Distilled GPT for Source Code Summarization

Chia-Yi Su[1*] and Collin McMillan[1]

[1*]Department of Computer Science, University of Notre Dame,
Holy Cross Dr, Notre Dame, 46556, Indiana, USA.

*Corresponding author(s). E-mail(s): csu3@nd.edu;
Contributing authors: cmc@nd.edu;

Abstract

A code summary is a brief natural language description of source code. Summaries are usually only a single sentence long, and yet form the backbone of developer documentation. A short descriptions such as "changes all visible polygons to the color blue" can give a programmer a high-level idea of what code does without the effort of reading the code itself. Recently, products based on Large Language Models such as ChatGPT have demonstrated a strong ability to write these descriptions automatically. However, to use these tools, programmers must send their code to untrusted third parties for processing (e.g., via an API call). This loss of custody is not acceptable to many organizations. In this paper, we present an alternative: we train an open source model using sample output generated by GPT-3.5 in a process related to knowledge distillation. Our model is small enough (350m parameters) to be run on a single 16gb GPU, yet we show in our evaluation that it is large enough to mimic GPT-3.5 on this task.

Keywords: source code summarization, software documentation generation

1 Introduction

A code summary is a brief, natural language description of source code. Summaries are typically only a single sentence. When reading a Java method, for instance, a programmer may start with the Javadoc sentence "changes all visible polygons to the color blue." The summary provides a quick way for the programmer to understand what the method does without having to read the code itself. The benefits of summaries in documentation have been studied for decades (Haiduc et al, 2010), and Software Engineering (SE) research has long sought to automate the process of writing them,

to reduce manual effort by programmers, support under-documented legacy programs, and build accessibility tools (Robillard et al, 2017). Code summaries form the backbone of much documentation for programmers, and the dream of automatic generation of these summaries has been described as a "holy grail" of SE research (Allamanis et al, 2018a; Forward and Lethbridge, 2002; LeClair et al, 2019).

Recently, the dream seems within reach. Years of effort on neural code summarization techniques has culminated in products such as Copilot (Github, 2022) and ChatGPT (OpenAI, 2022), which exhibit an ability to describe arbitrary code (Ma et al, 2023). At the heart of these products is a language model that is trained using big data input. The language model in the most powerful products may be tens or hundreds of billions of parameters, and the data input often includes trillions of tokens, such as the entirety of public GitHub repositories, plus StackOverflow, Wikipedia, etc. The effectiveness of these products has captured the public imagination and helped drive a new wave of research (Sun et al, 2023). Like in many research areas, the decades-long effort toward automatic code summarization suddenly seems at hand.

Yet a major problem looms. For programmers to use these tools, they must send their code to third parties for processing. An IDE plugin wishing to use GPT-3.5, for example, must harvest code from the programmer's codebase and send it via an API call to OpenAI. This call is a loss of data custody and a non-starter for many institutions (Derner and Batistič, 2023). In addition, the closed nature of these products has caused controversy among researchers, who point out a lack of reproducibility, potential data contamination from public test sets to private training data, and resultant loss of scientific rigor (Hellendoorn and Sawant, 2021). The situation for many programmers is that the technology to automate a major portion of code summarization exists, but it is not usable.

In this paper, we present an alternative: knowledge distillation from a large model (GPT-3.5) to smaller models. Our paper has three key novel research contributions:

1. We present a study comparing summaries from GPT-3.5 to the reference summaries written by human programmers, and show that the generated summaries tend to be superior, indicating they are good source of training data. (Section 3)
2. We present a study of knowledge distillation of these summaries for the size of model (38m - 15.5B parameters) and size of training data (170k - 2.15m samples). We collect 2.15m summaries generated by GPT-3.5 for Java methods. We use a simple prompt and methods from open-source Java programs. As foundation models, we compare `jam` (Su et al, 2023) and `starcoder` (Li et al, 2023b). The `jam` model is pretrained on 52m Java methods and has an easily-searchable dataset to ensure reproducibility and a controlled experimental framework. The `starcoder` model is much larger and has a larger pretraining dataset, but is also more expensive and has more non-controllable experimental variables due the the dataset. (Section 4)
3. We evaluate our distilled model against GPT-3.5 in a study with human experts. (Section 5)

We release all code and implementation details. The model we recommend from our experiments can be run from a workstation with a single 16GB GPU, which is relatively low cost for many organizations. This low cost and open source structure enables

programmers to access automatic code summarization while keeping data custody. Although there are some papers that have already formulated the code summarization as a fine-tuning problem such as Wang et al (2021); Bender et al (2021) and studied knowledge distillation for smaller models from larger models Hsieh et al (2023); Yu et al (2023), we thoroughly explore the data and model size for knowledge distillation on code summarization and conduct the human study to compare the language models generated summary and human reference with human experts.

2 Background and Related Work

This section discusses the two key areas of background and related work: code summarization and knowledge distillation.

2.1 Source Code Summarization

The term "source code summarization" was coined in 2010 by Haiduc et al (2010) to refer to the task of writing natural language descriptions of code. Until 2017, most of research methods focused on IR-based and template-based methods. From 2017 to

	I	N	G	T	C
McBurney et al (2016)	x				x
Iyer et al (2016)		x			
Rodeghero et al (2017)	x				x
Fowkes et al (2017)	x				
Loyola et al (2017)		x			
Jiang et al (2017)		x			
Hu et al (2018a)		x	x		
Hu et al (2018b)		x			
Allamanis et al (2018b)		x	x		
Wan et al (2018)		x			
Liang and Zhu (2018)		x			
Alon et al (2019a,b)		x	x		
Gao et al (2019)		x			
LeClair et al (2019)		x	x		
Nie et al (2019)		x			
Lu et al (2019)		x			
Gao et al (2019)		x			
Haque et al (2020)		x			x
Haldar et al (2020)		x			
Zügner et al (2021)		x	x		
Liu et al (2021)		x	x		
Bansal et al (2021b)		x			x
Wang et al (2021)		x			
Bender et al (2021)		x			

Table 1: Snapshot of the past five years in source code summarization. Column *I* stands for IR-based techniques. *N* means neural network-based. *G* means the code is modeled as a graph. *T* means Transformer designs. *C* means learning from context.

present, neural models for code summarization becomes the most dominant research direction. Table 1 shows the history and families of different methods. Although the most dominant research line is neural models, there are different families of neural models, which include better modeling of code itself and using more context. For example, LeClair et al (2019); Alon et al (2019a,b) combined AST with source code and Allamanis et al (2018b) modeled AST as a graph for neural models. Haque et al (2020) applied the attention mechanism to the file context. Bansal et al (2021b) combined information different software projects.

More recently, Wang et al (2021); Bender et al (2021) introduce the technique to fine-tune Large Language Models (LLM) for code summarization. Although some proprietary LLM such as OpenAI's ChatGPT has demonstrated the great capability on program comprehension, we cannot avoid data leak problems because of the inaccessibility of training data on these models. In this paper, we examine the data and model size for fine-tuning and distill the knowledge from GPT-3.5 by training small models with public and controllable datasets and mimic GPT-3.5's capability for code summarization.

2.2 Knowledge Distillation

Knowledge Distillation refers to teaching a small machine learning model to behave like a larger one for a niche task (Hsieh et al, 2023; Yu et al, 2023; Wang and Yoon, 2021). Knowledge distillation is useful in scenarios where a large model may be capable of many tasks or even considered general purpose, such as ChatGPT or Copilot, but is too expensive or impractical to use for certain specific tasks. A classic application is in image classification, where a powerful model capable of classifying many types may be used to teach a smaller model with some tradeoffs, such as lower accuracy or recognizing fewer categories (Gou et al, 2021; Wang and Yoon, 2021; Zagoruyko and Komodakis, 2016). More recently, general purpose models such as GPT-3.5 have been used to teach smaller models to perform question-answering tasks (Zhang et al, 2023), assessment of student answers (Li et al, 2023a), follow specific types of instructions (Tang et al, 2023), and to improve output from existing smaller chatbots (Chen et al, 2023).

An important point in knowledge distillation is that there is almost always a trade made in exchange for the reducing model size. Gudibande et al (2023) make the point that small models attempting to replicate all capabilities of ChatGPT, for instance, are likely to face problems with generalization to prompts unlike those in the training samples. They highlight how the benefit of knowledge distillation is focused on training small models to perform specific tasks – one should not necessarily expect the small model to do all tasks well, but the small model can mimic the large one by specializing on a single task. In this paper, we focus on the task of code summarization, and demonstrate how a small model can perform on par with the large one in this specialty.

3 GPT-3.5 and Human-written References

This section discusses our comparison of source code summaries generated by GPT-3.5 to summaries written by human programmers. We also compare the summaries

generated by the model trained with GPT-3.5 summaries and the summaries written by human programmers. The purpose of this comparison is to determine whether summaries generated by GPT-3.5 are suitable replacements for human-written ones. This comparison is necessary in this paper because we seek to distill GPT-3.5's ability to generate summaries for a smaller model, so we should measure the quality of the summaries GPT-3.5 generates. If GPT-3.5 generates poor quality summaries, then it would not be suitable for distillation. Thus, we ask the Research Question (RQ):

RQ1 How well do summaries generated by GPT-3.5 compare to human-written reference summaries, across key quality criteria established in relevant literature?

By "human-written reference summaries", we mean code summaries written by the programmers or other team members who wrote the underlying software, such as the summary sentence of the Javadocs for Java methods. By "key quality criteria", we mean the concepts of Accuracy, Completeness, and Concision that were first used to evaluate code summaries over ten years ago by Sridhara et al (2010) and have been used in numerous studies since McBurney et al (2016). We will establish how we measure these concepts in the next subsection. Note we also measure overall preference of one summary to another, for a gauge on how people balance the quality criteria.

3.1 Research Method

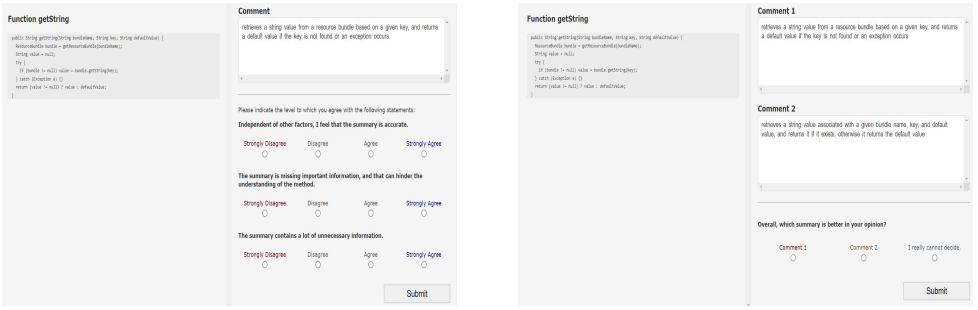
Our research method is a survey in which we show programmers different code and summaries of that code, and ask them to rank the summaries in four questions. We designed our survey to be consistent with years of best-practice in evaluating summaries, namely from Sridhara et al (2010); McBurney et al (2016); Bansal et al (2021a), including the wording we use in the survey questions. The survey has four questions per summary, divided over two pages. On the first page, the survey shows a Java method and a summary of that method, plus these three questions:

1. Independent of other factors, I feel that the summary is accurate.
2. The summary is missing important information, and that can hinder the understanding of the method.
3. The summary contains a lot of unnecessary information.

The first question is intended to measure accuracy, the second measures completeness, and the third measures concision. Following each question are four radio buttons to select one of: "Strongly Agree", "Agree", "Disagree", and "Strongly Disagree." On the next page, the survey shows the same Java method and summary, plus another summary for the same Java method. The first summary is either the summary from GPT-3.5 or the human-written reference (decided randomly). The second summary is the alternative. The second page also shows a single question:

4. Overall, which summary is better in your opinion?

Following this question are three options: "Summary 1", "Summary 2", and "I really cannot decide." We phrased the third option such that participants would be encouraged to select between the two summaries, but still have an option to avoid the question in very difficult or impossible cases, such as if the summary were the same



(a) Page One (b) Page Two

Fig. 1: Example of the two pages of our survey for each Java method.

or both were illegible. For quality control (see Section 3.3), we also asked participants to enter a rationale for their answer to this question. Figure 1 shows each page.

3.2 Subject Source Code, Summaries, Participants

We obtained the subject source code from the `funcom-java-long` dataset provided by Su et al (2023). This dataset is a revision of the dataset provided by LeClair and McMillan (2019) to include various fixes such as those proposed by Bansal et al (2021b); Shi et al (2022). This dataset includes a training set of around 170k Java methods paired with summaries written by human experts (the origin of these summaries was Javadocs provided with the source code), as well as a test set of around 8k Java methods. The dataset also includes a total of over 52m Java methods that do not contain human-written summaries. The `funcom-java-long` dataset is diverse in that it contains over 50k Java projects from many domains collected over at least one decade. The test set of 8k methods represents 880 projects.

To obtain summaries written by GPT-3.5, we used a prompt in the format:

```
Write a one sentence description of this Java method:
```

Followed by the source code for the method. We collected summaries from GPT-3.5 using this prompt for a total of 2.15m Java methods. The 2.15m Java methods included all 170k from the `funcom-java-long` training set, all 8k from the `funcom-java-long` test set, plus 2m additional methods randomly selected from the 52m Java methods in the dataset that do not have human-written summaries. We filtered approximately 20k summaries which were either empty or not in English.

The survey randomly selected 30 Java methods from the 8k test set to show to each participant. We chose 30 because we found that participants tended to spend between two and three minutes per method, and we aimed to keep the survey time to a maximum of 90 minutes to prevent fatigue bias (Sievertsen et al, 2016).

We recruited 15 participants for each study via the Prolific platform[1]. We used Prolific's features to filter for people who were at least 25 years of age, were located in

[1] https://www.prolific.co/

the United States or United Kingdom, and had a university degree in Computer Science or Computer Engineering. We describe additional filters for biases in the Threats to Validity, Section 3.3. With 15 participants and 30 methods each, we collected feedback for 450 Java methods. Approximately half showed the human-written summaries on the first survey page (therefore answering questions 1-3), with the remainder showing the GPT-3.5 summaries on the first page. All saw both summaries on the second page. To reduce the subjective bias, we randomly selected 150 functions from the original 450 samples and we recruited additional five different participants (30 methods for each participant) to evaluate the methods with the same criteria and the same website via the Prolific platform.

3.3 Threats to Validity

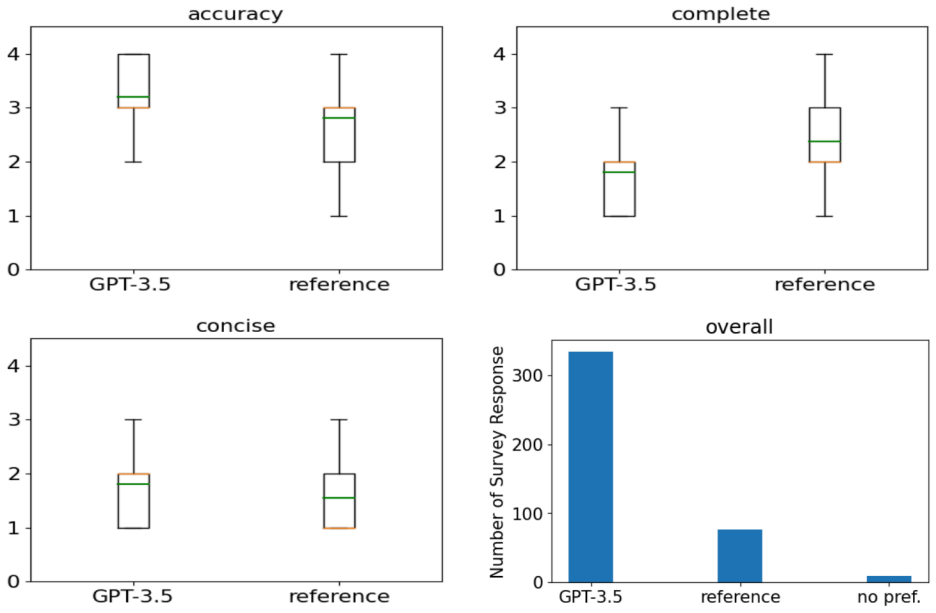
The key threats to validity in this study are: 1) the participants, 2) the GPT-3.5 version and prompt, and 3) the subject Java methods. The participant pool can be a threat to validity because online survey participants can fake work history. Danilova et al (2021) recommend programming-based screening questions, but Ghorbani et al (2023) point out that these questions are now easily circumvented with online AI-based tools such as Copilot and ChatGPT. Therefore, we manually inspect each participant's survey results for clear patterns of fraud. We rejected one participant who completed the survey in under fifteen minutes (thirty seconds per method).

The GPT-3.5 version and prompt are threats to validity because GPT-3.5 is a commercial product and subject to change without notice, and also may give different answers with different prompts. We collected the summaries between June 1 and June 30, 2023, during which no changes were reported to GPT-3.5, though these changes could have occurred unreported. Note that while the results of this study could change with different prompts or model versions, we view this paper as still valid as a framework for distilling knowledge from large models, and is still reproducible because we release the responses from GPT-3.5 as a separate dataset. In this way, our procedure is consistent with other research distilling commercial language models, such as Zhang et al (2023).

The subject Java methods are also a key threat to validity because our study results could change with a different set of Java methods. We reduce the risk by using methods from a large and well-studied dataset, with methods from projects in many domains collected over many years. Our study used a total of 450 of these methods, randomly selected from the dataset. The total of 450 is a representative sample: considering a test set population size of 8k methods, Israel (1992) sample size recommendations shows a minimum sample size of 381 at +/-5% precision tolerance.

3.4 RQ1 Results

We find that GPT-3.5 produces source code summaries that are higher quality than the reference summaries. The evidence for this finding comes in two key forms, depicted in Figure 2. First, the ratings provided by the human evaluators for accuracy and completeness are better for GPT-3.5 than for the references by a statistically-significant margin. The mean values for accuracy of GPT-3.5 are higher than the references,



	gpt-3.5		reference				
	med	mean	med	mean	Zobs	Zcrit	p
accurate	3	3.196	3	2.811	6.816	1.645	<0.01
complete	2	1.799	2	2.368	7.670	1.645	<0.01
concise	2	1.808	1	1.542	3.991	1.645	<0.01

Fig. 2: Comparison of human reference summaries to GPT-3.5, along the four questions we discuss in Section 3.1, followed by a statistical summary of the RQ1 results in Figure 2 with Mann-Whitney tests showing statistical significance.

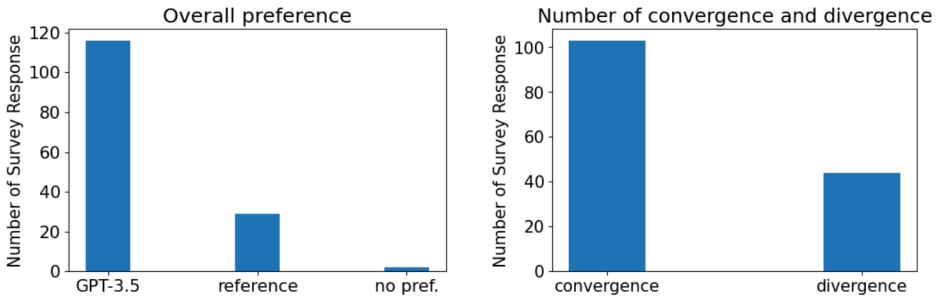


Fig. 3: Comparison of human reference summaries to GPT-3.5 in terms of overall preference and the number of convergence and divergence functions on the duplicate studies

with the 90% of values for GPT-3.5 in the 3-4 range. In other words, human evaluators marked "Strongly Agree" or "Agree" to a statement about accuracy for a 90% of summaries. For the references, only 74% were in this range. Likewise, for completeness, human evaluators rated GPT-3.5 with better scores by a statistically-significant margin (lower is better for completeness and conciseness due to the wording of the survey questions, see Section 3.1). Second, when comparing summaries, human evaluators preferred GPT-3.5 summaries in 80% of comparisons, versus 18% for references

```
private Time simulate() throws SimulationException{
  Time current = startSimulation();
  try {
    if (endTime.isLT(current)) {
      throw new SimulationException(""Requested time "" + endTime
        + "" is smaller than current time "" + current + ""!"");
    while (isRunning()) {
      if (current.isGE(endTime)) { break; }
      current = continueSimulation(current, endTime);
      Thread.yield();
  } finally {
    finishSimulation();
  }
  return currentTime();
}
```

GPT-3.5	simulates a process until a given end time and returns the final current time, handling exceptions if the end time is reached before the current time
Human Reference	increase the simulation time and execute all events with an earlier time
Trained with GPT-3.5	simulates a simulation by checking if the requested time is greater than the current time and returns the simulation time, or throws a SimulationException if the requested time is not found
Trained with Human Reference	simulate the simulation

```
private Literal promoteDecimal(Literal numeral) {
  Long result;
  try {
    result = Long.valueOf(numeral.getLabel());
  } catch (NumberFormatException e) {
  throw new TypeError(""Cannot promote non-numeral to a decimal value"");
  }
return this.factory.createLiteral(result.toString(), SPARQLConstants.DECIMAL_TYPE);
}
```

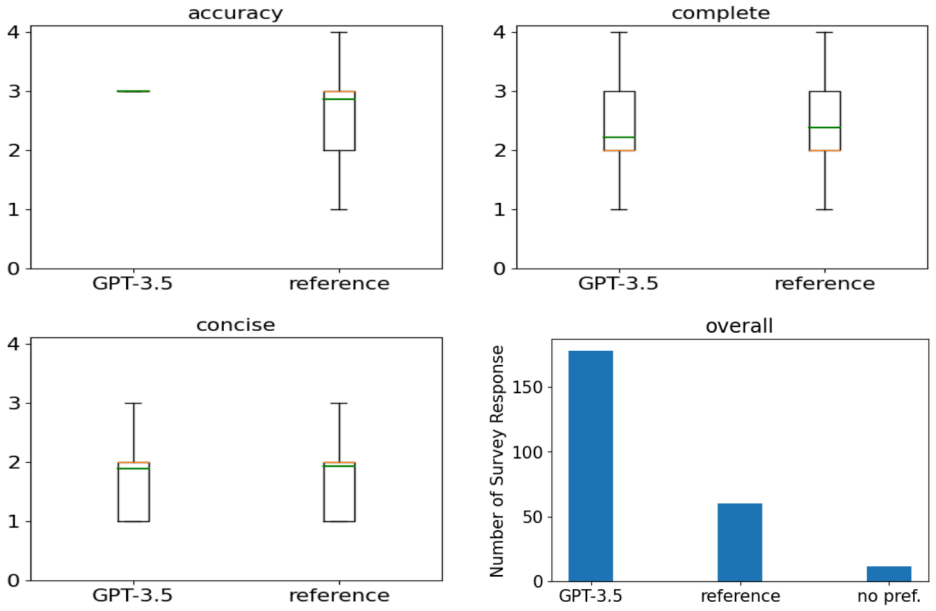
GPT-3.5	promotes a Literal numeral to a decimal value by converting it to a Long and creating a new Literal with the converted value and the DECIMAL_TYPE
Human Reference	promotes a literal to a decimal datatype reparsing the label
Trained with GPT-3.5	promotes a given Literal to a decimal type and returns the promoted Literal
Trained with Human Reference	promotes a literal to a decimal literal

Table 2: Examples of summaries generated by GPT-3.5 and the human reference for two Java methods from the dataset.

and 2% undecided. Note that the references were rated more concise that GPT-3.5 summaries, but this result is likely because the references tend to be shorter, with an average of 8.28 words versus 18.40 for GPT-3.5.

Figure 3 depicts the results for the study that we randomly sampled 150 functions from 450 functions in the first study for RQ1. We observe that 79% of the human evaluation prefers GPT-3.5 summaries, 19% of the human evaluation prefers human reference, and 2% of the answers are undecided. The overall results show that GPT-3.5 summaries are the better summary than the human reference and align with the first study. We observe that 70% of the functions converge with the first study. We do not calculate the agreement between two studies because Donker et al (1993); Delgado and Tibau (2019) point out that Cohen kappa cannot reflect the agreement in the unbalanced data. All things considered, our results show that GPT-3.5 summary is better than the human reference.

In addition, we trained the models with 170k human reference proposed by LeClair and McMillan (2019) and with the 170k summaries generated by GPT-3.5 for comparison. We use 170k data for comparison because we do not have 2.15m summaries on



	gpt-3.5		reference				
	med	mean	med	mean	Zobs	Zcrit	p
accurate	3	2.992	3	2.860	-1.232	-1.645	0.109
complete	2	2.380	2	2.219	1.440	1.645	0.075
concise	2	1.890	2	1.926	0.155	-1.645	0.561

Fig. 4: Comparison of summaries generated by models trained with human reference summaries to summaries generated by models trained with GPT-3.5, along the four questions we discuss in Section 3.1, followed by a statistical summary of the RQ1 results in Figure 4 with Mann-Whitney tests.

human reference. We depict the results in Figure 4. Overall, the evaluators prefer the summaries generated by the model trained with the GPT-3.5 summaries although the statistical test shows no statistical significance on accuracy, completeness, and conciseness. We find 71% of the answers prefer the summaries generated by the model trained with GPT-3.5 summaries versus 24% trained with human reference and 5% undecided. In terms of accuracy, we observe that the summaries generated by the model trained with GPT-3.5 is slightly higher than the summaries generated by the model trained with human reference. Similarly, evaluators rated the summaries generated by the model trained with the GPT-3.5 summaries more complete than the summaries generated by the model trained with human reference. Although we observe that the results of accuracy and complete only edges to the summaries generated by models trained with GPT-3.5, the summaries generated by the model trained with GPT-3.5 outperforms the summaries generated by the model trained with human reference in overall preference. The possible explanation is that the training data is not enough to mimic the ability of GPT-3.5 for code summarization. For example, compared with "promotes a Literal numeral to a decimal value by converting it to a Long and creating a new Literal with the converted value and the DECIMAL TYPE" generated by GPT-3.5 in Table 2, the trained model generates "promotes a given Literal to a decimal type and returns the promoted Literal", which is less complete and accurate compared with the original GPT-3.5 summary. But, compared with the reference version, this summary is still more informative. Therefore, the evaluator prefers the summary generated by the model trained with GPT-3.5. This result aligns with findings by Roy et al (2021) that ratings by human evaluators often do not appear statistical significantly different unless the summaries are very qualitatively different due to noise in how people give subjective ratings, yet people may prefer one group of summaries when asked directly.

This result is surprising considering that human-written references have long been considered the gold standard in evaluating software documentation generation LeClair and McMillan (2019); Shi et al (2022). Yet consider the examples in Table 2, which are representative of typical summaries from each source. The human-written references tend to be concise, but lack detail which can lead to confusion (e.g., the meaning of "reparse the label" is unclear in the second human-written example in Table 2, but the GPT-3.5 provides detail). These observations are borne out in the study results where GPT-3.5 is superior in accuracy and completeness, but not conciseness. They are also supported by evidence in the literature that people often produce low-quality documentation Aghajani et al (2019). In short, we find that in practice, GPT-3.5 outperforms people when writing software documentation.

4 Distilling GPT-3.5

This section discusses the knowledge distillation process and evaluation. Our objective is to study knowledge distillation of a large language model for code summarization, using the norms for model architecture and datasets prevalent in current literature. Therefore, we ask the following Research Question:

RQ2 How closely do language models mimic GPT-3.5 for code summarization, across different model and dataset sizes?

By "different model sizes", we mean models in terms of numbers of parameters. Much of the current research frontier is focused on Generative Pretrained Transformer (GPT)-like models, with model size considered a major contributor to both model output quality and resource cost (Xu et al, 2023). One goal of RQ2 is to help decision-making in balancing model output quality and costs. It is likely that a "price break" will emerge after which the expense of model size will increase faster than output quality gains, and practitioners may wish to choose a model at this break point. Likewise, by "different dataset sizes", we mean size in terms of number of samples. We collected 2.15m samples from GPT-3.5 in Section 3, though it is possible that fewer samples are needed. Additional samples increase training cost, so it may be desirable to use fewer samples.

4.1 Distillation Process for Decoder-only Language Models

At a high level, our distillation process is straightforward: we fine-tune a pretrained language model to generate source code summaries, using the summaries generated by GPT-3.5 as training samples. We use a fine-tuning process proposed by Su et al (2023) wherein we use a training prompt of the form:

```
TDAT: <Java method code>
COM: <summary of Java method>
```

We create training prompts in this form for the entire 2.15m samples we collected from GPT-3.5 in Section 3. During fine-tuning, we use the standard autoregressive process in which the model learns to produce each token in the prompt conditioned on all previous tokens. The model will learn to generate code one token at a time after the `TDAT`, then learn to generate a summary comment after the `COM`. The model will learn to generate summary comments one token at a time, conditioned on the previous tokens in that comment, as well as the Java code prior to the `COM` token. In almost all cases, the first word of the summary is an action word such as "gets", "prints", or "calculates", so the model learns to decide this word based on the Java code (Haque et al, 2021). The model decides the next word using the action word and the Java code, and continues until it produces an end-of-sequence token to denote the summary's end.

4.2 Subject Decoder-only Language Models

We fine-tune two language model architectures as part of our study, using different settings for the model and dataset sizes. Because the heart of our target model for distillation, GPT-3.5, is a decoder-only Transformer (Brown et al, 2020), both language model architectures we use are also decoder-only Transformers. One is a called `jam` and is a GPT-2-like model that is pretrained using 52m Java methods. The `jam` model was released by Su et al (2023) as a language model for working with Java code. The 52m Java methods in the pretraining dataset are searchable for code clones, to limit the possibility of data leakage from test to training set. The pretraining dataset excludes 8k samples in `funcom-java-long` test set, which we used in the study in Section 3 and later in this section. We use the key configuration parameters in Table 3, which in a

GPT-2-like architecture such as `jam` result in total network parameter sizes of 350m, 110m, and 38m.

As an alternative, we use `starcoder` (Li et al, 2023b), which is a 15.5B parameter GPT-2-like model. The `starcoder` model is pretrained with "The Stack", which is a collection of 6TB of source code in over 350 programming languages. This model serves as a strong alternative to `jam` because it represents a state-of-the-art language model, with billions of parameters and an internet-scale pretraining dataset size. Whereas `jam` is an inexpensive model with very closely-controlled experimental variables, `starcoder` is an industrial-strength model with more potential variables due to the much larger and difficult-to-search pretraining data (i.e., there is more risk of data contamination).

		`jam`			`starcoder`
d	embedding dimension	512	768	1024	6144
L	number of layers	4	10	24	40
h	attention heads	4	8	16	48
r	learning rate	3e-5	3e-5	3e-5	1e-4
e	epochs	3	3	3	3
o	dropouts	0.2	0.2	0.2	0.05
	total number of parameters	38m	110m	350m	15.5B

Table 3: Key model settings and parameters sizes.

4.3 Subject Encoder-Decoder Language Models

In addition to decoder-only transformer GPT architecture, we also use encoder-decoder models to distill the knowledge of GPT-3.5 for code summarization. We train `attendgru` (LeClair and McMillan, 2019), `transformer` (Ahmad et al, 2020), and `setransformer` (Li et al, 2023c) for 10 epochs on four different datasets. We pick the one with the best accuracy on the validation set as the model for prediction. Table 4 shows the settings and parameters that we use for encoder-decoder models. We summarize three different models as follows:

`attendgru` is the model that uses GRU as a backbone with the attention mechanism to form the encoder-decoder architecture.

`transformer` uses multi-head attention with the key, query, and value vector for parallelization of the `attendgru` model.

`setransformer` uses transformer as a base with an additional context, abstract syntax tree of the function, as an input and computes the convolution of each input feature.

Parameters	Description	Settings
d	embedding dimension	100
b	batch size	50
l	learning rate	0.001
s	summary vocabulary size	10,908
f	functions vocabulary size	70k
t	number of tokens for functions	50
c	number of tokens for summaries	13

Table 4: Settings for encoder-decoder models

4.4 Dataset Sizes

We use four dataset sizes during fine-tuning: 2.15m, 1.25m, 620k, and 170k. The 170k dataset size uses the same Java methods in the training set from `funcom-java-long` to maintain consistency with previous studies and enable experiments comparing against human-written references (all 170k Java methods in `funcom-java-long` have human-written summaries, which we remove prior to fine tuning and replace with GPT-3.5-generated summaries). We then randomly sub-sample the 2.15m dataset of GPT-3.5-generated summaries we created in Section 3 and add these to the 170k, to create 1.25m and 620k datasets. The 2.15m dataset also contains the 170k samples. Thus, all dataset sizes contain the same 170k Java methods for comparison, with additional samples added for a maximum size of 2.15m.

4.5 Hardware and Software Requirement

We use NVIDIA RTX A5000 GPU with 24GB VRAM and Intel i9-10900X CPU with 128GB RAM as the hardware to train the models. In addition, we use Pytorch 2.0.1, transformers 4.29.2, and tensorflow 2.12.0 as our software.

4.6 Evaluation Metrics

We use two metrics for evaluation: METEOR and USE. METEOR (Banerjee and Lavie, 2005) is a metric that considers the similarity between each word and word overlap for evaluation. USE (Haque et al, 2022) is a metric that encodes the reference and the predicted summary to a fixed-length vector by using universal encoder and computes the similarity scores between two summaries. Haque et al (2022); Roy et al (2021) point out that METEOR and USE are closer to human preference because these metrics assign partial credits to words instead of treating the importance of the words equally. Therefore, older metrics that only consider word overlap such as BLEU (Papineni et al, 2002) are considered as deprecated so we don't report it.

4.7 RQ2 Results

The automated metrics METEOR and USE indicate a general trend towards better matching of GPT-3.5 as the training dataset and number of model parameters increases, as shown in Tables 5 and 6. The METEOR score for the 170k dataset using the 38m parameter `jam` model is 33.88, which rises to 40.73 for the 350m parameter `jam` model and 44.8 for `starcoder`. Likewise, the 40.73 score for the 350m `jam` model rises to 44.77 as the dataset increases from 170k to 2.15m samples. The USE scores show the same pattern, with the 350m `jam` model ranging between 68.21 and 70.85 as the dataset size increases. This pattern is not surprising given that larger model and dataset sizes are widely regarded as resulting in better automated scores in many domains (Schaeffer et al, 2023), and are consistent with a view that the models are able to learn to mimic at least a portion of GPT-3.5's ability to summarize code.

A "price break" occurs favoring the use of the 350m `jam` model. Table 7 shows that `starcoder` is 28 times more expensive than the 350m `jam` model, yet reaches only 10% higher METEOR and 5% higher USE scores. The `starcoder` model was not feasible

to fine tune for the 2.15m dataset due to an estimated 14 days cost requirement, while previous metric increases were low (less than 1% difference in USE between 1.25m and 620k datasets, for example). In contrast, the 350m `jam` model cost is only slightly higher than smaller models, and can be operated on a single 16GB GPU (Su et al, 2023). Note that while it is tempting to write off training (or even inference) costs as sunk costs, in fact model expense is a key engineering detail affecting model deployment and cost/benefit analyses in industrial products (Bender et al, 2021). Overall, the 350m `jam` model provides a balance between cost and performance.

We find that the performance of some models such as 38m and 110m parameter `jam` does not increase as the data size increases. For example, in the 38m parameter `jam`, 170k data size outperforms any other datasets that are larger than 170k. Pérez-Mayos et al (2021) also observed the similar phenomenon that the performance of certain models do not always increase as the data size increases in the syntactic generation task. The possible explanation is that models learn the trivial features when the models are not large enough for the certain size of dataset (Chang and Bergen, 2023). Also, the smaller models saturate faster than the larger models (Zhai et al, 2022). This can further show that the 350m parameter `jam` is small enough to operate on 16GB GPU (Su et al, 2023), but large enough to mimic the portion of GPT-3.5's ability for code summarization.

In terms of the encoder-decoder language models, we find that the `attendgru` and `setransformer` follow the trend of reverse performance when they reach certain amount of data size. For example, `attendgru` has the best performance at 620k data

		`jam`			`starcoder`	`encoder-decoder`		
		38m	110m	350m	15.5B	attendgru	transformer	setransformer
datasets	170k	33.88	36.71	40.73	44.8	22.39	23.17	21.33
	620k	28.29	33.98	41.57	45.59	22.88	23.59	22.93
	1.25m	30.19	35.58	42.63	46.38	16.64	25.10	23.18
	2.15m	32.11	37.18	44.77	-	19.53	25.38	22.45

Table 5: METEOR scores for RQ2.

		`jam`			`starcoder`	`encoder-decoder`		
		38m	110m	350m	15.5B	attendgru	transformer	setransformer
datasets	170k	62.52	64.88	68.21	71.55	48.94	50.49	42.54
	620k	57.78	62.84	69.24	72.16	50.17	51.19	47.27
	1.25m	59.67	64.28	70.08	72.74	40.93	52.84	46.94
	2.15m	60.43	64.82	70.85	-	45.14	53.33	44.83

Table 6: USE scores for RQ2.

		`jam`			`starcoder`	`encoder-decoder`		
		38m	110m	350m	15.5B	attendgru	transformer	setransformer
datasets	170k	0.2	0.3	1.0	28	0.05	0.05	0.07
	620k	0.5	1.5	3.5	97	0.18	0.18	0.23
	1.25m	0.9	2.5	6.5	195	0.37	0.45	0.78
	2.15m	2.0	4.0	10.5	-	0.67	0.63	0.78

Table 7: Training time required in hours. The training time for `jam` and `starcoder` is the complete finetuning time. The training time for encoder-decoder models is the training time for one epoch.

size instead of 2.15m data size. This is because the models are much smaller than our pretrained models. Compared with the pretrained 38m and 110m parameter `jam` models, `attendgru` reaches the plateau at 620k data size instead of 170k data size because we train the model from scratch. Although we observe that performance of `transformer` increases as the data size increases, the improvement of `transformer` is relative small compared with 350m parameter `jam` (4.8% improvement on 350m parameter `jam` versus 1.1% on `transformer` between 2.15m and 1.25m dataset, for example). All things considered, we show that 350m parameter `jam` is the better model for the knowledge distillation for code summarization.

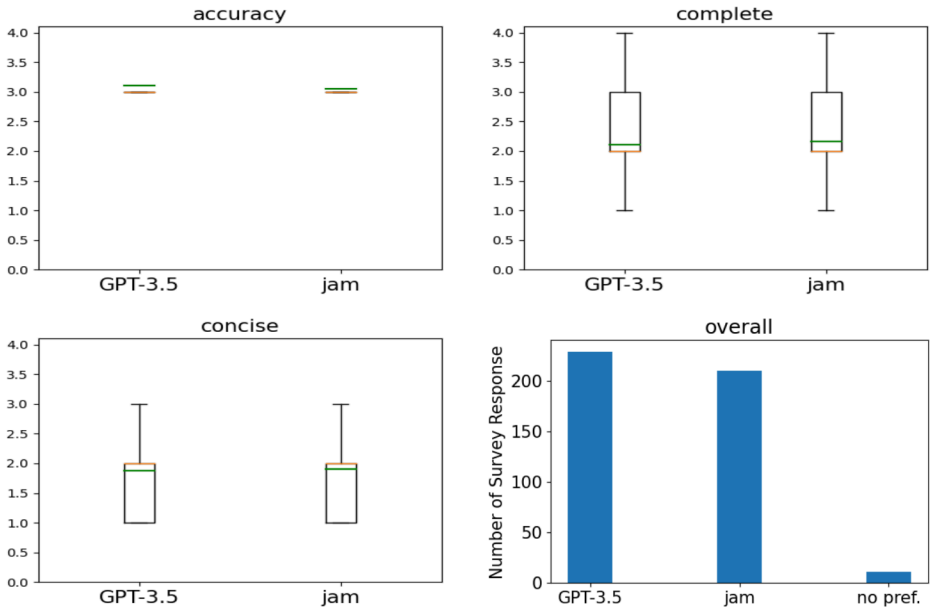
5 GPT-3.5 and Distilled Summaries Comparison

This section discusses our comparison with human experts of source code summaries generated by GPT-3.5 to summaries generated by the 350m parameter version of the `jam` model trained with the 2.15m example dataset. The purpose of this comparison is to measure how closely the `jam` model replicates GPT-3.5 for the task of code summarization. We chose to compare the 350m parameter version of the `jam` model instead of the various alternatives in Section 4 because the 350m `jam` model achieves a balance between performance and affordability. This model is within 10% performance of `starcoder` in terms of METEOR and USE scores (e.g., 44.8 versus 40.73 METEOR), while also requiring only a single 16GB GPU a relatively short amount of time. The `starcoder` model does have higher performance, but at much higher cost. Meanwhile, the 350m parameter `jam` model achieves a greater-than 10% improvement in METEOR and USE scores over the 110m parameter model, at relatively low cost in practice. The 350m parameter `jam` model is a balance between cost and performance. To compare `jam` to GPT-3.5, we ask the following RQ:

RQ3 How closely does the distilled model mimic GPT-3.5 for code summarization, as measured by human experts?

Our rationale for asking this question is that while the study in Section 4 measures how well different models mimic GPT-3.5 in terms of automated metrics, these automated metrics are known to diverge from human expert opinion at times (Novikova et al, 2017). To fill this potential gap, we also perform a study with human experts to compare the models based on human preference. Our research method to answer RQ3 is identical to how we answer RQ1, except that we change the source of the samples and recruit a new group of participants. We use the same survey pages and ask the same questions. As with RQ1, we do not mark the summaries as coming from any particular model, to avoid demand characteristic bias (Dell et al, 2012). We recruit 15 new participants via the Prolific platform using the same criteria as for RQ1. Also, we randomly select 150 functions from 450 functions that we use for the first study to answer RQ3 and recruit five different participants to evaluate the summary with the same criteria to reduce bias as in RQ1.

Figure 5 depicts the results. In short, we do not observe a statistically-significant difference between the 350m parameter `jam` model and GPT-3.5 in terms of accuracy, completeness, or conciseness. The mean value for accuracy and completeness for



	gpt-3.5		jam				
	med	mean	med	mean	Zobs	Zcrit	p
accurate	3	3.112	3	3.051	1.009	1.645	0.27
complete	2	2.107	2	2.162	0.600	1.645	0.73
concise	2	1.902	2	1.876	0.244	1.645	0.40

Fig. 5: Comparison of `jam` to GPT-3.5 followed by a statistical summary of the RQ3 results with Mann-Whitney tests showing statistical significance.

GPT-3.5 is better than `jam`, though these differences are not statistically-significant. Likewise, the mean value for conciseness is slightly better for `jam`, but is also not significant. In terms of overall preference, participants preferred summaries from GPT-3.5 52% of the time versus `jam` 46% of the time (2% undecided). Taken together, these results support a conclusion that `jam` reproduces GPT-3.5's source code summarization, though with a slight edge to GPT-3.5 in terms of overall preference.

Figure 6 shows the results for the study for the 150 functions that we randomly selected from 450 functions in the first study for RQ3. We find participants prefer GPT-3.5 56% of the time versus the 350m parameter `jam` model 43% of the time and 1% undecided. This result aligns with the first study that the overall preference edges to GPT-3.5. In terms of the convergence rate, we find only 59% of the answers converge with the first study, which is lower than the results in RQ1. This is expected because the 350m parameter `jam` model aims to reproduce the ability of the GPT-3.5 for code summarization. This can further show the 350m parameter `jam` model's ability for reproduction of GPT-3.5 for code summarization although the overall results still show that GPT-3.5 is slightly better.

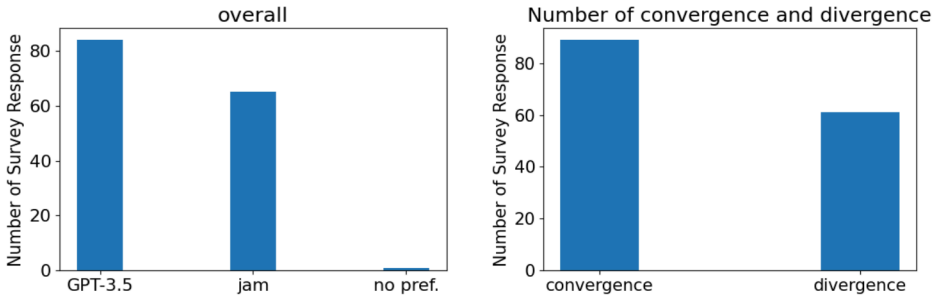


Fig. 6: Comparison of `jam` to GPT-3.5 in terms of overall preference and the number of convergence and divergence functions on the duplicate studies

preferred	model	summary
×	GPT-3.5	recycles a view for a given index, checking whether the index is within the range of items and whether the view needs to be added to the empty items or items list
	jam	receives a View object and an index, and if the index is out of range, it adds the View to the wheel and sets the emptyItems flag to true
×	GPT-3.5	reads the next byte from a buffer and returns the corresponding character, while throwing an exception if there is not enough data available
	jam	reads a character from a buffer and throws an exception if there is not enough data available
	GPT-3.5	inserts a new item into the database with the given attributes such as user ID, category, description, start/end dates, minimum bid, and name
×	jam	inserts a new item into a database table with specified parameters
	GPT-3.5	searches for a specific Physical2DObject in a Physical2DEnvironment based on its id and returns it
×	jam	retrieves a Physical2DObject from an ArrayList based on a given ID

Table 8: Examples of summaries generated by GPT-3.5 and `jam`. We show two in which each model was preferred in the human study.

Yet these results do not mean that the summaries from `jam` and GPT-3.5 are identical. Consider Table 8. Participants favored the first two samples from GPT-3.5 and the second two from `jam`. While we cannot include full code due to space limitations, these samples illustrate how the models provide broadly similar content, though with different surface realization of the text or different decisions about what information to condense. As a general observation, we note that GPT-3.5 tends to be more verbose, though whether this verbosity is perferred is unclear. For example, the second example shows how GPT-3.5 wrote "reads the next byte from a buffer and returns the corresponding character" whereas `jam` wrote "reads a character from a buffer" for the same information. The `jam` summary is more concise but less complete, but the GPT-3.5 summary was preferred. Likewise, the third sample shows how `jam` writes "with specified paramters" rather than listing them like GPT-3.5 does, though in this example `jam` is preferred.

6 Discussion/Conclusion

This paper moves the state-of-the-art forward in three key ways:

1. We present a study comparing source code summaries generated by GPT-3.5 to summaries provided as references with the code itself. We found that human readers preferred the summaries generated by GPT-3.5 by a significant margin. This result has two important implications. First, AI-based models are likely to be useful tools for augmenting or even replacing documentation written by people, which supports a vision of researchers for on-demand developer documentation (Robillard et al, 2017). Second, the research community may reconsider using human-written references as the gold standard for training and evaluating source code summarization tools. AI-based models may be superior at times.
2. We present a study in which we distilled GPT-3.5's code summarization abilities into several smaller models and thoroughly examine the exact model and data size for knowledge distillation on code summarization. Some of these models are several orders of magnitude smaller than GPT-3.5, and yet are able to achieve comparable results in a large portion of instances when measured by the automated metrics METEOR and USE. Specifically, we observe that 350m jam model with 2.15m data is a tradeoff. A key advantage to these models is that they can be run locally, with tractable costs for many consumers (a single 16GB consumer GPU, for instance). Local model execution means local custody of data. With our distilled model, it is possible to replicate much of the benefit of a large model for code summarization, without losing control of ones sensitive data and source code.
3. We present a study comparing GPT-3.5 to a distilled model (350m parameter `jam`) with human experts. This study shows how `jam` is able to reproduce GPT-3.5 for code summarization. We did not observe a statistically-significant difference between the summaries from the two models in terms of accuracy, completeness, or conciseness. We did observe a slight preference favoring GPT-3.5 in direct comparisons by participants (52% GPT-3.5, 46% `jam`, 2% undecided). Overall, these results support a conclusion that an inexpensive model such as 350m `jam` can replicate a very large model such as GPT-3.5 for the task of code summarization when provided sufficient examples.

Finally, we release all data and code for our studies and approach via an online appendix. We encourage reproducibility of our results, as well as access to the technology via our implementation in Data and Code Availability Section.

Funding

This work is supported in part by NSF CCF-2100035 and CCF-2211428. Any opinions, findings, and conclusions expressed herein are the authors and do not necessarily reflect those of the sponsors.

Contributions

Chia-Yi Su did the experiments and implemented the code for finetuning tasks. Collin McMillan wrote the manuscript, built the infrastructure and generated the dataset for experiments.

Conflict of Interest

The authors declare that they have no competing interests.

Data Availability

All of the datasets and models are in our APCL Hugginface repository, https://huggingface.co/datasets/apcl/Jam-CGPT and https://huggingface.co/apcl/Jam-CGPT

Code Availability

We release our code for experiments in our APCL Github repository, https://github.com/apcl-research/Jam-CGPT

References

Aghajani E, Nagy C, Vega-Márquez OL, et al (2019) Software documentation issues unveiled. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 1199–1210

Ahmad W, Chakraborty S, Ray B, et al (2020) A transformer-based approach for source code summarization. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, pp 4998–5007, https://doi.org/10.18653/v1/2020.acl-main.449, URL https://aclanthology.org/2020.acl-main.449

Allamanis M, Barr ET, Devanbu P, et al (2018a) A survey of machine learning for big code and naturalness. ACM Comput Surv 51(4). https://doi.org/10.1145/3212695

Allamanis M, Brockschmidt M, Khademi M (2018b) Learning to represent programs with graphs. In: International Conference on Learning Representations, URL https://openreview.net/forum?id=BJOFETxR-

Alon U, Brody S, Levy O, et al (2019a) code2seq: Generating sequences from structured representations of code. International Conference on Learning Representations URL https://openreview.net/forum?id=H1gKYo09tX

Alon U, Zilberstein M, Levy O, et al (2019b) code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages 3(POPL):1–29. https://doi.org/10.1145/3290353

Banerjee S, Lavie A (2005) Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, pp 65–72, URL https://aclanthology.org/W05-0909

Bansal A, Eberhart Z, Wu L, et al (2021a) A neural question answering system for basic questions about subroutines. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 60–71, https://doi.org/10.1109/SANER50967.2021.00015

Bansal A, Haque S, McMillan C (2021b) Project-level encoding for neural source code summarization of subroutines. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), IEEE, pp 253–264

Bender EM, Gebru T, McMillan-Major A, et al (2021) On the dangers of stochastic parrots: Can language models be too big? In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. Association for Computing Machinery, New York, NY, USA, FAccT '21, p 610–623, https://doi.org/10.1145/3442188.3445922

Brown T, Mann B, Ryder N, et al (2020) Language models are few-shot learners. In: Larochelle H, Ranzato M, Hadsell R, et al (eds) Advances in Neural Information Processing Systems, vol 33. Curran Associates, Inc., pp 1877–1901, URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Chang TA, Bergen BK (2023) Language model behavior: A comprehensive survey. arXiv preprint arXiv:230311504

Chen Z, Jiang F, Chen J, et al (2023) Phoenix: Democratizing chatgpt across languages. arXiv preprint arXiv:230410453

Danilova A, Naiakshina A, Horstmann S, et al (2021) Do you really code? designing and evaluating screening questions for online surveys with programmers. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, pp 537–548

Delgado R, Tibau XA (2019) Why cohen's kappa should be avoided as performance measure in classification. PloS one 14(9):e0222916

Dell N, Vaidyanathan V, Medhi I, et al (2012) " yours is better!" participant response bias in hci. In: Proceedings of the sigchi conference on human factors in computing systems, pp 1321–1330, https://doi.org/10.1145/2207676.2208589

Derner E, Batistič K (2023) Beyond the safeguards: Exploring the security risks of chatgpt. arXiv preprint arXiv:230508005

Donker D, Hasman A, Van Geijn H (1993) Interpretation of low kappa values. International journal of bio-medical computing 33(1):55–64

Forward A, Lethbridge TC (2002) The relevance of software documentation, tools and technologies: A survey. In: Proceedings of the 2002 ACM Symposium on Document Engineering. Association for Computing Machinery, New York, NY, USA, DocEng '02, p 26–33, https://doi.org/10.1145/585058.585065

Fowkes J, Chanthirasegaran P, Ranca R, et al (2017) Autofolding for source code summarization. IEEE Transactions on Software Engineering 43(12):1095–1109. https://doi.org/10.1109/TSE.2017.2664836

Gao S, Chen C, Xing Z, et al (2019) A neural model for method name generation from functional description. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, pp 414–421, https://doi.org/10.1109/SANER.2019.8667994

Ghorbani A, Cassee N, Robinson D, et al (2023) Autonomy is an acquired taste: Exploring developer preferences for github bots. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, pp 1405–1417

Github (2022) Co-pilot. URL https://github.com/features/copilot

Gou J, Yu B, Maybank SJ, et al (2021) Knowledge distillation: A survey. International Journal of Computer Vision 129:1789–1819

Gudibande A, Wallace E, Snell C, et al (2023) The false promise of imitating proprietary llms. arXiv preprint arXiv:230515717

Haiduc S, Aponte J, Moreno L, et al (2010) On the use of automated text summarization techniques for summarizing source code. In: 2010 17th Working Conference on Reverse Engineering, IEEE, pp 35–44, https://doi.org/10.1109/WCRE.2010.13

Haldar R, Wu L, Xiong J, et al (2020) A multi-perspective architecture for semantic code search. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, Online, pp 8563–8568, https://doi.org/10.18653/v1/2020.acl-main.758, URL https://aclanthology.org/2020.acl-main.758

Haque S, LeClair A, Wu L, et al (2020) Improved automatic summarization of subroutines via attention to file context. International Conference on Mining Software Repositories https://doi.org/10.1145/3379597.3387449

Haque S, Bansal A, Wu L, et al (2021) Action word prediction for neural source code summarization. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp 330–341, https://doi.org/10.1109/SANER50967.2021.00038

Haque S, Eberhart Z, Bansal A, et al (2022) Semantic similarity metrics for evaluating source code summarization. In: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, pp 36–47, https://doi.org/10.1145/3524610.3527909

Hellendoorn VJ, Sawant AA (2021) The growing cost of deep learning for source code. Commun ACM 65(1):31–33. https://doi.org/10.1145/3501261

Hsieh CY, Li CL, Yeh CK, et al (2023) Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. arXiv preprint arXiv:230502301

Hu X, Li G, Xia X, et al (2018a) Deep code comment generation. In: Proceedings of the 26th Conference on Program Comprehension. Association for Computing Machinery, New York, NY, USA, ICPC '18, p 200–210, https://doi.org/10.1145/3196321.3196334

Hu X, Li G, Xia X, et al (2018b) Summarizing source code with transferred api knowledge. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence. AAAI Press, IJCAI'18, p 2269–2275

Israel GD (1992) Determining sample size

Iyer S, Konstas I, Cheung A, et al (2016) Summarizing source code using a neural attention model. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Berlin, Germany, pp 2073–2083, https://doi.org/10.18653/v1/P16-1195, URL https://aclanthology.org/P16-1195

Jiang S, Armaly A, McMillan C (2017) Automatically generating commit messages from diffs using neural machine translation. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, ASE '17, p 135–146

LeClair A, McMillan C (2019) Recommendations for datasets for source code summarization. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp 3931–3937

LeClair A, Jiang S, McMillan C (2019) A neural model for generating natural language summaries of program subroutines. In: Proceedings of the 41st International Conference on Software Engineering, IEEE Press, pp 795–806, https://doi.org/10.1109/ICSE.2019.00087

Li J, Gui L, Zhou Y, et al (2023a) Distilling chatgpt for explainable automated student answer assessment. arXiv preprint arXiv:230512962

Li R, Allal LB, Zi Y, et al (2023b) Starcoder: may the source be with you! arXiv preprint arXiv:230506161

Li Z, Wu Y, Peng B, et al (2023c) Setransformer: A transformer-based code semantic parser for code comment generation. IEEE Transactions on Reliability 72(1):258–273. https://doi.org/10.1109/TR.2022.3154773

Liang Y, Zhu KQ (2018) Automatic generation of text descriptive comments for code blocks. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence. AAAI Press, AAAI'18/IAAI'18/EAAI'18

Liu S, Chen Y, Xie X, et al (2021) Retrieval-augmented generation for code summarization via hybrid GNN. In: International Conference on Learning Representations, URL https://openreview.net/forum?id=zv-typ1gPxA

Loyola P, Marrese-Taylor E, Matsuo Y (2017) A neural architecture for generating natural language descriptions from source code changes. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Association for Computational Linguistics, Vancouver, Canada, pp 287–292, https://doi.org/10.18653/v1/P17-2045, URL https://aclanthology.org/P17-2045

Lu Y, Zhao Z, Li G, et al (2019) Learning to generate comments for api-based code snippets. In: Li Z, Jiang H, Li G, et al (eds) Software Engineering and Methodology for Emerging Domains. Springer Singapore, Singapore, pp 3–14

Ma W, Liu S, Wang W, et al (2023) The scope of chatgpt in software engineering: A thorough investigation. arXiv preprint arXiv:230512138

McBurney PW, Liu C, McMillan C (2016) Automated feature discovery via sentence selection and source code summarization. Journal of Software: Evolution and Process 28(2):120–145. https://doi.org/10.1002/smr.1768

Nie P, Rai R, Li JJ, et al (2019) A framework for writing trigger-action todo comments in executable format. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Association for Computing Machinery, New York, NY, USA, ESEC/FSE 2019, p 385–396, https://doi.org/10.1145/3338906.3338965

Novikova J, Dušek O, Cercas Curry A, et al (2017) Why we need new evaluation metrics for NLG. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Copenhagen, Denmark, pp 2241–2252, https://doi.org/10.18653/v1/D17-1238, URL https://aclanthology.org/D17-1238

OpenAI (2022) Chatgpt. URL https://openai.com/blog/chatgpt

Papineni K, Roukos S, Ward T, et al (2002) Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting on association for computational linguistics, Association for Computational Linguistics, pp 311–318, https://doi.org/10.3115/1073083.1073135

Pérez-Mayos L, Ballesteros M, Wanner L (2021) How much pretraining data do language models need to learn syntax? arXiv preprint arXiv:210903160

Robillard MP, Marcus A, Treude C, et al (2017) On-demand developer documentation. In: 2017 IEEE International conference on software maintenance and evolution (ICSME), IEEE, pp 479–483, https://doi.org/10.1109/ICSME.2017.17

Rodeghero P, Jiang S, Armaly A, et al (2017) Detecting user story information in developer-client conversations to generate extractive summaries. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp 49–59, https://doi.org/10.1109/ICSE.2017.13

Roy D, Fakhoury S, Arnaoudova V (2021) Reassessing automatic evaluation metrics for code summarization tasks. In: Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), https://doi.org/10.1145/3468264.3468588

Schaeffer R, Miranda B, Koyejo S (2023) Are emergent abilities of large language models a mirage? arXiv preprint arXiv:230415004

Shi L, Mu F, Chen X, et al (2022) Are we building on the rock? on the importance of data preprocessing for code summarization. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Association for Computing Machinery, ESEC/FSE 2022, p 107–119

Sievertsen HH, Gino F, Piovesan M (2016) Cognitive fatigue influences students' performance on standardized tests. Proceedings of the National Academy of Sciences 113(10):2621–2624. https://doi.org/10.1073/pnas.1516947113

Sridhara G, Hill E, Muppaneni D, et al (2010) Towards automatically generating summary comments for java methods. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, ACM, pp 43–52, https://doi.org/10.1145/1858996.1859006

Su CY, Bansal A, Jain V, et al (2023) A language model of java methods with train/test deduplication. In: 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Demonstrations (FSE'23 Demos)

Sun W, Fang C, You Y, et al (2023) Automatic code summarization via chatgpt: How far are we? arXiv preprint arXiv:230512865

Tang Y, da Costa AAB, Zhang J, et al (2023) Domain knowledge distillation from large language model: An empirical study in the autonomous driving domain. arXiv preprint arXiv:230711769

Wan Y, Zhao Z, Yang M, et al (2018) Improving automatic source code summarization via deep reinforcement learning. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Association for Computing Machinery, New York, NY, USA, ASE '18, p 397–407, https://doi.org/10.1145/3238147.3238206, URL https://doi.org/10.1145/3238147.3238206

Wang L, Yoon KJ (2021) Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. IEEE transactions on pattern analysis and machine intelligence 44(6):3048–3068

Wang Y, Wang W, Joty S, et al (2021) CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Moens MF, Huang X, Specia L, et al (eds) Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, pp 8696–8708, https://doi.org/10.18653/v1/2021.emnlp-main.685, URL https://aclanthology.org/2021.emnlp-main.685

Xu C, Xu Y, Wang S, et al (2023) Small models are valuable plug-ins for large language models. arXiv preprint arXiv:230508848

Yu Y, Zhuang Y, Zhang J, et al (2023) Large language model as attributed training data generator: A tale of diversity and bias. arXiv preprint arXiv:230615895

Zagoruyko S, Komodakis N (2016) Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In: International Conference on Learning Representations

Zhai X, Kolesnikov A, Houlsby N, et al (2022) Scaling vision transformers. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 12104–12113

Zhang R, Han J, Zhou A, et al (2023) Llama-adapter: Efficient fine-tuning of language models with zero-init attention. Parameters 7:13B

Zügner D, Kirschstein T, Catasta M, et al (2021) Language-agnostic representation learning of source code from structure and context. In: International Conference on Learning Representations, URL https://openreview.net/forum?id=Xh5eMZVONGF