.
[25] Bashar Nuseibeh. 1996. To be and not to be: On managing inconsistency in software development. In *Proceedings of the 8th International Workshop on Software Specification and Design*. IEEE, 164–169.
[26] OpenAI. 2022. Chatgpt: Optimizing language models for dialogue. (2022).
[27] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).
[28] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.
[29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *ACL*.
[30] Victor Sanh, Albert Webson, Colin Raffel, et al. 2021. Multitask Prompted Training Enables Zero-Shot Task Generalization. (Oct. 2021). arXiv:2110.08207
[31] Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang, and Hongbin Sun. 2022. On the evaluation of neural code summarization. In *ICSE*. 1597–1608.
[32] Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021. CAST: Enhancing Code Summarization with Hierarchical Splitting and Reconstruction of Abstract Syntax Trees. In *EMNLP*. 4053–4062.
[33] Ensheng Shi, Yanlin Wang, Wei Tao, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022. RACE: Retrieval-augmented Commit Message Generation. In *EMNLP*. 5520–5530.
[34] Qingyi Si, Tong Wang, Naibin Gu, Rui Liu, and Zheng Lin. 2023. Alpaca-CoT: An Instruction Fine-Tuning Platform with Instruction Data Collection and Unified Large Lnguage Models Interface. https://github.com/PhoebusSi/alpaca-CoT.
[35] Terry Stone. 2010. *Managing the Design Process-Implementing Design: An Essential Manual for the Working Designer*. Rockport Publishers.
[36] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A Strong, Replicable Instruction-Following Model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html* (2023).
[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[39] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: Consensus-based image description evaluation. In *CVPR*.
[40] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560* (2022).
[41] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *EMNLP*. 5085–5109.
[42] Jason Wei, Maarten Bosma, Vincent Y. Zhao, et al. 2021. Finetuned Language Models Are Zero-Shot Learners. *arXiv:2109.01652 [cs]* (Sept. 2021). arXiv:2109.01652 [cs]
[43] Terry Winograd. 1973. Breaking the complexity barrier again. *ACM Sigplan Notices* 10, 1 (1973), 13–30.
[44] BigScience Workshop, Teven Le Scao, Angela Fan, et al. 2022. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model.
[45] Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. 2023. Baize: An open-source chat model with parameter-efficient tuning on self-chat data. *arXiv preprint arXiv:2304.01196* (2023).
[46] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).
[47] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: An Open Bilingual Pre-trained Model. In *ICLR*. https://openreview.net/forum?id=-Aw0rrrPUF
[48] Susan Zhang, Stephen Roller, Naman Goyal, et al. 2022. OPT: Open Pre-Trained Transformer Language Models. arXiv:arXiv:2205.01068
[49] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).
[50] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, et al. 2020. Fine-Tuning Language Models from Human Preferences.