A Survey of Neural Code Intelligence: Paradigms, Advances and Beyond

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, Qipeng Guo, Xipeng Qiu, Pengcheng Yin, Xiaoli Li, *Fellow, IEEE,* Fei Yuan, Lingpeng Kong, Xiang Li, and Zhiyong Wu

Abstract—Neural Code Intelligence – leveraging deep learning to understand, generate, and optimize code – holds immense potential for transformative impacts on the whole society. Bridging the gap between Natural Language and Programming Language, this domain has drawn significant attention from researchers in both research communities over the past few years. This survey presents a systematic and chronological review of the advancements in code intelligence, encompassing over 50 representative models and their variants, more than 20 categories of tasks, and an extensive coverage of over 680 related works. We follow the historical progression to trace the paradigm shifts across different research phases (*e.g.*, from modeling code with recurrent neural networks to the era of Large Language Models). Concurrently, we highlight the major technical transitions in models, tasks, and evaluations spanning through different stages. For applications, we also observe a co-evolving shift. It spans from initial endeavors to tackling specific scenarios, through exploring a diverse array of tasks during its rapid expansion, to currently focusing on tackling increasingly complex and varied real-world challenges. Building on our examination of the developmental trajectories, we further investigate the emerging synergies between code intelligence and broader machine intelligence, uncovering new cross-domain opportunities and illustrating the substantial influence of code intelligence across various domains. Finally, we delve into both the opportunities and challenges associated with this field, alongside elucidating our insights on the most promising research directions. An ongoing, dynamically updated project and resources associated with this survey have been released at https://github.com/QiushiSun/NCISurvey.

Index Terms—Code Intelligence; Natural Language Processing; Language Models; Software Engineering

1 Introduction

CODE is one of the elegant languages created by humans, which replaces the diverse forms of natural language (NL) through a high degree of abstraction [1]. As a conduit between humans and machines, it is ultimately transformed into specific programs[1] that substitute human effort in accomplishing various tasks, characterized by advantages such as precision, logic, modularity, and excitability.

The fusion of rapidly advancing deep learning techniques with the availability of "Big Code" [2, 3] has led to the emergence of neural code intelligence. This domain, which applies neural approaches to understand, generate, and manipulate code, has garnered significant attention from the research community. Figure 1 illustrates the cumulative publication statistics[2] over recent years, showcasing the growing interest and significant efforts being dedicated to this area. Notably, this domain transcends disciplinary boundaries, spanning Natural Language Processing (NLP) [4], Software Engineering (SE) [5], Robotics [6], and beyond. Moreover, the unique duality of code, which combines human-readable semantics with executability, establishes research in this area as a critical bridge between artificial intelligence and the real world, laying a corner-

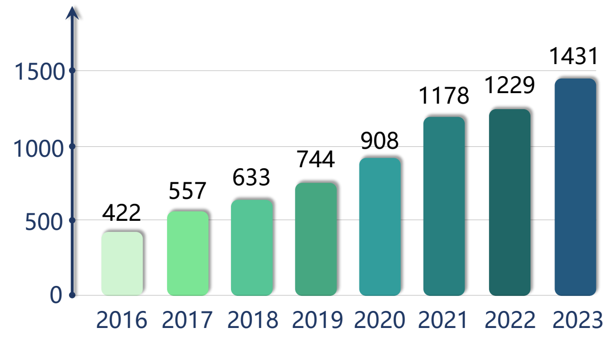

Fig. 1: Cumulative number of publications/preprints related to neural code intelligence (from arXiv). Over the past few years, the number of articles has been steadily increasing.

- *Our project is available at: https://github.com/QiushiSun/NCISurvey*
- *Qiushi Sun (qiushisun@u.nus.edu), Fangzhi Xu, Kanzhi Cheng, Chang Ma, Shuai Yuan, Fei Yuan, and Zhiyong Wu (wuzhiyong@pjlab.org.cn) are with Shanghai AI Laboratory, Shanghai, China.*
- *Zhirui Chen, Jianing Wang, Chengcheng Han, and Xiang Li (xiangli@dase.ecnu.edu.cn) are with the School of Data Science and Engineering, East China Normal University, Shanghai, China.*
- *Zhangyue Yin, Qipeng Guo, and Xipeng Qiu are with the School of Computer Science, Fudan University, Shanghai, China.*
- *Renyu Zhu is with NetEase Fuxi AI Lab, Zhejiang, China.*
- *Pengcheng Yin is with Google DeepMind, Mountain View, CA, USA.*
- *Xiaoli Li is with the Institute for Infocomm Research (I^{2}R), Agency for Science, Technology and Research (A*STAR), Singapore, and also with the School of Computer Science and Engineering at Nanyang Technological University, Singapore.*
- *Lingpeng Kong is with the Department of Computer Science, The University of Hong Kong, Hong Kong, China.*

Version 1.1

1. We use *code* and *program* interchangeably in this paper.

2. The statistics are derived by querying a set of specific keywords (*e.g.*, code representation, code generation, code intelligence) through an exact match search in the titles or abstracts of documents.

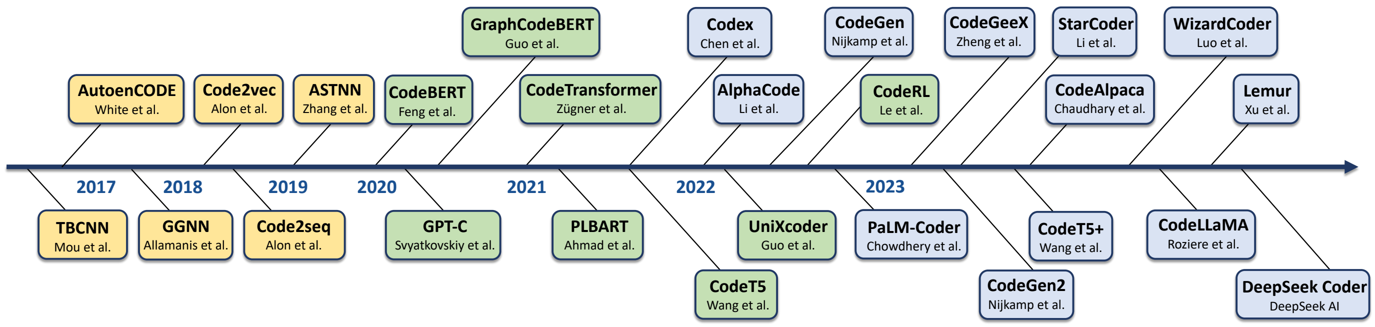

Fig. 2: A chronological overview of representative works in neural code intelligence over recent years. Works are differentiated by background colors to represent distinct evolutionary phases: 🟨 represents milestones of neural language models using code structures, 🟩 denotes code pre-trained models with typical architectures, and 🟦 signifies some influential large language models for code. The timeline is established mainly according to the release date of the paper or model.

stone on the path toward artificial general intelligence [7]. At the heart of these studies lies the foundational concept of "Software Naturalness" [8], which posits that programming language (PL), much like human languages, is characterized by predictable and repetitive patterns that can be effectively modeled. From a macroscopic perspective, the progression of techniques for processing these artificial languages has largely mirrored the evolution observed in NLP [9].

Specifically, after a brief initial period characterized by statistical [10, 11, 12] and probabilistic [13] modeling, the learning paradigms of neural code intelligence has transitioned from the earliest word embedding techniques to Large Language Models (LLMs), broadly categorized into three main phases of evolution:

• *Neural Language Models for Code.* This era witnessed the early yet fruitful efforts of applying deep learning to process code. The methods designed during this period primarily relied on well-developed recurrent [14] or convolutional [15] structures to model code. Notably, they not only leverage the textual information of the code but also intricately incorporate structural information extracted from code structures like Abstract Syntax Trees (ASTs) into the modeling process [16, 17], aligned closely with the principles of neural semantic parsing [18, 19, 20].

Meanwhile, as code snippets can be represented as continuous vectors, the techniques evolved during this period were also known as code embeddings [21]. The most representative techniques, code2vec [22] and code2seq [23], captured the semantic and structural information of code by embedding paths from ASTs into a vector space, enabling the application of neural methods to diverse scenarios.

• *Code Pre-trained Models (CodePTMs).* Pre-trained language models [24, 25] with multi-layer Transformer architecture [26] have established a "pre-train" and "fine-tune" learning paradigm [27]. Following this, various studies have emerged on how to build CodePTMs. These models, exemplified by CodeBERT [28], CodeT5 [29], and PLBART [30], have long dominated the mainstream approaches in code intelligence and initiated a surge of research interest – a sharp increase in arXiv papers can be observed in Figure 1 after the release of CodeBERT in 2020. During the pre-training phase, they learn general-purpose context-aware representations from massive GitHub code data, as well as their structural information of code (*e.g.*, ASTs). Subsequently, they are fine-tuned on task-specific code data or NL-PL pairs, significantly raising the performance of various code-related tasks. This approach marks a shift from previous learning paradigms, where CodePTMs no longer require individual modeling for each task, but adapt to different scenarios by fine-tuning on relatively smaller labeled datasets.

• *Large Language Models for Code (CodeLLMs).* Recent studies have indicated that scaling language models through increasing their parameters or the volume of training data [31, 32] consistently results in an enhancement of the model's capacity to perform effectively on downstream tasks. Following the success of general LLMs such as GPT-3 [33] and PaLM [34] in both academia and industry [35], models like Codex [36] have sparked a new wave of research in code intelligence – a notable increase of related papers can be observed in Figure 1 after the debut of *ChatGPT*[3] in late 2022. This phase has also seen a shift in the learning paradigm from task-specific fine-tuning to prompting [37] and in-context learning [38], as well as expanding the application of code intelligence from solely code-related tasks to a broader array of real-world scenarios.

Aware that the development of code intelligence is significantly reflected in the evolution of language models designed for code, we present in Figure 2 a chronological summary of representative works that outline the developmental trajectory of neural code intelligence. It provides a framework for this paper by outlining an overview of the technical advances within the domain. Building upon this timeline, we embrace a new perspective characterized by the *paradigm shifts* in models, applications, evaluations, and beyond, to explore these exciting advancements in code intelligence. We thoroughly investigate the literature and distill the key findings, techniques, and interconnections between research across various epochs. Moreover, we broaden our scope to explore the integration of other domains with code intelligence, discussing how code generation assists in machine reasoning, how code training enhances models' mathematics capabilities, and how code serves as a medium to provide new approaches for solving typical NLP tasks. Furthermore, we explore a wide array

3. https://openai.com/blog/chatgpt

of real-world cross-domain applications, spanning coding assistants, data science, autonomous agents, and AI4science. A GitHub project associated with this survey is actively maintained at https://github.com/QiushiSun/NCISurvey, which contains all the resources used to construct this paper, as well as comprehensive reading lists to assist readers in further exploration. We hope that this will facilitate the continued development of the field.

The following parts of this survey are organized as follows. Section 2 begins by reviewing the preliminaries. We then examine classic yet crucial methods for processing code using neural language models, as well as an introduction to quintessential code-related tasks. Section 3 delves into the evolution of code intelligence within the pre-train and fine-tune paradigm, offering a comprehensive review and discussion of the techniques of this era and their implications for future research. Section 4 explores the research advancements in the era of LLMs, alongside a review of progress in NL2Code, and conducts thorough discussions of representative models and benchmarks. Section 5 investigates the synergies between code intelligence and other domains of machine intelligence. Section 6 is on the application of code intelligence in real-world scenarios, demonstrating its practical utility. Section 7 engages in a dual-faceted discussion of open issues, both from the perspectives of model architecture and practical application, and shares what we believe are worthy of future research directions. Finally, Section 8 summarizes the insights and findings of the paper.

2 The Spark of Code Intelligence

In recent years, the vast availability of source code from public repositories has significantly boosted the application of deep learning techniques to source codes [3]. Under the concept of "software naturalness" [8], neural language models designed for processing text can be naturally applied to code. Combined with the evolving demands of software engineering for automated code processing, neural code intelligence has experienced a prosperous development.

By conceptualizing code snippets as language sequences, sequential neural architectures, such as LSTM [14], are naturally adaptable for code understanding and generation [39, 40]. However, it is imperative to recognize that, unlike natural language sentences, programs contain explicit and complicated structures, which introduces more opportunities and possibilities for modeling code. Subsequently, code embeddings have emerged, which can be defined as numerical sequences representing the inherent concepts found in codes. This development has had a pioneering and long-lasting impact on future research in code intelligence

In this section, we first provide readers with preliminaries regarding code structure, followed by a review of some classic methods of modeling based on it. Subsequently, we introduce a series of classic, significant, and continuously explored code-related tasks.

2.1 Code Features Through Structural Views

Viewing source code merely as a token sequence overlooks their inherent structures, a characteristics can greatly enhance the model's ability to comprehend code. To illustrate this, let's consider a straightforward example. Consider the expression `s = min_value + max_value`, `s` is evidently derived from the maximum and minimum values. However, since programmers do not always adhere to naming conventions, it is challenging for models to grasp the semantics of `s` solely from its variable name. Nevertheless, by leveraging the dependency relation between variables, it becomes possible to facilitate the comprehension of the semantics of the `s` and to predict program properties [41]. Structural information represented by such dependency relations plays a crucial role in modeling code [42, 43]. Therefore, in this part, we will briefly cover three typical carriers of structural information, providing readers with some background knowledge.

• **Abstract Syntax Tree.** AST stands as a quintessential intermediate representation during code compilation, where a program is parsed into a tree structure of operations and their operands. Serving as a syntactic-level structure, it encapsulates both the syntax and structural information of a program, while its components also embody distinct semantics [44]. It can be obtained by applying parsers (*e.g.*, Tree-sitter[4], pycparser[5] and javalang[6]) on source codes.

• **Data flow.** Unlike syntactic-level code structures like AST, Data flow (Graph) represents a semantic-level structure within code. Its nodes represent variables, and the edges reflect the relationships and origins among these variables. It can be extracted from AST and characterized by reduced complexity and does not entail a deep hierarchy, resulting in relatively lower costs for modeling and analysis [45].

• **Control flow.** Contrasting with data flow's semantics, Control Flow (Graph) provides a structural view of code executive information. Here, nodes represent executable blocks, and edges indicate control transitions between them. This emphasizes the sequence and potential paths of program execution rather than variable interactions [46]. Control Flow is key for understanding program dynamics, and offering insights into program logic. It can be constructed through the use of static analyzers [47].

Building on the above code features, in processing codes with neural methods, one can consider not only the plain text of the source code but also leverage code structural information. This process can be typically divided into three main strategies: (1) Directly Encoding AST: A representative method is TBCNN [16], which utilizes tree-based convolution kernels on ASTs to capture information from subtrees. The features of subtrees will be aggregated through pooling to formulate the embedding of the program. Following this, subsequent work has increasingly integrated ASTs with convolution to capture local code features [48, 49]. (2) Utilizing AST Paths: This approach is exemplified by code2vec [22] and code2seq [23]. Code2vec integrates the representations of AST leaf nodes and aggregates their path representation to build combined context vectors. Following this idea, code2seq extracts more fine-grained information from the AST paths and leverages an LSTM to encode the entire path to suit generation tasks. (3) Transforming AST, represented by: AutoenCODE [50] converts ASTs into bi-

4. https://github.com/tree-sitter
5. https://github.com/eliben/pycparser
6. https://github.com/c2nes/javalang

nary trees and utilizes autoencoder to learn code embedding from it. GGNN [51] introduces additional edges to explicitly represent data/control flows and employs graph neural networks to learn nodes' representations. To address long-term dependency issues, ASTNN [17] breaks each AST into a sequence of statement subtrees and encodes them into vectors by capturing both the lexical and syntactical knowledge of the statements. The approach has been further extended for industrial applications [52]. InferCode [53] employs self-supervised learning by exploiting the structural similarities within code to automatically generate labels for training.

Moreover, beyond utilizing these features in their original forms, researchers have adapted them to apply various deep learning approaches. Wang et al. [54], Wang and Li [55] initially augment AST with explicit control and data flow edges to facilitate the application of graph algorithms [56, 57, 58]. Later, issues related to the low connectivity of ASTs and the out-of-vocabulary problem [59] during modeling are identified [60]. To mitigate these issues, researchers connecting adjacent leaf nodes, which aids in the graph partitioning [61] and analysis [62].

2.2 Overview of Core Tasks in Code Intelligence

This part provides an overview of the most important tasks in code intelligence and the challenges they face, categorized based on the form of their inputs and outputs.

2.2.1 Code-Code Tasks

Code-code tasks refer to a series of tasks that involve operations on source code with the aim of understanding, generating, or transforming code.

Clone Detection. Clone detection is widely studied in SE research [63, 17, 54], which predicts whether two code snippets are clones of each other, and can be conceptualized as a binary sentence classification task. Code clones refer to pairs of code snippets that display notable similarities, occurring within or across different software systems [64]. Programmers often create clones by reusing code through copy and paste. While cloning can offer advantages, such as accelerated software development, it presents significant drawbacks. When buggy code is cloned, the bug is duplicated throughout the system, exacerbating the complexity of debugging and maintenance [65]. Furthermore, clones may introduce new bugs if updates to a code fragment are not uniformly applied to its clones. Such practices can adversely impact software by unnecessarily inflating the system's size and consequently increasing the expenses related to re-engineering [66]. Beyond finding suitable matching algorithms or metric [67], The primary challenge in developing automated approaches for clone detection lies in equipping the detector with the ability to fully comprehend syntactic [68, 69, 70] or semantic [71, 72, 73] similarities, thereby minimizing the risk of false positives [74].

Defect Detection. The incidence of source code defects and vulnerabilities has been increasing rapidly, as evidenced by public reports through CVE (Common Vulnerabilities & Exposures), as well as identified within proprietary code bases. This trend poses a crucial yet complex challenge in security. In response to this, defect (vulnerability) detection has emerged as a solution, aiming to liberate human programmers from the extensive demands of manual code inspection [75, 76, 77, 78, 79]. The task can be formalized as binary classification, *i.e.*, learning to determine whether a given code snippet contains a defect or not. As a non-generative task, it shares similar challenges with clone detection, and further, the calibration of models [80, 81] plays a vital role in ensuring reliability as well.

Code Repair. Writing codes often involves errors, a common experience for programmers. Often, these errors are minor, necessitating only limited modifications to the original program. Such errors can interrupt the workflow of experienced developers and may pose significant challenges to beginners while localizing and rectifying them is known to be effort-prone and time-consuming. Code repair aims to refine the code by automatically localizing [82] and fixing these bugs, which can be modeled as a seq2seq task [83, 51, 84, 85, 86, 87]. By integrating code repair with defect detection, it becomes possible to streamline the processes of identifying issues and implementing fixes [88].

Code Completion. Code completion is one of the most common application scenarios for coding assistants like Copilot[7], and its usage is bifurcated into two subcategories: token-level completion and line-level completion. The former involves predicting a single code token, while the latter entails completing an entire, yet unfinished line. The objective of the task is to predict subsequent token(s) within a given code context [89, 90, 91, 92, 93]. It can also be viewed as a seq2seq task, but the target needs to be a continuation of the input. With the evolution of code intelligence, code completion has also begun to encompass infilling tasks [94, 95], which entails not only left-to-right completion but also filling in code before or in the middle of a given context.

Code Translation. Code translation, also known as transpilation, involves translating a code snippet from one PL to another. It has many use cases, such as modernizing artifacts [96] implemented in PLs like COBOL or Python 2, and migrating legacy software in proprietary PLs to applications written in general-purpose PLs [97]. Over the past decades, the paradigm of code translation has undergone a significant transformation, shifting from labor-intensive rewriting methods to more efficient and reliable automated solutions. While it can also be described as a seq2seq task, it presents more difficulty compared to previous tasks. The greatest challenges include (1) the need to faithfully preserve the original functionality, and (2) the requirement to generate syntactically correct code without introducing bugs [98]. Existing research includes strategies that utilize annotated PL pairs and their syntactic structures for training [99, 100], as well as unsupervised methods that learn from monolingual source code without parallel data [101, 102, 103]. Additionally, the scope of this area also includes pseudocode-to-code translation [104, 105]. In comparison to NL machine translation, the functional correctness of all translated code is more critical than its similarity [106, 107] to the reference.

2.2.2 Code-Text Task

Code-text tasks refer to the challenge of generating natural language from source code.

7. https://github.com/features/copilot

TABLE 1: Representative benchmarks for different types of code-related downstream tasks, including the number of programming languages they cover and brief descriptions. Complete benchmarks are listed in Table 6, Table 7 and Table 8.

Task	Dataset	Date	# PLs.	Description
Clone Detection	POJ-104 [16] [link]	2014	2	a program classification dataset of 52K C/C++ programs
	BigCloneBench [108] [link]	2015	1	a clone detection dataset of eight million Java validated clones
	CLCDSA [109] [link]	2019	3	a cross-language clone dataset of more than 78K solutions
Defect Detection	Devign [78] [link]	2019	1	a dataset of vulnerable C functions
	CrossVul [110] [link]	2021	>40	a dataset of 13K/27K (vulnerable/non-vulnerable) files
	DiverseVul [111] [link]	2023	2	a dataset of 18K/330K (vulnerable/non-vulnerable) functions
Code Repair	Defects4J [link]	2014	1	a database of real Java bugs
	DeepFix [83] [link]	2017	1	a dataset of 7K erroneous C programs for 93 programming tasks
	QuixBugs [112] [link]	2017	2	a multilingual benchmark of similar buggy programs
Code Search	CodeSearchNet [113] [link]	2019	6	a dataset of 6M functions and natural language queries
	AdvTest [114] [link]	2021	1	a Python code search dataset filtered from CodeSearchNet
	WebQueryTest [114] [link]	2021	1	a testing set of Python code search of 1K query-code pairs
Code Translation	CodeTrans [114] [link]	2021	2	a C#/Java dataset collected from several repos
	CoST [115] [link]	2022	7	a dataset containing parallel data from 7 programming languages
	CodeTransOcean [116] [link]	2023	45	a large-scale comprehensive benchmark for code translation
Code Completion	GitHub Java Corpus [2] [link]	2013	1	a giga-token corpus of Java code from a wide variety of domains
	Py150 [117] [link]	2016	1	a corpus of Python programs from GitHub
	LCC [118] [link]	2023	3	a benchmark of code completion with long code context
Code Summarization	CODE-NN [119] [link]	2016	2	a dataset of (title, query) pairs from StackOverflow
	TL-CodeSum [120] [link]	2018	1	a dataset containing 69K pairs of (API sequence, code, summary)
	CodeSearchNet [113] [link]	2019	6	a dataset of 6M functions and natural language queries
GitHub	CommitGen [121] [link]	2017	4	a multilingual dataset collected from open source projects
	CommitBERT [122] [link]	2021	6	a multilingual dataset of code modification and commit messages
	SWE-bench [123] [link]	2023	1	a benchmark of 2K SE problems and corresponding PRs

Code Summarization. Code summarization represents a prominent task in the field of code intelligence, which entails generating concise and descriptive comments for codes, derived from analyzing its semantics [124]. It is vital for updating and maintaining software systems [125], particularly those with collaboration among multiple developers. Code summarization can be modeled in a seq2seq format, aiming to take a code snippet (and its structure) as input and produce an NL description [119, 23, 126, 127, 128] or the function/method's name as output [129, 130, 131, 132]. Beyond directly synthesizing summaries, strategies include retrieving keywords from the source code [133, 134] or employing clone detection to find comments from similar code snippets [135]. Additionally, leveraging the API knowledge can further enhance the relevance of generated content [120].

Commit Message Generation. Developers may frequently edit their code for bug fixing, adding new features, etc. Version control systems like Git often track these edits, which utilizes commit to document the changes. When code is updated frequently, manually writing commit messages becomes a laborious task. Fortunately, code embeddings can also be employed to represent these edits [136]. Commit message generation is an emerging task aimed at automating the creation of commit messages for code changes. It involves taking two versions of the code, before and after the edits, as input and generating summaries that describe their differences [137, 138]. In practice, the methods employed for commit message generation span a range of techniques, including the use of predefined rules or templates [139], leveraging commit messages from similar code changes [140], employing seq2seq modeling [141, 142, 122], and incorporating retrieval-augmented approaches [143].

2.2.3 Text-Code Tasks

Text-code tasks involve finding or generating executable source code aligned with NL descriptions.

Code Retrieval. To enhance developers' productivity, seeking ready-made solutions that closely match their requirements serves as a shortcut. The objective of Code retrieval, also known as NL code search is to identify and retrieve codes that are functionally relevant, in response to NL queries [113, 144, 145]. A common practice involves using specially designed metrics to measure the similarity between the contextual embeddings of the given query and the candidate code snippets [146, 147]. Additionally, there is a parallel task to code retrieval known as code search [148, 149], where the key difference lies in the query: here, the query is also a code snippet. This task can be viewed as searching for clones within a candidate pool, allowing developers to find code snippets that perform similar functions or have similar implementations based on code-based queries. The code obtained can serve as a reference for generating more complex yet related code [150].

Code Generation. Code generation, also known as program synthesis, broadly refers to the use of NL to generate code (NL2Code). This long-standing task aims to lower the barriers associated with coding, streamline some routine tasks through automation, and empower non-programmers to obtain solutions tailored to their intentions [151]. Beyond merely treating it as another form of text generation, early research primarily relied on the guidance of code syntax to generate small code snippets for specific scenarios, and view it as a semantic parsing task [152, 153, 154]. Over time, this focus gradually shifted towards general-purpose code generation [155, 156, 150, 157] for a single program-

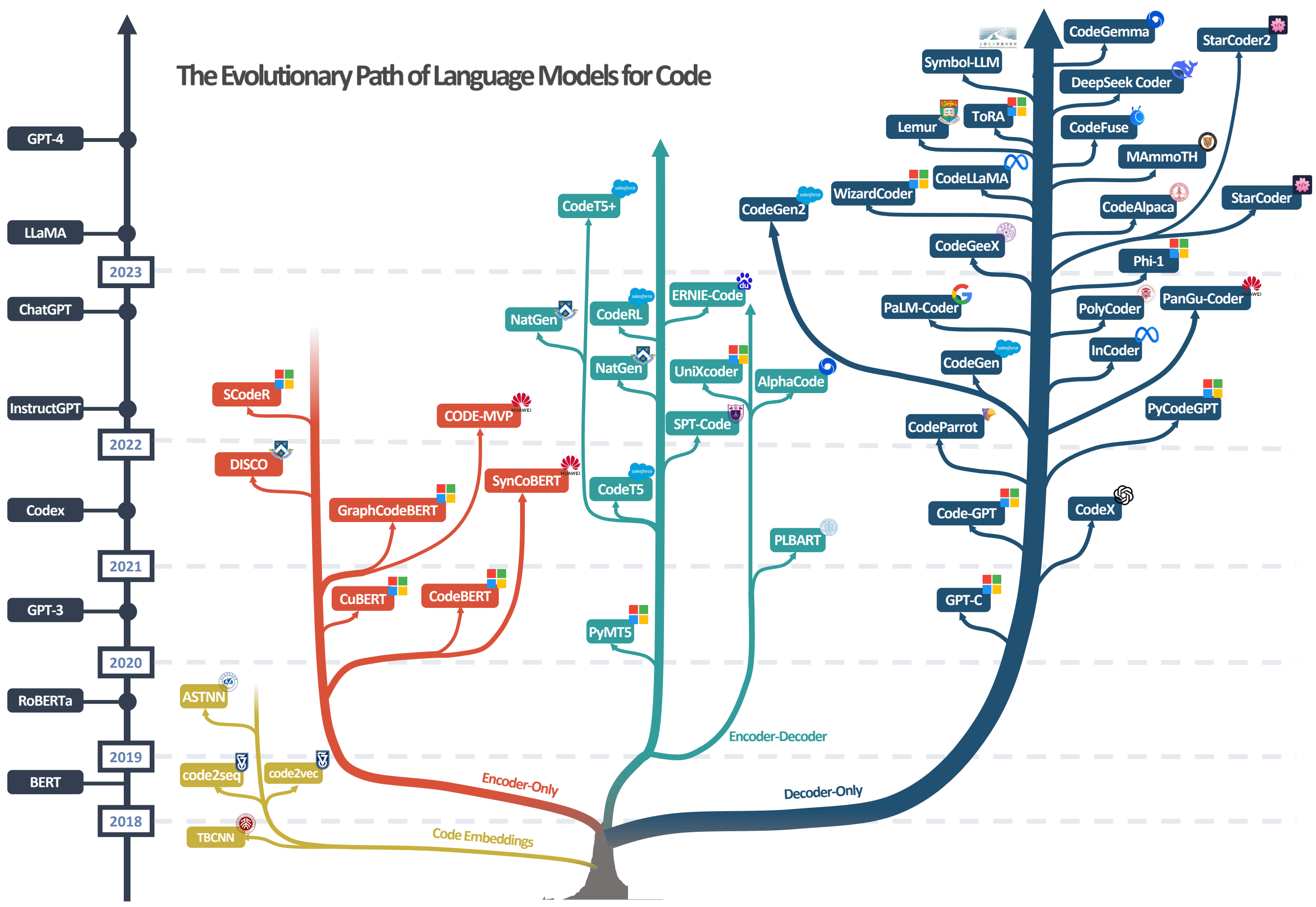

Fig. 3: The trajectory of neural code intelligence's evolution is encapsulated through the development of language models for code. This is delineated by four principal branches, each representing a distinct category of models. The first branch showcases models based on code embedding techniques, while the subsequent three branches feature Transformer-based models, each exemplifying unique architectures: Encoder-only, Encoder-Decoder, and Decoder-only. Models on the same sub-branch have closer relationships. Additionally, the vertical axis chronicles the timeline of these models' release dates, paralleled by some seminal NLP models. The details of creating this figure are listed in Appendix A.2.

ming language. Examples such as Hearthstone [158], CONCODE [159], and NL2Bash [160] respectively represent efforts to convert natural language into Python, Java, and Bash. Subsequently, with advancements in the field, especially with the help of LLMs, this domain has experienced prosperous growth [4]. The scope has gradually covered the generation of code in multiple PLs [161], data science notebooks [162, 163], and other more complex scenarios. We will delve into more detailed discussions of these developments in Section 4.3.

Besides, Text-to-SQL can be viewed as a special case of code generation, which translates NL requirements into SQL statements [164, 165, 166]. This has been a long-studied topic and plays a significant role in bridging the gap between humans and relational database management systems [167, 168, 169]. As for their more recent developments, we will delve into this topic in Section 6.2.

For the tasks mentioned above, we have compiled a list of representative benchmarks along with their brief descriptions in Table 1. Furthermore, we expand on this list in Table 6, 7 and 8, which include additional benchmarks and cover some tasks that are not detailed extensively, such as question answering [170], comments generation [171], log analytics research [172], document translation [114], and programming learning [173, 174, 175].

The neural modeling for the code-related tasks discussed above was primarily developed before 2020, as illustrated on the left branch of Figure 3. Although most models were designed ad hoc for specific tasks and may seem relatively simple by today's standards, they signified the rise of code intelligence and laid the foundation for subsequent research.

◊ ***Takeaway:*** (1) *The application of neural networks to code marked a pioneering step, signifying the onset of neural code intelligence.* (2) *Compared to modeling NL, the incorporation of code structures represents a critical divergence. These code features can be extracted and encoded in diverse ways, a practice that subsequent research has continued to leverage.* (3) *A wide range of code-related tasks were introduced and formalized during this period, which subsequently experienced significant development. Beyond their practical value, they also became testbeds for future* *advancements.* (4) *Despite these intricately designed techniques becoming somewhat overshadowed in the subsequent transformer era, they remain lightweight and easy-to-use in practice, serving as an effective means to develop high-quality code representations.*

3 An Odyssey of Pre-train and Fine-tune

Following the remarkable success that pre-trained language models [27] have achieved in NLP, the code intelligence community rapidly integrated their architecture and learning paradigms, leading to the proliferation of CodePTMs. Coming after the era of neural language modeling, this marks a flourishing period for code intelligence which retains code structural insights while incorporating transformer-based models. The construction of language models for code has undergone a major paradigm shift, characterized by the following features:

1) Architecture: Multi-layer transformer [176] has become the de facto choice for model backbone, moving away from building models from scratch for each task or relying on tailored feature engineering.
2) Training Data: Pre-training is primarily conducted on large volumes of unlabeled data harvested from GitHub [177], with a smaller portion of labeled data typically used for adapting the model to various downstream tasks.
3) Learning objectives: While optimized through self-supervised objectives, the approach still retains the utilization of structural information to varying degrees to enable more effective learning of code representations.

The development of this stage is reflected in the three main branches shown in Figure 3. In this section, we will first conduct a systematic review of representative CodePTMs and their variants, followed by an in-depth discussion of their other aspects.

3.1 Pre-trained Language Models for Code

In this part, we discuss a wide range of CodePTMs, categorizing them based on their architectures.

3.1.1 Encoder-only

Existing CodePTMs with encoder-only architecture can be classified based on their use of structural information into two distinct categories: *structure-free* and *structure-based*. The former only utilizes raw code texts, whereas the latter incorporates code structure during pre-training to more effectively grasp the inherent structure of code.

• **Structure-free Models**. CuBERT [178] marks a pioneering endeavor in the integration of transformer architecture into the realm of code intelligence. It is trained on a corpus of Python data collected from GitHub, employing the same training objectives as BERT [24] and replicating its training pipeline. Another milestone is CodeBERT [28], which distinguishes itself from CuBERT by adopting a cross-modal training strategy that utilizes both bimodal NL-PL data and unimodal data. The pre-training of CodeBERT is centered around two objectives: Masked Language Modeling (MLM) and Replaced Token Detection (RTD) [179]. For implementation, it is initialized through RoBERTa [180] and trained on CodeSearchNet [113], a pioneering and influential corpus encompassing six PLs constructed by scraping open-source GitHub repositories.

Regardless of whether task-specific fine-tuning is applied, both models have achieved performance far surpassing previous word2vec models and multi-layered bidirectional LSTMs (discussed in Section 2) across a wide range of code-related tasks, paving the way for the successful application of the pre-train and fine-tune paradigm on code.

• **Structure-based Models**. After the success of CodePTMs that solely rely on code tokens for training, researchers revisit earlier strategies centered on code features, innovatively incorporating code structural information in other modalities (*e.g.*, data flow) into the training process of transformer-based models. GraphCodeBERT [181] represents one of the earliest endeavors, which leverages data flow in the pre-training stage. Beyond MLM, it innovatively introduces two tasks: predicting code structure edges and aligning code with its structure. These tasks collectively aim to enable the model to understand the relationships between variables as well as between variables and tokens. Trained on CodeSearchNet, this structure-aware training approach allows GraphCodeBERT to outperform previous models (*e.g.*, CodeBERT) on a range of code-related tasks, pioneeringly demonstrating the importance of structural information in code understanding.

Contrastive pre-training [182] emerges as another pathway. SynCoBERT [183] extends structure-aware training further by not only considering the structural information of code but also synchronizing the embeddings of code and its corresponding comments through contrastive learning, aiming to bridge the gap between code semantics and NL comments. Similarly employing contrastive objectives, CODE-MVP [184] explores multi-view learning of code. It processes different code structures, such as AST, data flow, and control flow in parallel. The contrastive objects come into play when it compares these multiple views of the same code snippet in training, thus identifying and reinforcing the commonalities and differences across these representations. DISCO [185] leverages code transformation algorithms to generate synthetic code clones and inject real-world security bugs, utilized respectively to construct positive and negative samples. This approach enables models to discern subtle differences in functionalities. Later, Li et al. [186] observe that positive samples created through transformation algorithms, such as variable renaming [187] or injecting non-functional code [188], could lead the model to prioritize learning superficial code structures over significant code semantics. To prevent the model from being misled by superficial content, SCodeR further employs code comments and subtrees of ASTs to build positive samples, compelling the model to deeply understand code semantics and learn to infer code based on its context.

Additionally, in the context of training with program transformations, the concept of identifier deobfuscation in SE has also been employed. DOBF [189] objective begins by concealing the names of functions and variables using placeholder tokens, then trains the CodePTM to restore the original names through dictionary mapping.

3.1.2 Encoder-Decoder

In contrast with encoder-only models, which excel in code understanding, their encoder-decoder counterparts possess inherent advantages in the realm of controllable text or code generation. Yet, their evolution mirrors that of encoder-only models in significant ways. Initially, code was treated purely as text and applied to encoder-decoder transformers [190], followed by the integration of various structural information to enhance the learned code representations [191]. This evolution has gradually led to the development of three distinct categories of models equipped with classic architectures, which will be discussed as follows:

• **BART**. Ahmad et al. [30] propose PLBART. Following the training objectives of BART [192], it is pre-trained on a corpora constructed by Java and Python functions (from GitHub) and NL documents from StackOverflow[8] via denoising autoencoding. PLBART is distinctive for its unified training on code and NL, aiming to learn the alignment between semantic spaces across different PLs.

• **T5**. The initial exploration of the T5 [200] architecture's potential on source code is initiated by Mastropaolo et al. [201], who drew inspiration from the concept of multitask learning. This approach commenced with the presentation of a series of code-related tasks as text-to-text transformations. Similarly, PyMT5 [193] replicates this approach by leveraging Python methods and method-docstring data.

CodeT5 [29] is the first structure-aware encoder-decoder model and is among the most influential models today. It follows the T5-learning [200] pipeline and, in addition to the original span corruption training objective, it incorporates: (1) identifier tagging, which informs the model about whether a code token is an identifier or not; (2) masked identifier prediction, similar to the deobfuscation mentioned earlier, a variant of span corruption where all identifiers tokens are masked; and (3) text $\leftrightarrow$ code generation. For pre-training data, it not only utilizes the CodeSearchNet but also extends to include C/C# data to accommodate a wider task range. In fine-tuning, it is capable of performing task-specific transfer learning as well as multi-task learning to address both code generation and code understanding simultaneously. Building on this, CodeRL [202] innovatively combines code generation with deep reinforcement learning (using Unit Test Signals), and enhances CodeT5 in terms of learning objectives, model sizes, and pretraining data, to better adapt to the NL2Code task. In light of the outstanding performance, CodeT5 has seen further development in the future, which will be discussed in Section 4.1.

Meanwhile, SPT-Code [203] enhances its input by integrating linearized ASTs, thereby enabling the use of both natural language and code structures as inputs during the pre-training phase. To improve the code generation ability of T5-based models, researchers also explore strategies to enable the decoder part to learn syntax and data flow [204]. NatGen [195] represents an extension of CodeT5 that leverages the bimodal and dual-channel nature of structural information. Like DOBF, it is trained by "Naturalizing" source code to exploit the codes' naturalness and semantics. It requires the model to receive "unnatural" synthetic code as input and produce semantically equivalent code, mirroring the quality and style a human developer would prefer to write. CodeT5Mix [205] is composed of a mixture of encodes and decoders, each with specific code functionality. They can be flexibly combined to suit different scenarios and enjoy mutual benefits from joint pretraining on various targets. Further, weight-sharing strategies in decoders are used to act as task-specific experts to reduce interference across code-related tasks. Very recently, AST-T5 [206] employs a structure-aware code segmentation method during its training process, enabling the model to reconstruct code structures at various granularities.

Beyond the aforementioned general models, specialized models also emerged for the first time during this stage. For instance, JuPyT5 [163] emerges as a CodePTM tailored for the data science domain. It is trained on Jupyter Notebook repositories from GitHub, with each cell in each notebook considered as a target during the pre-training process, aiming to serve as a data science assistant. After that, leveraging code intelligence to address data science problems has seen substantial advancement, which will be discussed in subsequent sections.

• **UniLMs**. It is noteworthy that the UniLM [207, 208] architecture has also been adopted by researchers to develop its successors trained on source code. One such model, CugLM [209], adopts BERT-like training objectives, focusing on code completion tasks.

Another essential UniLM-style CodePTM is UniXcoder [194], which integrates various novel training objectives (*e.g.*, code fragment representation learning) and utilizes cross-modal content such as AST and code comments. Interestingly, it has developed a lossless method to convert ASTs into sequences, incorporating these alongside code comments as cross-modal content for pre-training. The model also expands its training dataset by utilizing both the C4 dataset and the CodeSearchNet data. Further, it employs a prefix mechanism to determine whether the model functions as an encoder-decoder model, a decoder-only model, or an encoder-only model.

3.1.3 Decoder-only

During the era where pre-training and fine-tuning are the primary paradigms for code learning, CodePTMs with a decoder-only architecture are essentially replicas of GPT [25] models applied to code, mostly adhering to the original GPT architectures and employing Causal Language Modeling. GPT-C [198] is a variant of the GPT-2 [210] trained from scratch on multilingual source code corpora. Its purpose is to serve as the *Intellicode* extension in the Visual Studio IDE, representing one of the initial attempts to utilize language models for code as coding assistants. Subsequently, GPT-CC [211], derived through fine-tuning GPT-Neo [212] on The Pile [213], has been used to build an open-source version of GitHub Copilot. Additionally, CodeGPT, a GPT-style pre-trained model is released alongside CodeXGLUE [114]. It shares a similar parameter size with CodeBERT and is utilized to help solve completion and generation problems during benchmarking machine learning research for program generation.

For task-specific variants, apart from adapting for specific PL [214], there is also a model: PyCodeGPT [199], designed to generate library-oriented codes, which share sim-

8. https://stackoverflow.com/

TABLE 2: An overview of Code Pre-trained Models' architecture and pre-training strategies, along with whether these models leverage code structure information during the pre-training phase. Due to space limitations, we abbreviate some of the training strategies, with detailed descriptions provided in Table 10.

Architecture	Models	Struct.	Base	Strategy	Size
Encoder	CuBERT [178]	✗	-	MLM + NSP	340M
	CodeBERT [28]	✗	RoBERTa	MLM + RTD	125M
	GraphCodeBERT [181]	✓	CodeBERT	MLM + Edge Pred. + Node Align.	125M
	SynCoBERT [183]	✓	CodeBERT	MMLM + IP + TEP + MCL	125M
	CODE-MVP [184]	✓	GraphCodeBERT	FGTI + MCL + MMLM	125M
	SCodeR [186]	✓	UniXcoder	Soft-Labeled Contrastive Pre-training	125M
	DISCO [185]	✓	-	MLM + NT-MLM + CLR	110M
Enc-Dec	PLBART [30]	✗	-	Denoising Pre-training	140M/406M
	CodeT5 [29]	✓	-	MSP + IP + MIP + Bimodal Generation	60M/220M/770M
	PyMT5 [193]	✗	-	MSP	374M
	UniXcoder [194]	✓	-	MLM + ULM + MSP + MCL + CMG	125M
	NatGen [195]	✓	CodeT5	Code Naturalization	220M
	TreeBERT [191]	✓	-	TMLM + NOP	210M
	ERNIE-Code [196]	✗	mT5	SCLM + PTLM	560M
	CodeExecutor [197]	✗	UniXcoder	Code execution + Curriculum Learning	125M
	LongCoder [118]	✗	UniXcoder	CLM	150M
Decoder	GPT-C [198]	✗	-	CLM	366M
	CodeGPT [114]	✗	-	CLM	124M
	PyCodeGPT [199]	✗	GPT-Neo	CLM	110M

ilar code sketches (the code structure after anonymizing the user-defined terms). Innovatively, it employs a specialized trained tokenizer for Python and judges the data quality during the training process through each file's star count and unit test function rate, prioritizing the high-quality portions. These efforts enable PyCodeGPT to achieve excellent code generation capabilities, and the corresponding techniques have been adopted in subsequent research.

In comparison to the previous two types of CodePTMs, the development of decoder-only models at this stage is somewhat constrained, predominantly revolving around developing the "code-version" of GPT-2. Owing to their autoregressive characteristics, these models find it hard to incorporate structural information of code into their training process. Nonetheless, they will demonstrate remarkable achievements in later research endeavors, which will be comprehensively investigated in Section 4.1.

3.2 Task-specific Adaptation of CodePTMs

Unlike the practices in the pre-transformer era where each task requires individualized modeling, CodePTMs, similar to their counterparts in NLP, can adapt to the required scenarios through task-specific fine-tuning or by adding new training objectives without significant modifications to the architecture. Additionally, they benefit from readily available end-to-end pipelines [215, 216, 217] and toolkits [218].

We first revisit the task-enhanced variants of CodePTMs mentioned in Section 3.1. Multiple variants have opted to build upon GraphCodeBERT [181] for their developments. To enhance the capability of code search, CodeRetriever [219] incorporates additional Uni/Bimodal training objectives, making the model more aware of the semantic similarities between code-code or code-text pairs. In efforts to utilize external information to bolster code generation and summarization, both REDCODER [144] and ReACC [220] extend GraphCodeBERT by integrating an additional dense retriever. Additionally, to automate code review activities, CodeReviewer [221] is constructed by training real-world code changes and code reviews on CodeT5 [29]. Later, CodeExecutor [197] utilizes UniXcoder [194] as its basis, further learning to execute programs and predict their execution traces for improved code generation capabilities.

We then consider models retrained from scratch for specific scenarios. In terms of further leveraging code features for enhanced generation, CodeTransformer [222] leverages both the context and structure of code to extract language-agnostic features, aiming to build a multilingual code summarization model. GrammarFormer [223] utilizes code syntax to guide the generation of code completions, enabling it to refrain from making predictions in instances where the context is ambiguous. Regarding long-range modeling, Clement et al. [224] enhance transformer models by prioritizing higher-level syntactic elements to extend context windows, considering a broader range of contextual information. Conversely, to facilitate code completions with longer contexts, LongCoder [118] employ sparse attention. It utilizes a sliding window to attend only the local information, enabling models to maintain high performance even when dealing with extensive code segments. Moreover, ERNIE-Code [196] represents a specialized case, being among the first to recognize the importance of moving beyond English-centric texts. It advocates for multilingual text-to-code generation and summarization, considering both NL and PL, increasing the accessibility for a global user base.

3.3 Understanding and Analyzing CodePTMs

As the field progresses, in addition to exceptional performance, explorations into why these models work and what features they can capture have gradually begun. Furthermore, researchers have also started to pay attention to the security threats to these models.

3.3.1 Understanding the Inner Mechanisms of CodePTMs

It is crucial for us to understand the inner mechanisms of language models for code and their differences from parallels in NL. Drawing from the experiences in explainable deep learning over the past few years [225, 226, 227, 228], research into their interpretability has primarily focused on two main areas: (1) task-level inspection and (2) internal mechanisms exploration in conjunction with code structure.

Karmakar and Robbes [229, 230] first construct diagnostic tasks to discover to what extent CodePTMs learn about specific aspects of source code. Troshin and Chirkova [231] also employ probing tasks to verify models are aware of code syntactic structure. The follow-up research delves into the inner workings, drawing inspiration from previous research targeting NLP models [232, 233, 234] and protein models [235]. A primary focus is analyzing models' attention within Transformer layers, utilizing the structural information of code to provide additional signals for analysis. Specifically, Wan et al. [236] conduct qualitative analyses to evaluate how CodePTMs interpret code structure, discovering that attention aligns strongly with the code's syntax. Subsequently, quantitative characterization of the code structure learned by models is also established by linking attention weights to AST nodes [62]. Probing experiments also point out that CodePTMs can induce entire ASTs [237].

Attention analysis also highlights CodePTMs' propensity to prioritize specific types of tokens and statements, notably keywords and data-relevant statements. Based on these findings, input codes can be simplified for the model's lightweight application [238]. More recent research has highlighted how the lexical, syntactic, and structural properties of code are distributed across different model layers, which could pave the way for more efficient fine-tuning strategies for code models via layer freezing [239].

3.3.2 Evaluating the Robustness and Safety of CodePTMs

Just like their NL counterparts, CodePTMs might not be resistant to changes in the input and, thus, are potentially susceptible to adversarial attacks [240] and perturbations [241]. In such scenarios, the robustness of these models requires careful investigation. Yefet et al. [242] utilize gradient-based methods to slightly perturb the input code to force a given trained model to make an incorrect prediction. For robust training, program transformations with preserved semantics are also employed [243]. Considering the "naturalness" of textual perturbation, Yang et al. [244] pioneer an example generation strategy that adversarially transforms inputs to make victim models produce wrong outputs, balancing both natural semantic and operational semantics.

Attacks on CodePTMs can also be based on the structural information of code. CodeAttack [245] is a representative black-box attack method that leverages code structure to generate imperceptible adversarial code samples, achieving higher attack success rates than direct applications of adversarial attacks in NLP [246]. It has also been used to explore vulnerabilities in less common programming languages, such as code entities in R [247]. Zhang et al. [248] exploit the uncertainty in CodePTMs' outputs, using it to guide the search for adversarial examples through variable name substitution. Later on, traversing the ASTs in different ways to construct adversarial samples has also been adopted as a strategy to test the sensitivity of models to input variations [249].

In parallel, backdoor attacks have also gained attention alongside the advancement of neural code intelligence. A backdoor-attacked CodePTM can behave as usual on benign examples but will generate pre-defined malicious outputs when injected with inputs embedded with backdoor triggers [250, 251]. Subsequent developments include the creation of more stealthy backdoors [252], as well as the proposal of multi-target backdoors that simultaneously aim at code understanding and generation tasks [253].

◊ ***Takeaway:*** (1) *Applying pre-trained transformers to code represents a groundbreaking initiative, effectively addressing the previously encountered dilemma of having to model each task from scratch. Moreover, it demonstrates that pre-training on a vast corpus of unlabeled code followed by task-specific fine-tuning can significantly enhance the performance across all downstream tasks.* (2) *Various code features mentioned in Section 2.1 have been leveraged during pre-training to enhance models' perception of code structure. However, this explicit modeling of structures is not a free lunch; alterations to the model's inner mechanisms due to explicit structural modeling make it challenging for CodePTMs to generalize across different tasks.* (3) *Leveraging code structure offers an additional perspective for interpreting and analyzing CodePTMs. Compared to their NL counterparts, researchers are becoming increasingly aware of utilizing these structures to analyze model behavior. As we step into the era of LLMs, such research is still in its nascent stage [254, 255].*

4 The LLM Era: A New Frontier

The domain of code intelligence has been significantly transformed by the swift advancement of Large Language Models (LLMs) [256, 257, 258], signaling the dawn of a new era and introducing new opportunities [259]. In addition to displaying emergent abilities [260], typical LLMs such as PaLM [34], LaMDA [261] and BLOOM [262], inherently possess competent coding capabilities. This innate ability stems from their pre-training data, which is often a diverse mixture containing a considerable amount of code corpus. For instance, commonly used datasets like ROOTS [263] and the Pile [213] corpora include significant portions of code data; Pile contains 95.16GB of GitHub data out of 800GB, while ROOTS comprises 163GB out of 1.6TB. This substantial inclusion of code enables these models to learn and understand programming concepts, syntax, and semantics, thus equipping them with the ability to generate and interpret code across various PLs and tasks.

4.1 Large Language Models for Code

To harness the power of LLMs to further propel the field of code intelligence, CodeLLMs have emerged. Benefiting from the advantages of being successors to general LLMs, CodeLLMs are naturally equipped with: I. access to vast and high-quality data for training [213, 263]; II. modern positional encoding and interpolation techniques [264, 265] for tackling longer sequences. III. efficient strategies for training deployment [266, 267, 268]; IV. the resources and weights from mature open-source LLMs [269, 270, 271, 272].

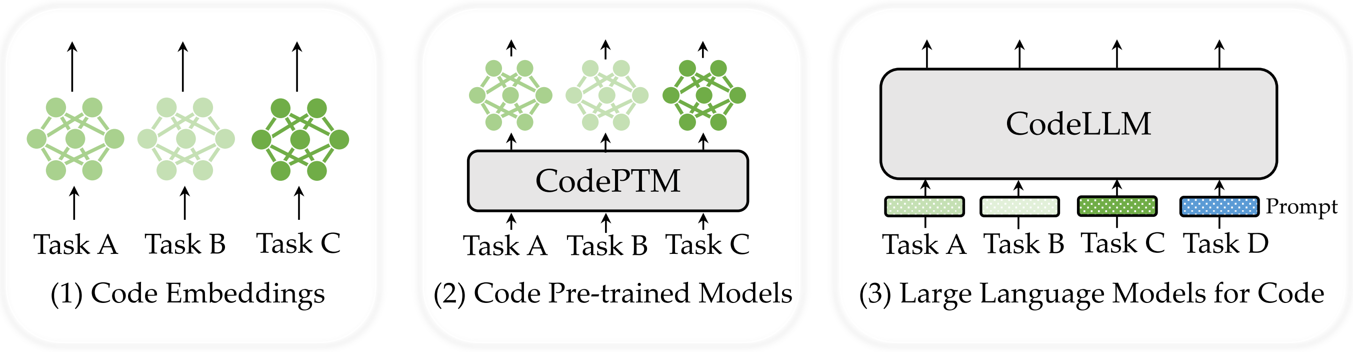

Fig. 4: Schematic illustration of different paradigms of applying language models for code to downstream applications. 🟩 🟩 🟩 indicate different code downstream tasks (*e.g.*, Defect Detection, Code Translation, NL2Code) and 🟦 indicates tasks that can be addressed by employing code-based solutions (*e.g.*, mathematical reasoning).

Supported by these technological advancements, the design philosophy of CodeLLMs has, at a macro level, evolved from the CodePTMs era in the following ways:

1) Architecture: Aside from a few cases [273, 274] retaining the encoder part, the majority of CodeLLMs have embraced decoder-only autoregressive models to better align with generative tasks.
2) Training data: Compared to their widely adopted predecessor like CodeSearchNet [113], the new emerging corpora have rapidly grown in size and the number of PLs covered, as listed in Table 3.
3) Learning objectives: There is a shift away from explicitly learning code structural information. Moreover, beyond left-to-right generation, some models are designed to learn infilling tasks [94] to support scenarios such as code completion.

TABLE 3: Representative open-source corpora for pre-training CodeLLMs with their sizes (measured by GB/TB for disk size, or measured by M in number of files) and the number of PLs they cover.

Dataset	Size	# PLs.
BigQuery (GitHub data) [link]	2.8M	6
CodeSearchNet [113] [link]	20GB / 6.5M	6
GitHub Code [link]	1TB / 115M	32
BigPython [275]	217GB	1
The Stack [276] [link]	6.4TB / 546M	358
StarCoderData [277] [link]	783GB / 207M	86
The Stack v2 [278] [link]	67.5TB	> 600

In terms of application paradigms, there is no longer a necessity to meticulously select annotated data for training from scratch or to engage in task-specific fine-tuning. As illustrated in Figure 4, the primary approach now leans towards leveraging prompt learning and providing relevant in-context demonstrations, which are usually non-invasive to models [4, 279]. Consequently, the approach to handling various code-related tasks has gradually evolved from the diversified forms mentioned in Section 2.2 to predominantly generative methods [280].

Within this ongoing evolutionary path, the contemporary CodeLLMs that have emerged can mainly be observed on the branches on the right side of Figure 3. In the ongoing discussion, we will concentrate on some of the most representative models (and their derivatives), subsequently offering a comparatively concise review of other models and the hallmark techniques they employ. As for all available CodeLLMs and their properties, we present them in Table 4, aiming to offer a comprehensive overview of the current landscape.

Codex Series. The initial version of Codex is a GPT model [33] fine-tuned on publicly available code from GitHub. Literature [36] describes a 12B version of Codex, fine-tuned on a 159 GB corpus of deduplicated, filtered Python code, which later presumably evolved to be the *code-cushman-001* within the OpenAI APIs. In 2022, OpenAI initiated the development of a new Codex variant, termed *code-davinci-002*, which is perceived as a larger model (175B) further trained on a blend of text and code data [281], and subsequently fine-tuned on instructions [282].

The advent of Codex has had a profound impact on the development of code intelligence, pioneering and demonstrating the potential of building large-scale language models specialized for code. However, it is slightly regrettable that the exact sizes, corpora, and certain training details of the Codex series remain undisclosed to the public. Nevertheless, it is undeniable that Codex is a landmark, and researchers also believe that it has played an essential role in the evolution of other seminal OpenAI models, such as *text-davinci-003* and *ChatGPT* [281]. After Mar, 2023, The Codex series is no longer publicly available, access is limited to API credit applications via the researcher access program[9].

CodeGen Series. The CodeGen series [275] is one of the first open-source CodeLLMs for code generation, featuring models ranging from 350M to 16.1B. It stands as one of the initial efforts utilizing autoregressive transformers to learn from both NL and code data, with next-token prediction language modeling training objective. CodeGen innovatively proposes a multi-turn code generation approach, where a user interacts with the model by progressively providing requirements in natural language and then receives responses in the form of "subprograms." In terms of training, CodeGen first learns general knowledge on The Pile, followed by training on a subset of Google BigQuery

9. https://openai.com/form/researcher-access-program

(which includes 6 PLs) to obtain CodeGen-Multi. The model is then further trained on BigPython to acquire a Python-oriented model, termed CodeGen-mono. At the time, these models demonstrated capabilities close to those of CodeX. Moreover, CodeGen serves to fill a niche between larger and smaller language models for code, partially marking a transition from CodePTMs to CodeLLMs.

CodeGen2 [283] represents a comprehensively upgraded iteration that delves into the training of CodeLLMs from four main aspects: model architectures, learning methods, training objectives, and data distributions. Compared to its predecessor, this version imposes stricter control over the quality of training data and is trained on mixed objects of causal language modeling and span corruption. The resulting model is capable of infilling and supports a broader range of PLs, marking a significant advancement of contemporary CodeLLMs.

The most recent model release is CodeGen2.5[10], which adopts multi-epoch training on StarCoderData [277] and employs span corruption for training. This model exemplifies the principle that with a robust data recipe—specifically, running multiple epochs and utilizing data augmentation—a relatively smaller CodeLLM (7B) can be on par with its larger predecessors (typically greater than 15B), paving the way for subsequent enhancement of models from the perspective of training data.

Interestingly, given the robust performance and flexible size options of the CodeGen series, it has also been used as an initializer for other general-purpose LLMs. A case in point is MOSS [284], which is initialized with CodeGen-mono-16B and then further pre-trained on Chinese tokens, as well as samples drawn from the Pile and BigQuery.

BigCode Models. BigCode[11] represents an open-scientific collaboration focused on CodeLLMs, aiming to provide the research community with full insight into the development process. It has made significant contributions in various aspects, including data, models, evaluation, and ethics. The CodeLLMs developed by BigCode are among the first to adopt the fill-in-the-middle (FIM) objectives [95] (also known as causal masking objective utilized by InCoder [94], which is a 6.7B model specialized in infilling). For using CodeLLMs, it is a common necessity to generate or insert contextually appropriate content based on given snippets of code. However, due to the dependency relationship with code, solely relying on "predicting next token" falls short of capturing these complexities. FIM addresses this challenge by innovatively segmenting the code into three parts, then shuffling these segments and reconnecting them with special tokens. This strategy aims to enhance the model's pre-training by incorporating a "fill-in-the-blank" that goes beyond causal modeling, enabling an understanding of bidirectional context by considering code dependencies.

One of the most representative models, StarCoder [277] has a size of 15.5 B, equipped with infilling capabilities and the capacity for efficient generation [266]. In terms of its training data, StarCoder employs heuristic filtering, manual inspection, and cleaning processes to compile the StarCoderData, which includes 86 programming languages. The format of this dataset encompasses text-code pairs, GitHub issues, Jupyter notebooks, and GitHub commits. SantaCoder [288] is an earlier variant with 1.1 B size. It shares the same architecture as StarCoder but is exclusively trained on Python, Java, and JavaScript. Additionally, this family of models also includes fine-tuned versions on conversation data to act as coding assistants [306].

Later developments release OctoCoder along with OctoPack [307], an instruction-tuned model created by fine-tuning StarCoder on newly collected commit messages that resemble instructions (CommitPackFT) and the OpenAssistant (OASST) conversations dataset. Furthermore, the data is also used to construct OctoGeeX based on CodeGeeX [290], Both models demonstrate performance that rivals non-permissive models, showcasing the effectiveness of instruction tuning and specialized training. The most recently released StarCoder2 [278] represents an even more capable version, utilizing training data that is 4 x larger and extends beyond to include notebooks from Kaggle, GitHub pull requests, and code documentation. Moreover, it uses data augmentation strategies to boost low-resource language performance by enhancing source code by pairing it with its LLVM [308] intermediate representation.

Beyond the success and impact of the aforementioned models, FIM has also been widely adopted in subsequent research. While Bavarian et al. [95] propose that FIM could be learned without harming the ability to do left-to-right generation, some research holds the view that equipping the model with this infilling ability might not be a "free lunch" [283]. For instance, models trained through FIM have been observed to sometimes struggle with determining the appropriate moments to cease infilling. We believe that these conflicting views may arise because models must learn to balance the nuances of generating code linearly with the ability to jump back and forth to fill gaps as needed. Further evidence is required to delve deeper into this discussion.

CodeT5+. Unlike the previously mentioned models that are designed as decoder-only models, CodeT5+ [274] is a family of open-source CodeLLMs with encoder-decoder architecture, ranging from 220M to 16B. This model not only scales up from its predecessor, CodeT5 [29], but also showcases refined architectural design and training objectives. Architecturally, the encoder is tasked with encoding contextual representations, whereas the decoder is adept at generating diverse types of outputs. Distinctively, CodeT5+ adopts a "shallow encoder and deep decoder" structure [273], with both parts initialized using CodeGen and connected by cross-attention. For pre-training, the objectives employ a combination of span denoising and CLM [309, 310], thereby equipping the models with the capability to learn code context representations and to reconstruct missing information at different levels, including code spans, partial programs, and complete programs.

In contrast to previous models that are treated as a single system across all tasks, CodeT5+ offers the versatility to operate in encoder-only, decoder-only, and encoder-decoder modes to accommodate different downstream applications. This flexibility significantly mitigates the issue of inter-task interference encountered in UniLM-style models [194, 209].

For further improvement, InstructCodeT5+ is created as an instruction-tuned variant to align with NL instructions,

10. https://blog.salesforceairesearch.com/codegen25/
11. https://www.bigcode-project.org/

TABLE 4: An overview of CodeLLMs categorized based on their architecture, along with their parameter size, base model (if any), vocabulary size, context length, training objectives, data scale used for training (measured by K/B/T in number of tokens, or measured by GB for disk size), and their public availability. Due to space limitations, we do not differentiate between various versions of CodeLLMs and their bases. For models built upon a base, the data scale refers to the size of the corpora used during additional pre-training.

Arch.	Model Name	Size	Base	Vocab.	Context	Training Objs.	Data Scale	Public
Enc-Dec	AlphaCode [273]	284M/1.1B/2.8B/8.7B/41.1B	-	8.0K	1536+768	MLM+CLM	354B/590B/826B/1250B/967B	✗
	CodeT5+ [274]	220M/770M/2B/6B/16B	CodeGen	50.0K	2048+2048	MSP+CLM+CL	51.5B	✓
Decoder	Codex [36]	2.5B/12B	-	50.3K	4,096	CLM	100B/159GB	✓
	CodeParrot [285]	125M/1.5B	-	32.8K	1,024	CLM	26B/50GB	✓
	PolyCoder [286]	160M/0.4B/2.7B	-	50.3K	2,048	CLM	39B/254GB	✓
	CodeGen [275]	350M/2.7B/6.1B/16.1B	-	50.0K	2,048	CLM	1.2T	✓
	PaLM-Coder [34]	8B/62B/540B	PaLM	256k	2,048	CLM	7.75B	✗
	InCoder [94]	1.3B/6.7B	-	50.3K	2,048	FIM	52B/159GB	✓
	PanGu-Coder [287]	317M/2.6B	-	42K	1,024	CLM+MLM	387B/147GB	✗
	SantaCoder [288]	1.1B	-	49.2K	2,048	FIM	236B/268GB	✓
	phi-1 [289]	350M/1.3B	-	50.0K	2,048	CLM	7B	✓
	CodeGeeX [290]	13B	-	52.2K	2,048	CLM	850B	✓
	CodeGen2 [283]	1B/3.7B/7B/16B	-	50.0K	2,048	MLM+CLM	400B	✓
	StarCoder [277]	15.5B	-	49.2K	8,192	FIM	1T/815GB	✓
	CodeAlpaca [291]	7B/13B	LLaMA	32.0K	4,096	CLM	20K	✓
	WizardCoder [292]	1B/3B/7B/13B/15B/34B	StarCoder	32.0K	2,048	CLM	78k	✓
	AquilaCode [293]	7B	Aquila	100.0K	2,048	CLM	-	✓
	CodeGeeX2 [290]	6B	ChatGLM2	65.0K	8,192	CLM	600B	✓
	CodeLLaMA [294]	7B/13B/34B/70B	LLaMA2	32.0K	4,096	FIM	500B	✓
	ToRA-Code [295]	7B/13B/34B	CodeLLaMA	32.0K	2,048	Min. NLL	223K	✓
	MAmmoTH-Coder [296]	7B/13B/34B	CodeLLaMA	32.0K	2,048	CLM	260K	✓
	Code-Qwen [297]	7B/14B	Qwen	152.0K	8,192	CLM	90B	✓
	CodeFuse [298]	1.3B/6.5B/13B/34B	Multiple	100.9K	4,096	CLM	1.6TB	✓
	CodeShell [299]	7B	-	70.1K	8,192	CLM	500B	✓
	Lemur [300]	70B	LLaMA2	32.0K	4,096	CLM	90B	✓
	DeepSeekCoder [301]	1.3B/5.7B/6.7B/33B	-	32.0K	16,384	FIM	2T	✓
	Symbol-LLM [302]	7B/13B	LLaMA2	32.0K	4,096	CLM	2.25GB	✓
	Stable Code [303]	3B	-	50.3K	16,384	FIM	1.3T	✓
	DeciCoder [304]	1B/6B	-	49.2K	2,048	FIM	446B	✓
	StarCoder2 [278]	3B/7B/15B	-	49.2K	16,384	FIM	900B/3TB	✓
	CodeGemma [305]	2B/7B	Gemma			FIM		✓

employing a strategy revolving around using synthetic instruction-following prompts [311, 291, 312].

CodeLLaMA. Shortly after the debut of LLaMA2 [313] in July 2023, CodeLLaMA [294] is swiftly released as a family of foundation models for code generation, covering model sizes 7B, 13B, and 34B (with 70B later announced in Jan 2024). Diverging from aforementioned models that are trained exclusively on code from scratch, CodeLLaMA is derived from LLaMA2 through training on an additional 500B code tokens. The 7B, 13B, and 34B versions have also been trained with FIM, allowing them to insert code into existing code. For the 70B version, an extra stage of long context fine-tuning was introduced, leveraging position interpolation [314] to extend the context length from the initial 4K, as was standard with the LLaMA2 model, to an extended 16K. Experiments have shown that the pursuit of tackling longer sequences might slightly hurt the performance on shorter sequences. Still, it boosts the model's ability to generate meaningful content in tasks like long code completion [118].

In addition to the foundation models, Meta has provided two additional variations, namely (1) CodeLLaMA - Python, a language-specialized variant of CodeLLaMA that has been further fine-tuned on 100B Python code. (2) CodeLLaMA - Instruct, which is trained on self-instruct [312] dataset created by prompting LLaMA2 with programming problems, as well as data aimed at improving safety and helpfulness. These models have exerted a profound and positive impact on the domain, being widely applied in the development of derivatives [295, 296, 315] and various instruction tuning practices [316, 317].

Beyond the pursuit of enhanced capacity, CodeLLaMA takes a forward-looking step toward responsible AI and safety. It rigorously compares itself against other CodeLLMs from the perspectives like truthfulness [318], toxicity [319], and bias [320] in generated code and text. Through empirical evaluations, CodeLLaMA demonstrates the possibility of achieving high coding performance without compromising harmlessness.

DeepSeek-Coder. DeepSeek Coder [301] represents a range of open-source CodeLLMs with sizes varying from 1.3B to 33B, trained from scratch on a curated code corpus. In terms of data composition, besides utilizing a filtering process for GitHub data similar to the StarCoderData rules [277], it uniquely constructs repository-level code data to enhance the model's capability for cross-file completion within repositories. Moreover, it incorporates a small code-unrelated Chinese corpus to aid the model in understanding instructions in Chinese. The model is trained on 2T tokens comprising 87 PLs, with combining objectives of next token prediction and FIM. Further, it employs linear scaling to extend the context window [314] to 16K, supporting repository-level code training. In terms of performance, it has achieved stunning results on tasks such as NL2Code and code completion, being considered one of the strongest open-source CodeLLMs currently available.

Beyond the models built from scratch, DeepSeek also provides versions that continue pre-training on general LLMs, termed DeepSeekCoder-v1.5 which can be viewed as a branch of DeepSeek LLM [321]. Compared to its code-exclusive counterpart, this variant, while witnessing a slight decrease in coding capabilities, performs better in tasks for math and natural language. Later, it played a vital role in constructing mathematical models [322].

Lemur. Contrasting with the prevailing open-source CodeLLMs that predominantly focus on code-centric optimization, Lemur [300] represents a novel endeavor to lay the groundwork for LLM-based agents. To achieve this, LLMs must possess not only robust coding capabilities to ensure precise grounding in the relevant environments [323, 324] but also the capacity to comprehend human intentions, reasoning, and planning. Therefore, Lemur advocates for harmonious integration of language understanding and coding proficiencies.

Similar to CodeLLaMA, the model is built by continually pre-training on LLaMA2. However, in terms of data composition, it employs a 90B corpus with a 10:1 ratio of code to text, followed by fine-tuning with instructions from diverse sources. Lemur can achieve stunning results on both NL benchmarks, *e.g.*, MMLU [325], BigBench [326] as well as on code-related benchmarks, including multilingual code generation [161], SQL [164], and data science [327]. Compared to other LLMs that exhibit a disparity between NL and code capabilities, Lemur stands out with a balanced skillset, achieving the highest overall performance when averaged across a variety of tasks. Moreover, Lemur uniquely excels in practical agent tasks, making significant strides in tool usage, self-debugging, following complex instructions, and navigating partially observable environments [328].

Beyond the aforementioned representatives, the community has witnessed a blossom of interesting and solid works, as illustrated by the right branch of Figure 3. These contributions mainly revolve around (1) additional pre-training on general LLMs, (2) instruction-tuned variants, (3) advanced tool-use capability, and (4) efficiency enhancements.

Regarding CodeLLMs that are built from additional pre-training, CodeGeeX2 [290] represents the second iteration of the CodeGeeX model lineage. Unlike its predecessor, which was trained from scratch, Diverging from its predecessor trained from scratch, it is developed upon ChatGLM2 [329, 330] with further pre-training on code tokens. Similarly, Code-Qwen [297] follows a training approach akin to CodeLLaMA [294], using base model Qwen trained on a combination of text and code data as an initialization and then continuing to pre-train on code data. This approach is mirrored in other models such as AquilaCode[12], which also enhances base models with extra training on code corpora. Recently released CodeGemma [305] denotes a compilation of lightweight models crafted through further code infilling training on Gemma [331], particularly adept at code completion and generation from provided code prefixes or suffixes.

Researchers also apply diverse instruction-tuning strategies to CodeLLMs as well. CodeAlpaca [291, 311] is initially built as an instruction-following LLaMA model for code generation. WizardCoder [292] is constructed by fine-tuning StarCoder [277] using Evol-Instruct [332] and ChatGPT feedback seeded by CodeAlpaca dataset [291] WaveCoder [333] enhances CodeLLMs through an innovative instruction tuning process. It employs an LLM-based generator-discriminator framework to produce a wide array of high-quality instruction data for multiple code-related tasks, focusing on improving data quality and task diversity for fine-tuning. Recently, DolphCoder [334] also adopts diversified tuning strategies. It begins by leveraging multiple chain-of-thought [335] responses to the same instruction and then combines the tasks of code generation and code evaluation in the form of natural language generation. Likewise, MoTCoder [336] utilizes a modular approach for instruction tuning. It segments complex coding tasks into logical sub-modules, guiding models to first outline and then implement these sub-modules.

Regarding tool use [337, 338], ToRA [295] targets building tool-integrated agents, addressing complex mathematical reasoning. To achieve it, ToRA is trained upon the collected interactive trajectories of invoking tools. Nevertheless, its coding ability is mainly limited to Python. In the same vein, MammoTH [296] concentrates on equipping off-the-shelf LLM with Python-integrated reasoning abilities (will be discussed in Section 5.1). It utilizes a hybrid composition of data generated from intermediate steps when reasoning with NL or code for further pre-training on LLaMA. This approach is expected to unleash both program-aided and NL-centric power in mathematics. Concurrent with Lemur's success in harmonizing text and code capabilities, a new foundational model, Symbol-LLM [302], expands the scope of code capabilities to encompass the entire range of symbol-centric capabilities, complemented by an external symbolic solver. This extension broadens the application scope of LLMs to more intricate scenarios beyond code generation, such as neuro-symbolic reasoning.

As for efficiency-optimized CodeLLMs, phi-1 [289] distinguishes itself through its compact size and the unique approach of utilizing high-quality, "textbook-quality" data for training. It demonstrates the efficacy of quality over quantity in data selection and the potential for smaller code models to compete with or outperform larger counterparts. Factors influencing the quality of code data have also been explored in subsequent studies [339]. When it comes to CodeLLMs that can run on consumer-level hardware, Stable Code [303] (based on Stable LM [340]) and DeciCoder [304] adopt a more compact and concise design. These models uphold a certain performance standard while allowing users

12. https://huggingface.co/BAAI/AquilaCode-multi

to deploy them locally, enhancing the accessibility of code generation to a broader range of users.

In the realm of application frameworks, CodeTF [217] initially provides an interface for both training and inference, facilitating the integration of CodeLLMs into practical applications. This framework aims to make it easier for developers to leverage the power of CodeLLMs in efficiency and functionality of software development processes. MFT-Coder (CodeFuse) [298] presents a multi-task fine-tuning framework specifically designed for CodeLLMs. It supports the efficient tuning and deployment of a broad spectrum of models, enabling developers to adapt and optimize these models for various coding tasks and challenges swiftly. For example, CodeFuse-CodeLLaMA is a model created by further training of CodeLLaMA through MFT, which achieves performance surpassing that of GPT-4 on the HumanEval benchmark.

4.2 Learning with Execution Feedback

Another pathway to further enhance CodeLLMs involves integrating Reinforcement Learning (RL), which incorporates non-differentiable reward signals into the training process. Unlike approaches represented by reinforcement learning from human feedback (RLHF) that utilize human preferences [341, 282, 342], the inherently compilable and executable nature of codes allows for the use of compilers or interpreters to automatically generate precise feedback. Such endeavors were initiated in the era of CodePTMs, as COMPCODER [343] harnesses the compilability signals to optimize both the generator (*e.g.*, CodeGPT [114]) and the discriminator (MLPs) via RL strategies.

As the capability for code generation improves, using RL in code training becomes increasingly flexible. CodeRL [202] exploits the code unit test signals in both training and inference stages and uses RL to optimize the model. PPOCoder [344] combines CodeLLMs with Proximal Policy Optimization [345] for code generation. RLTF [346] is another novel online RL framework, which uses unit test feedback of multi-granularity for refinement. RLCF [347] further enhances a CodeLLM by incorporating feedback from a grounding function, which assesses the quality of generated codes. Pangu-Coder2 [348] introduces an RRTF (Rank Responses to align Test & Teacher Feedback) framework, aimed at steering the model towards producing higher-quality code achieved by synergistically using test signals and human preferences as combined feedback. ExeDec [349] innovates in decomposing tasks into execution subgoals, improving compositional generalization through tackling complex tasks step-by-step. In a similar vein, recently released StepCoder [350] innovates by breaking complex tasks into a curriculum of subtasks, tackling code generation's exploration and optimization challenges.

4.3 Advancements in NL2Code

In the era of LLMs, the ability of NL2Code has leaped forward, with machine learning models now truly capable of assisting professional developers through crafting accurate code snippets based on human intent. This holds a tantalizing promise of "programming in natural language". Moreover, the role of NL2Code has also transitioned; it has transcended the initial function as merely a downstream coding task and has become a pivotal metric for evaluating the capabilities of LLMs. Here we first discuss the shift in evaluation paradigms and then focus on extensively utilized benchmarks and their derivatives.

4.3.1 *The Shift in Evaluation Metrics*

Limitations of Match-based Approaches. In the Pre-LLM era, code generation capabilities were primarily benchmarked by matching samples against a reference solution, using metrics like (smoothed) BLEU scores [358, 359, 67]. Nevertheless, in addition to the lingering issues already identified in NLG systems [360, 361], BLEU-based evaluations struggle to capture semantic features specific to code [362]. Although variants like CodeBLEU [363] propose several semantic modifications based on code structure, a fundamental problem remains: match-based metrics are unable to fully represent the broad and complex space of programs that are functionally equivalent to the reference solutions. This dilemma extends to other evaluation metrics [106, 107, 364, 365] as well. Consequently, Chen et al. [36] advocate the use of execution-based evaluation to measure functional correctness for NL2Code tasks instead.

The Rise of Execution-based Evaluation. For evaluating the functional correctness of a generated code snippet, the most reliable approach is to examine if it can be successfully executed and passes a set of unit tests, a method commonly employed in software engineering's test-driven development. pass@k is initially designed for assessing pseudocode-to-code translations [105]. Generating k code samples for each problem, a problem deemed solved if any of the samples pass, and it reports the total fraction of problems solved. However, due to the high variance, Chen et al. [36] refine it into a more stable metric as delineated in Equation 1.

$$\text{pass@}k := \underset{\text{Problems}}{\mathbb{E}} \left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right] \tag{1}$$

This revised method involves producing $n \geq k$ samples for each task, counting correct samples $c \leq n$ that pass unit tests successfully, thereby deriving an unbiased estimator. pass@k represents a milestone in NL2Code evaluation, which transcends the limitations of earlier metrics incapable of accurately reflecting functional correctness and has become a standard practice in all subsequent benchmarks.

4.3.2 *Commonly Used Benchmarks and Evaluations*

Herein, we offer detailed descriptions and discussions for HumanEval [36] and MBPP [352], two of the most widely utilized contemporary benchmarks, along with MultiPL-E [161] benchmark derived from them. Subsequently, we provide a bird's eye view of other datasets and the new benchmarks based on them. Subsequently, we present an overview of additional datasets and introduce benchmarks developed for specific purposes.

• **HumanEval**. HumanEval was initially released in conjunction with Codex [36], comprising 164 manually crafted Python coding problems. These problems are validated using test cases (with an average of 7.7 tests per problem) to evaluate the code generated by CodeLLMs, typically in

TABLE 5: An overview of the representative NL2Code benchmarks categorized according to task purpose, along with the number of programming languages they cover and brief descriptions. The complete benchmarks are listed in Table 9.

Purpose	Dataset	Date	# PLs.	Description
Open Domain	CONCODE [159] [link]	2018	1	a dataset with over 1M examples consisting of Java classes
	ODEX [351] [link]	2023	1	an open-domain execution-based NL to Python code generation dataset
Code Exercise	HumanEval [36] [link]	2021	1	a dataset of 164 handwritten programming problems with unit tests
	MBPP [352] [link]	2021	1	a dataset containing 974 short Python programs
	BIG-Bench [353, 326] [link]	2023	-	a benchmark containing over 12 tasks can be solved by coding
Competitions	APPS [354] [link]	2021	1	a benchmark including 10K less-restricted problems for code generation
	CodeContests [273] [link]	2022	3	a dataset specifically for competitive programming problems
Multilingual	MBXP [355] [link]	2023	12	a benchmark to evaluate code generation for 12 programming languages
	HumanEval-X [290] [link]	2023	4	a benchmark of 164 code problems for evaluating multilingual models
	MultiPL-E [161] [link]	2022	18	a parallel, multilanguage benchmark for NL2Code generation
Data Science	JuICe [162] [link]	2018	1	a corpus of 1.5M examples with a curated test set of 3.7K instances
	DSP [163] [link]	2022	1	a collection of 1K problems curated from 306 pedagogical notebooks
	DS-1000 [327] [link]	2023	1	a Python code generation benchmark of 1K data science problems
Python Libs	PandasEval [199] [link]	2022	1	a dataset consisting of 101 programming problems on Pandas library
	NumpyEval [199] [link]	2022	1	a dataset consisting of 101 programming problems on Numpy library
	TorchDataEval [356] [link]	2022	1	a dataset with 50 programming problems using the TorchData library
Multi-Turn	MTPB [275] [link]	2023	1	a benchmark containing 115 problem sets factorized into multi-turn prompts
Command Line	NL2Bash [160] [link]	2018	1	a corpus of 9K English-command pairs covering over 100 Bash utilities
AI4Science	BioCoder [357] [link]	2023	2	a benchmark to evaluate LLMs in generating bioinformatics-specific code

a zero-shot setting. The necessity for these problems to be manually crafted stems from the fact that the majority of contemporary CodeLLMs are trained on a large fraction of GitHub data, which already contains solutions to some ready-made programming challenges, such as those collected from Codeforces[13], LeetCode[14] and CodeChef[15]. HumanEval includes problems of varying difficulty levels, ranging from basic string and array manipulations to relatively complex data structures and algorithms. Each problem provides a function signature, clarifying the input/output format to aid the model in understanding the requirements. Since its release, HumanEval has been widely adopted by the community as a primary tool for evaluating code generation capabilities, and subsequently, it has been extended with the goals of (1) covering more programming languages and (2) constructing more robust evaluation.

Multilingual HumanEval [355] introduces a scalable automated framework capable of converting datasets from Python into variants for 12 different languages. In contrast to the automated approach, HumanEval-X [290] is another multilingual extension constructed by the authors of CodeGeeX. Similar to the original HumanEval, it is manually crafted, involving a rewrite of prompts, canonical solutions, and test cases for C++, Java, JavaScript, and Go.

Unlike the strategies of expanding the number of languages to achieve comprehensive evaluation, researchers are aware that the quantity and quality of test cases in problems can be inadequate for fully assessing functional correctness [366]. This gap may inadvertently lead to flawed code being considered correct due to the paucity of testing. To address this issue, HumanEval^{+} is developed by extending the unit tests of HumanEval, augmenting the scale of test cases by 80 times. This substantial increase in testing has proven to catch significant amounts of previously undetected incorrect code.

• **MBPP**. **M**ostly **B**asic **P**rogramming **P**roblems [352] stands as another widely recognized benchmark for assessing the performance of CodeLLMs in Python programming, especially under few-shot scenarios. Unlike HumanEval, which features varying levels of difficulty, MBPP is tailored to be solvable by entry-level programmers. It contains 974 short Python functions, each accompanied by an English description, a predefined function signature, and three manually written test cases for validation. As for its compilation, MBPP amalgamates a vast collection of crowd-sourced questions and a smaller manually edited and verified subset. The difficulty of these questions ranges from simple numerical manipulations to those requiring some external knowledge (*e.g.*, the definition of the Fibonacci sequence).

In a similar vein, MBBP has also been expanded to accommodate multilingual scenarios, aiming to offer a more comprehensive and diverse evaluation. One of the most notable extensions is MBXP (Most Basic X Programming Problems, where X = Java, Go, Ruby, etc) [355], constructed in a parallel to Multilingual HumanEval. Moreover, it has been extended to cover other code-related scenarios, encompassing zero-shot code translation, prompt perturbation test, code insertion, and code summarization.

• **MultiPL-E**. MultiPL-E [161] emerges as a new benchmark system tailored for multilingual scenarios, constructed upon the two benchmarks discussed previously. It extends their scope by translating them into 18 additional programming languages chosen according to TIOBE[16] rankings and GitHub usage frequencies. To accommodate the diverse languages, it features a containerized sandbox environment, which compiles programs (if necessary), runs them with test cases and appropriate timeouts, and categorizes each output as successful, syntax error, etc. Furthermore, MultiPL-

13. https://codeforces.com/
14. https://leetcode.com/
15. https://www.codechef.com/
16. https://www.tiobe.com/tiobe-index/

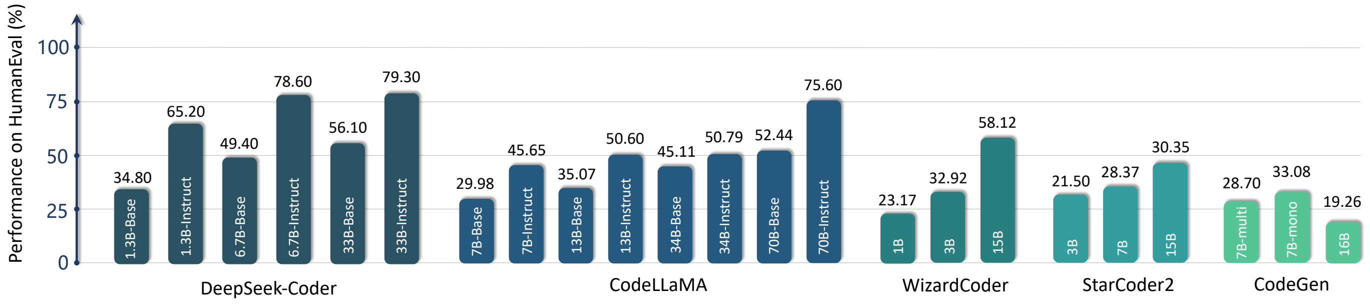

Fig. 5: The Pass@1 performance of selected open-source CodeLLMs on HumanEval, showing the result for each model across different sizes and versions.

E stands out for its scalability and extensibility, offering a streamlined process for incorporating new benchmarks and languages, thereby minimizing the need for manual intervention.

Beyond the aforementioned benchmarks, more works regarding evaluation are emerging alongside the surge in interest in CodeLLMs, broadly categorized into three types: (1) Those tailored for specific NL2Code scenarios, like Code-Contests [273] and APPS [354] designed for programming competitions. (2) The conversion of general benchmarks into testbeds for code generation, such as MathQA-Python [352] and selected BigBench [353] problems. (3) Those that consolidate, refine, and repurpose existing benchmarks, such as L2CEval [367], which has been developed as a standardized benchmark combining semantic parsing, math reasoning, and Python programming. Representative benchmarks for various purposes are shown in Table 9, and a comprehensive list of benchmarks currently available is presented in Table 5. We will further discuss the opportunities and challenges of benchmarking CodeLLMs in Section 7.4. Additionally, theoretical frameworks for evaluating CodeLLMs are also emerging, such as applying category theory [368] to express the structural aspects of NL and code.

To provide readers with an intuitive understanding of the current CodeLLMs' capabilities, we display the HumanEval performance of some open-source representatives in Figure 5. It is evident that, in addition to the positive correlation between model size and performance, models of the same size that have been fine-tuned with instructions significantly outperform their base versions. The benefits that instruction fine-tuning brings to CodeLLMs are more pronounced than those observed in general LLMs. We hold the view that this gap is likely because human intentions, compared to code comments or documents, are more intricate. Research has shown that models like *code-davinci-002* achieve top performance in benchmarks, yet their outputs may not always align with human expectations [369, 281]. Therefore, for developing practical CodeLLMs, dedicating more effort to instruction fine-tuning could be more cost-effective than merely expanding the model size.

4.3.3 Strategies for Enhanced Code Generation

Building upon our discussions above, here we explore a variety of innovative strategies that have significantly advanced the capabilities of code generation.

Decoding-Enhanced. In terms of decoding strategies, Zhang et al. [370] propose that conventional decoding algorithms may not be the best choice for code generation, and utilize lookahead search to guide future token choices. Self-planning [371] improves code generation by first creating a high-level plan to break down complex tasks into simpler steps, and then generating code for each step. Self-infilling [372] enhances code generation by combining sequential context generation with infilling, which allows for more controlled output. Yang et al. [373] break down the problem into multiple steps and utilize programming hints as support, allowing lightweight models to generate better programs. Considering the reuse of existing code, AceCoder [374] searches for programs with similar requirements as guidance generation for improved code generation. Taking the structural information into account, Li et al. [375] ask the model to first generate the program's structure (*e.g.*, loops and branches) and then implement the code.

Feedback-Driven. As discussed in Section 4.2, execution feedback represents an external signal that code can innately utilize. Employing/building test cases [376, 377, 378] within code generation processes emerges as a viable means to enhance the reliability of the generated code. CodeT [379] exemplifies this category of methods by unleashing the model's inherent capability to automatically generate unit tests, ensuring the consistency of outputs through broader tests. Similarly, TiCoder [380] enhances code reliability by constructing an augmented set of examples to cover a wider range of possible user inputs. LEVER [381] enhances NL2Code generation by integrating execution feedback provided by trained verifiers, harnessing both the generative and discerning capability. Further, ALGO [382] advances in using LLM-generated oracles for algorithmic program synthesis. Beyond NL2Code tasks, this concept of leveraging unit test feedback can also be applied to other tasks, such as multilingual code translation or code search, ensuring the functional equivalence of translated content [383] or retrieved code [384].

NL-Directed. Utilizing NL information represents another pathway. Given an instruction, Zhang et al. [385] evaluate the likelihood of the given instruction given the generated programs, thereby improving code quality through reranking. DocPrompting [386] enhances the code generation process by explicitly retrieving relevant document pieces from a pool based on user intent. Very recently, AlphaCodium [387] proposes a test-based, iterative process combining understanding NL and code generation.

In addition to the insights gleaned thus far, there are ongoing studies within the field that may hold significant implications for building CodeLLMs in the future, such as better tokenization and efficiency improvements. These topics will be elaborated upon in Section 7.6 and Section 7.7.

◊ ***Takeaway:*** (1) *The advent of CodeLLMs has been revolutionary for code intelligence, heralding a new learning paradigm. Whether as variants of general LLMs or built from scratch, they demonstrate exceptional capabilities not seen in their predecessors, achieved through larger sizes, premium code data, task-oriented training objectives, and intricately designed tuning strategies.* (2) *The integration of RL with CodeLLMs offers a promising avenue for enhancing code generation through the use of non-differentiable reward signals, such as compiler feedback and unit test results. This allows for the precise and automatic generation of feedback, overcoming the high costs associated with learning from human preferences.* (3) *Whether in terms of model capabilities or the diversity of tasks, NL2Code has experienced unparalleled expansion during this period. Further, despite a multitude of efforts to devise intricate matching-based evaluation metrics, the LLM era has seen a shift in evaluating code generation tasks toward reliance on execution to assess functional correctness.* (4) *Compared to the various ingenious methods mentioned in Section 3.2, aimed at enhancing code-related tasks through specific training, the approaches for handling downstream tasks have converged towards generative methods. Essentially, these are predominantly based on prompting, which is non-invasive in nature.*

5 Synergies in Machine Intelligence

Following our detailed exploration centered on code generation and comprehension, in this section, we delve into some synergies with other aspects of machine intelligence. Specifically, we will discuss from three perspectives: new reasoning paradigms based on code generation, mathematical abilities enhanced by code training, and the multi-dimensional capabilities of language models expanded through code.

5.1 Binding Code Generation with LLM Reasoning

Reasoning constitutes a long-standing task in the domain of machine intelligence [388]. The surge in LLM enthusiasm has brought to light approaches exemplified by chain-of-thought (CoT) prompting [335], proposed as a novel form of in-context learning [38] wherein the exemplar contains the rationales instead of merely an answer. These methodologies, which encourages LLMs to generate a series of intermediate steps toward a final solution [389], have made stunning progress across a wide spectrum of textual and numerical reasoning benchmarks [390, 391, 392, 393, 394].

Unlocking a New Reasoning Paradigm. CoT-based approaches do not serve as a silver bullet for solving all reasoning problems. Beyond factuality [393] and hallucinations [395, 396] issues in LLMs, given the inherent limitations language models face with complex arithmetic operations and managing large numbers [397, 398], LLMs are susceptible to logical and arithmetic mistakes in the calculation phase, despite the problem decomposition being correct. In light of this dilemma, code generation offers a potential

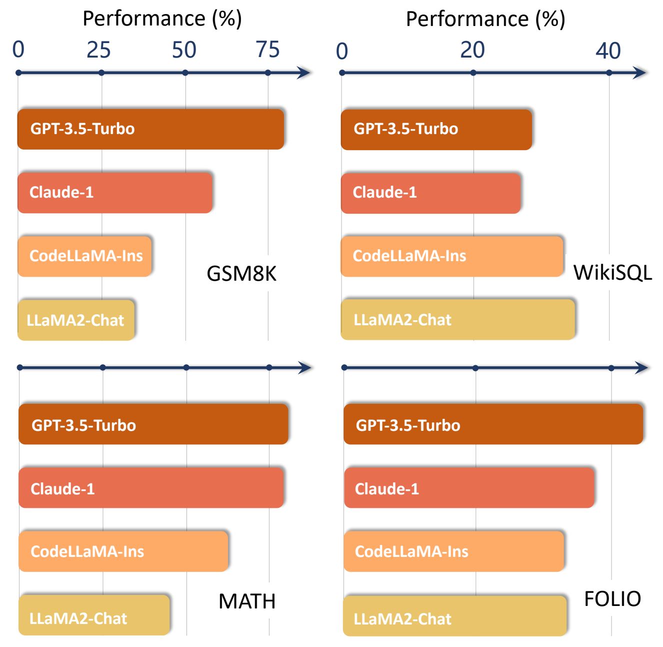

Fig. 6: The performance of four LLMs using PAL on four different tasks. Experimental details are shown in appendix E.1.

pathway for disentangling computation from reasoning. Program-Aided Large Language Models (PAL) [399][17] and Program of Thoughts (PoT) [400] employ LLMs to comprehend natural language problems and synthesize programs as intermediate reasoning steps. Crucially, they delegate the execution of the final solution to a symbolic solver (includes but not limited to a Python interpreter [401]), thereby resolving the computational limitations of LLMs.

This strategy of decoupling computation from reasoning and language understanding, has achieved great success across a wide range of datasets involving mathematical reasoning [402, 403, 404], symbolic reasoning [353, 326], and semi-structured understanding [405, 406]. For instance, bolstered by PAL, leveraging a CodeLLM like Codex has significantly outperformed the results obtained by larger models (*e.g.*, PaLM/Minerva) on math word problems, BIG-Bench Tasks, and financialQA datasets using NL reasoning chains. This approach has gradually emerged as a new reasoning paradigm for solving all numerical-related problems.

Laying Foundations for Broader Reasoning Scenarios. Emerging variations of PAL or code integration [407] into broader reasoning frameworks have also been developed. For example, combining it with CoT to merge the strengths of both approaches through model selection offers enhanced flexibility and generalizability [408]. Alternatively, integrating intermediate codes step with stochastic beam search to guide decoding presents another innovative direction [409]. Chameleon [410] modularizes the function generation process and integrates it with other tools, such as web search engines, to tackle more complex reasoning scenarios. For

17. The idea of integrating LLMs with an external PL interface was proposed by Gao et al. [399] and Chen et al. [400] within the same timeframe. Based on the descriptions adopted in the literature we surveyed, we use the terms PAL and PoT interchangeably in this paper.

collaborative reasoning involving multi-model interactions, following the use of NL reasoning chains [411], code has also been leveraged as an alternative format for communication [412, 413, 414]. In addition to logic and arithmetic task [415, 416], more recent research, Chain of Code [417], encourages models to represent semantic tasks in the form of pseudocode. This simple yet effective CoT extension enables models to explicitly identify and address undefined behaviors by simulating code execution.

Beyond offering performance gains in reasoning, the deterministic execution of code also can be utilized for interpretability. In pursuit of faithfulness, a notable example is Faithful CoT [418]. This approach converts a natural language query into a chain that interleaves natural language and a task-dependent symbolic language, such as Python or PDDL, helping us understand how the model arrives at the final answer.

In sum, integrating code generation with LLM reasoning is a great breakthrough. This new paradigm, which leverages models to interpret natural language and then generate programs as intermediate reasoning steps for execution, has transcended the entrenched limitations of traditional solutions. Moreover, it lays the groundwork for exploring the synergy between generative models and neuro-symbolic AI in a broad range of scenarios [419, 420, 421]. Nevertheless, PAL is not a panacea and faces challenges such as model misinterpretation of questions and the fact that generated codes are not always error-free. Furthermore, the relationship between code utilization and enhancements in reasoning capabilities has not been fully elucidated [422], making it an attractive and ongoing area of research.

5.2 Code Training Elicits Mathematical Capabilities

Mathematics capability has historically been considered the Achilles' Heel of language models [423]. However, with the advent of LLMs, there has been a significant shift in this perception [424]. Code training is believed to have played a pivotal role during this transformation.

Unforeseen Benefits of Code Training. Before the buzz around LLMs, it was already discovered that language models trained on code could directly solve basic algebra [425] or statistical problems [426]. Early research at the onset of the LLM era find that *code-davinci-002*, regardless of whether it utilizes chain-of-thought, significantly outperforms other models on mathematical tasks [427]. This observation has ignited the long-standing question among researchers: "Does code training improve mathematical abilities?" While the additional benefits from code training remain debatable, we collect some evidence to discuss the correlation between them. *Code-davinci-002* can easily demonstrate chain-of-thought capabilities in mathematical reasoning, achieving results far superior to those of *text-davinci-001*, which did not undergo code training [335]. Fu et al. [392] also note that when the reasoning chain is longer, *code-davinci-002* is the best-performing model on mathematical benchmarks. Finally, the traditional next-token prediction objective usually captures "local" information, while code often incurs longer dependencies and hierarchies, such as referencing distant function definitions. This may indirectly cultivate the model's ability to understand more complex structures [281].

From the perspective of training, Ma et al. [428] highlight that training with a mixture of code and text can significantly enhance LLM general reasoning capability, without almost any detriment to other aspects. Recent data-centric research [429] has further demonstrated that the outcomes achieved from equivalent mathematical training on CodeLLMs (*e.g.*, CodeLLaMA), markedly exceed the results compared to their base models or counterparts devoid of code training. We left the further examination of this intriguing yet unverified hypothesis to future works.

Building Math Models with Code. From a practical standpoint, CodeLLMs can serve as the foundation for developing sophisticated mathematical models. Llemma [315] utilizes CodeLLaMA [294] as the basis and is trained on a diverse mixture of math-related text and code to achieve remarkable mathematical capabilities, including the ability to use formal theorem provers. Meanwhile, DeepSeek-Math [322] is developed based on DeepSeekCoder [301], undergoes specialized training on web pages meticulously filtered for mathematical content. Both models exhibit superior performance on benchmarks such as MATH [404], STEM, and SAT, surpassing that of prior mathematics language models like Minerva [430]. Beyond the unanimous choice of conducting continual pre-training on CodeLLMs, comparative experiments from DeepSeek also suggest that code training can be a valuable prelude to math training, empirically showing the positive relationship between code training and mathematical capabilities.

Recently released InternLM-Math [431] incorporates code generated by proof assistants during its training process and innovatively interleaves coding processes within the problem-solving approach, presenting new state-of-the-art mathematical capabilities with the assistance of Python. To date, code learning has been demonstrated to play a pivotal and positive role across various facets of mathematical proficiency. A deeper understanding of its impact in this domain remains an area ripe for further investigation.

5.3 Alternative Formats for Solving NLP Tasks

Information Extraction. Information extraction (IE) aims to extract structural knowledge (*e.g.*, entities, relations, and events) from unstructured and/or semi-structured documents. However, due to the tasks having diverse output forms and requiring complex decoding strategies to post-process them into valid structures, relying solely on the plain text output of LLMs proves challenging for efficient modeling [432]. In the context of generative IE [433], these semantic structures can be smoothly converted into structured code, which plays a crucial role in processing various tasks in a unified schema [434]. CodeIE [435] shows that formulating text-to-structure IE tasks into structure-to-structure code generation with "code-style" prompting leads to superior performance compared to previous NL approaches. Additionally, CodeKGC [436] introduces a novel method in the construction of knowledge graphs by treating the generation of triples as code completion tasks and then develops schema-aware prompts to further leverage the structural knowledge inherent in code. Code4UIE [437] builds a framework for retrieval-augmented code generation, which utilizes Python classes to define various schemas under a unified format. GoLLIE [438] is developed through fine-tuning CodeLLaMA, by integrating Python code and comments into the input representation, thereby enhancing its zero-shot performance on unseen tasks.

Code as Intermedia Representation. In addition to the numerical reasoning (discussed in Section 5.1) and the aforementioned IE tasks, which can be formulated as code generation tasks in a relatively fixed paradigm, recent emerging research has applied code as a medium in more diverse scenarios, instead of text [439]. Early in the rise of LLMs, Madaan et al. [440] highlight that CodeLLMs could effectively represent desired graph predictions as code for use in structured commonsense tasks. Utilizing CodeX by few-shot prompting, they achieve performance surpassing that of general language models fine-tuned on the target task. Subsequently, this approach of constructing procedural processes based on code has also been applied to sequential decision making [441], story-based tasks [442], semantic parsing [443] and neurosymbolic understanding [444]. For graph scenarios, recently released InstructGraph [445] unifies the representation of graphs through a code-like format, eliminating the need for external specific encoders in graph reasoning and generation.

Beyond scenarios oriented toward linguistic structures, modular approaches based on code can also be applied to understanding both textual and graphical information. For example, VisProg [446] generates Python-like modular programs that involve off-the-shelf models or functions to facilitate visual reasoning. ViperGPT [447] creates programs for visual queries that take images or videos as arguments for execution, rather than traditional end-to-end methods with limited interpretability and generalization. Chen et al. [448] develop code-vision representations to assist in capturing visual structural information of varying granularity. More recently, Sharma et al. [449] use code to represent images for teaching models about more aspects of the visual world. Moreover, user queries received by robots can also be translated into executable actions through code generation, empowering intelligent robots with the capability for multi-step embodied reasoning [450]. In sum, integrating code promises to evolve into a novel paradigm for addressing diverse NLP tasks. However, this approach is not a free lunch, since utilizing code as an intermediary can introduce overhead (*e.g.*, defining a function) and consume more context windows. Further explorations in this vein are ongoing.

♢ ***Takeaway:*** *(1) The fusion of code generation and symbolic solvers with LLM reasoning represents a groundbreaking shift in tackling numerical tasks. Replacing natural language with executable code as the medium for reasoning, it not only overcomes lingering computation limitations but also enhances model interpretability and generalizability. (2) Although a theoretical foundation has yet to be established, the capacity of code training to enhance the mathematical abilities of LLMs has been empirically demonstrated and is gradually being accepted within the community. (3) Adopting code as intermediate representations can significantly elevate the efficacy and versatility of tackling diverse NLP tasks. By transcending traditional text-based processing, code-centric approaches, through the construction of a unified schema, can handle intricate and complex input and output forms that previous methods struggled with.*

6 Real-World Applications

Upon finishing our review of models, algorithms, and data, we shift our focus to discussing their real-world applications and current developments. Initially, our discourse centers around code processing and structured data, highlighting two main areas: (1) the direct application of language models for code in SE, aiming to bridge the gap between the NLP and SE communities [5]; (2) the exploration of utilizing CodeLLMs to augment data science research. Subsequently, we delve into hybrid scenarios, encompassing (1) the construction of agents through the application of code intelligence and (2) an emerging domain that leverages code intelligence to support AI4Science research.

6.1 Boosting Software Develop Workflows

Neural code intelligence is redefining the landscape of software development, steering it towards unprecedented levels of accessibility and automation across various scenarios.

Serving as Coding Assistants. It is an indisputable fact that coding assistants, exemplified by GitHub Copilot [451], are fundamentally transforming the landscape of software development [452]. Earlier attempts, such as aiXcoder[18] and Intellicode [198], have already demonstrated that deep learning models can assist with basic coding tasks. With the advent of the LLM era, emerging products like Code Whisperer [453], Tabnine [454], and Coze[19], which are built on commercial models, have begun to mature into established products. Concurrently, coding assistants based on open-source CodeLLMs are increasingly gaining popularity and are being utilized in multiple code downstream tasks. For instance, FauxPilot[20] and Starchat [306], respectively based on the CodeGen and BigCode models, can serve as locally hosted alternatives to Copilot. These versatile tools can be used for code generation, completion, repair, and even predicting the time complexity [455]. Additionally, when paired with a code interpreter (*e.g.*, ChatGPT Plugins[21]), chatbots are capable of aiding users in building temporary programs within conversations. Furthermore, interacting with users in the form of extensions has also started to become popular, with CodeGeeX [290] already adapting itself to IDEs such as Visual Studio Code and JetBrains. Empirical research has illuminated the profound impact of these AI coding tools on developer efficiency. For instance, A prior study [456] has indicated that the majority of GitHub Copilot users have experienced enhanced programming efficiency and improved code quality when coding with it. Complementarily, Peng et al. [457] quantify this enhancement in productivity, demonstrating a notable acceleration in task completion when developers employ Copilot.

As interest grows, attention is also directed towards the multifaceted performance of these assistants. Beyond evaluating the generated code itself [458, 459], researchers have also recognized the influence of prompts on their operation [460], the robustness issues faced when tackling varying scenarios [461], and their effectiveness across specific PLs [462] or NLs [463]. Moreover, the quality of generated

18. https://github.com/aixcoder-plugin
19. https://coze.com/
20. https://github.com/fauxpilot/fauxpilot
21. https://openai.com/blog/chatgpt-plugins#code-interpreter

codes [464] is another aspect that should not be overlooked. Fu et al. [465] have revealed common weakness enumeration and advocated for meticulous review processes to mitigate risks associated with automated code generation. So, we need a dual focus on enhancing productivity while safeguarding against vulnerabilities in code generation.

Streamlining Software Development. Compared to having models write or complete code of different granularities for you, allowing them to take over the entire software development process is also becoming a tangible reality. ChatDev [466] represents a virtual "software company", functioning through an array of intelligent agents assuming roles such as programmer, code reviewer, tester, and designer. Collectively, these agents establish a multi-agent ecosystem, facilitating comprehensive software development processes. AgentCoder [467] also implements a multi-agent system, assigning different roles to models, showing that role-playing strategy not only alleviates the need for manually crafted test cases but also achieves better outcomes than self-refinement methods [468, 469, 470].

After that, Qian et al. [471] enrich these agents with experience, encouraging them to accrue shortcuts from previous experience to avoid inefficient attempts or repetitive errors. This strategy enables the agents to tackle software engineering tasks within their interactions more efficiently, leading to improved automation and efficiency for unseen tasks. Beyond software kernel design, graphical design also falls within the scope of interest [472]. The newly released Design2Code [473] aims to automate front-end development by converting visual designs into code implementations.

As for leveraging code intelligence within GitHub, CodeAgent [474] suggests the use of external tools (*e.g.*, Format Checker) for repo-level code generation. Meanwhile, beyond using pre-defined function set [475, 476], GitAgent [477] autonomously integrates repositories in response to user queries, thereby expanding its toolkit. Regarding software maintenance, the quality of code documentation is directly linked to development efficiency [478]. To automate this, RepoAgent [479] has been proposed for documentation generation. It utilizes AST analysis to understand code structure and discern reference relationships within files, providing a contextual perspective for LLMs to assist in identifying the functional semantics to support fine-grained code documentation generation. For more reliable testing, Meta has introduced TestGen-LLM [480], which aims to improve existing human-written tests for automated unit test generation. The majority of its improvements have landed in industrial production. Meanwhile, Google has advocated for the automation of resolving review comments within daily development workflows [481].

6.2 Facilitating Data-Driven Decision-Making

Neural code intelligence has emerged as a transformative force in data-driven decision-making by unlocking new potentials in more accessible database interactions and streamlining data science processes.

Democratizing Database Interactions. Previous efforts primarily emphasize elaborate model design and optimization tailored for specific formal languages, which lack the innate capability to effectively adhere to instructions. Large language models implicitly contain extensive world knowledge, which brings some basic abilities to interact with users. However, the inherent autoregressive characteristics [210] of LLMs make it challenging for them to accurately retain and recall data, especially when facing large-scale private databases. Under such circumstances, it is necessary to equip LLMs with the capabilities to automatically interact with external databases. The ultimate challenge lies in the precise generation of formal calling languages (*e.g.*, SQL) based on the given query, and it is in this aspect that CodeLLMs truly excel.

With the advent of CodeLLMs [294, 301], recent endeavors have also focused on facilitating widespread access to database interactions, aiming to bridge the retrieval process [482]. Early attempt like Binder [483] leverages CodeX to generate the programming language to combine the external knowledge bases. [484] proposes the optimized prompt design to boost the performances of Text-to-SQL tasks. In addition, SQL-PaLM [485] targets powering off-the-shelf LLM with SQL-specific optimization. They represent two common types of practices in this direction: 1) prompting Code-LLMs; and 2) post-finetuning LLMs with SQL optimization.

To better measure the LLM performances in tackling database-related scenarios, the evaluation benchmarks are gradually improving [486]. The overarching trend is shifting from static, single-turn interactions constrained by limited domains towards dynamic multi-turn conversations encompassing diverse domains. At the early stage, benchmarks, such as ATIS [487], tackle domain-specific settings (*e.g.*, flights) with a limited number of schema. Following them, Spider [164], Sparc [165], Cosql [166], WikiSQL [488] and WikiTableQA [489] etc. are proposed to cover plenty of domains and over thousands of schema. Powered by the advent of LLMs, some challenging and comprehensive benchmarks arouse wide interest. Recent work [490] highly stresses pushing the boundaries of LLMs and making LLMs serve as a functional interface. InterCode-SQL [491] tackles the multi-turn interaction with the database, pushing LLM to more practical scenarios. To sum up, the potential of self-repairing abilities and multi-turn interactions with databases are highly valued in the current landscape.

Accelerating Data Insight Discovery. Data science entails the extraction of insights from data [492], and has evolved to be a pivotal component of decision-making and knowledge discovery over the past few years [493]. Earlier attempts in this domain included methods for querying tabular data in natural language [494] or generating Pandas codes for data analysis [199]. Recent research has progressed to handling real-world data science notebooks, which are more complex than mere code generation, as computational notebooks often mix codes, text, figures, and execution results [495].

Regarding models specifically designed for such scenarios, after the initial construction of JupyT5 [163] as a data science assistant, PaChiNCo [496] emerges as a 62B CodeLLM based on PaLM, specifically designed for Python data science. Beyond its substantial size, PaChiNCo is trained under massive multi-modal contexts, such as existing notebook cells, corresponding execution states, and previous interaction turns. These rich contents ensure the model aligns with the specific peculiarities of computational notebooks, accommodating more diverse elements.

As for benchmarks that more closely align with real-world applications, particularly those concerning data wrangling tasks to process raw data and exploratory data analysis, DS-1000 [327] stands out as an example. It comprises high-quality problems sourced from StackOverflow, including use cases involving NumPy, Pandas, TensorFlow, PyTorch, etc. In terms of evaluation, what sets it apart from conventional NL2Code tasks is not just the emphasis on functional correctness; it also imposes additional constraints, such as requiring the generated code to include specific APIs/keywords to ensure solutions are efficient and aligned with the query [497]. ARCADE [496] presents another challenging benchmark, with a focus on Pandas. Its cases are derived through repurposing high-quality portions of previous benchmarks [162, 163] and collecting interactions between professional data scientists and coding assistants. ARCADE features multiple rounds of code generation within the same notebook, emphasizing the iterative nature of real-world data science work. Research in this domain continues to thrive, with scholars also focusing on handling sophisticated DataFrames [498], tackling the challenges of complex data visualization [499, 500], and addressing hallucinations in conversations [501].

6.3 Building Code-Empowered Agents

A perennial topic in machine intelligence is to develop agent systems [502], such as robots, that can perform complex tasks requiring interaction with real-world environments. Recent breakthroughs in LLMs have notably enhanced the task-solving capabilities of AI agents. Typically focused on natural language generation, LLMs offer immense potential [324, 503, 323], yet the inherent ambiguity of natural language can sometimes render the precision and efficiency of planning and interaction [504]. To address this, emerging research advocates the integration of code as the de facto standard of agent systems [505, 506, 507, 508, 509]. The following discussions will delve into these advancements.

Augmenting Robotics System. Building robots capable of manipulating objects to accomplish diverse tasks in physical environments poses a significant challenge. Code generation enables robots to seamlessly combine environment perception [510, 511, 450], feedback loops [512, 513, 514], and parameterized actions [515, 516] into reasonable policy, effectively translating complex tasks into executable solutions. Generating executable actions is fundamental for robots; ProgPrompt [512] pioneered the use of GPT-3 to generate code-based actions, offering better environmental grounding than free-form text. Robot tasks often involve multiple sub-tasks and conditional loops [36], where code-based planning naturally excels over natural language due to its structured advantage. For instance, Liang et al. [510] explored utilizing LLM to write robot policy code, incorporating complex feedback loops and primitive API calls. Following studies concentrated on integrating visual input [516, 517, 518], refining code through environmental feedback [514, 519] or reinforcement learning [517], and continually expanding agent's skill library [515]. Faced with difficulties in collecting human operation trajectories, a series of studies explored leveraging LLMs for code generation to automatically synthesize diverse robot data [520, 514]. Additionally, Ha et al. [519] proposed enhancing data quality by simultaneously generating robot operations and code snippets to verify task success. Another line of research innovatively applies LLMs in reinforcement learning to program reward functions [521, 511, 522], guiding robots with greater efficiency than human experts. Text2Reward [523] demonstrates that LLM can produce interpretable dense reward functions for robotic manipulation tasks and allow iterative refinement with human feedback.

Elevating Intelligent Automation. Beyond the extensively discussed robotics, the paradigm of interacting with the environment through code generation also plays a pivotal role in elevating the capabilities of various intelligent automation systems. In digital device assistants, code serves as the natural choice for controlling interactions between the agent and digital environment [524, 300]. Gur et al. [525] designed a web agent that acts on websites via generated Python programs by PaLM [34]. CodeACT [526] introduced unifying agent actions through code and integrated LLM with Python interpreter. GAIA [527] constructed a benchmark for general AI assistants, encompassing tasks involving LLMs answering questions through programming. Recently released OS agent OS-Copilot [528] generates executable code for operating computers, enabling humans to interact with operating systems through natural language instruction. Contemporary progress in gaming agents has explored the use of code for perceiving the environment and facilitating action [529, 530]. For instance, Vogayer [531] employs programming for interactions with Minecraft and maintaining an ever-growing skill library of executable code, aiming to foster embodied lifelong embodied agents. Beyond these, code models are also widely applied in autonomous driving [532, 533, 534], automated data processing [535, 536] and multimodal tasks [447, 537].

Although code interfaces offer a superior method for planning and interacting with the environment for agents compared to natural language interfaces, these approaches still largely rely on predefined APIs or skills. Intelligent systems require ongoing exploration to boost their generalization across diverse tasks in the real physical world.

6.4 Advancing AI4Science Research

The progress in scientific fields can be greatly facilitated by advancements in code generation and the use of symbolic languages, particularly in the domains of mathematical proof, chemistry, and biology. These advancements play a crucial role in driving innovation and pushing the boundaries of scientific understanding. The following paragraphs will outline these contributions in each domain.

Automating Theorem Proving. Formal theorem proving is a discipline that necessitates finding proofs for given conjectures articulated in structured, formal statements governed by the principles of logic. Traditional formal theorem proving called upon human experts to meticulously convert mathematical concepts into formal statements, which could then be verified using interactive theorem provers (ITP) like Isabelle [538] and Lean [539]. Each formal statement is similar to a code statement, and ITPs are "compilers" specifically designed for math proofs. This process is greatly accelerated by utilizing formal statements generators that generate single-step proofs that ITPs then verify. A premise is often directly selected based on language modeling statistics [540, 541, 542], while Leandojo [543] uses a retrieval-augmentation generation pipeline where LLM directly generate proof step based on retrieved premises. Each step is then checked for accuracy by ITPs before progressing to the next, with the ultimate objective being the completion of the proof. To optimize this procedure, sophisticated search algorithms are employed that identify and develop promising premises with the potential to culminate in successful proofs. [544, 545]. A distinctive work in this field is AlphaGeometry [546], which uses a language model to generate auxiliary construction for geometric proofs and then solve the newly constructed problem with symbolic solver [547]. Furthermore, recent research [548, 549, 550, 551] has explored the use of language models with potent code generation capabilities to directly produce entire proofs that could be further refined by ITPs.

Code generation has demonstrated its potential in proving existing mathematical theorems yet recent works demonstrate its potential in solving open mathematical questions. AlphaTensor [552] tackles matrix decomposition by producing tensor decomposition statements and employing tree search to identify the most efficient solution. Similarly, FunSearch [553] addresses combinatorial problems such as cap sets and online bin packing through program generation and optimal selection.

Catalyzing Biochemistry Discoveries. Code generation has emerged as a transformative force in biochemistry research, significantly enhancing the efficiency and effectiveness of scientific investigations in the field [554]. This can be largely attributed to the wealth of open-source tools and code packages now available, including tools for processing biochemical sequences [555, 556, 557], accessing databases [558, 559, 560], and analyzing biochemical properties [561, 562]. To this end, Dias and Rodrigues [563], Bran et al. [564] pioneer in using LLMs to generate code APIs that use these tools for automating the chemical research pipeline. Ma et al. [565] have proposed retrieval APIs to expedite the analysis of protein sequences. Tang et al. [357] specifically designed to assess the capability of LLMs in generating bioinformatics code, marking a significant step towards standardized evaluation in this domain. In drug discovery, innovative approaches [566, 567, 568] to employ tool-using language models for automatically editing and optimizing molecular structures for therapeutic purposes – a crucial phase in drug development. Beyond the generation of APIs for existing biochemical tools, Steiner et al. [569] and Rauschen et al. [570] have also introduced novel chemical programming languages designed to automate the synthesis of chemical compounds. These progresses represent a leap forward in our ability to program and execute complex synthetic chemistry with precision and scalability.

◊ ***Takeaway:*** (1) *Coding assistants have revolutionized software engineering workflows by significantly enhancing programming efficiency and code quality. Further, the ongoing evolution towards fully automated software development ecosystems represents a leap towards utilizing intelligent code agents to alleviate humans from labor-intensive development tasks.* (2) *The evolution of the CodeLLMs has significantly broadened the scope of database interaction, thereby unlocking the potential of multi-turn retrieval in more generalized domains. Further, the rise of code intelligence has also motivated more research into automating and accelerating real-world data science workflows.* (3) *The code-centric paradigm orchestrates perception, decision-making, action, and feedback for intelligent agents to tackle complex tasks in real-world environments. With their potent reasoning capacity and efficient interaction, these models establish a strong foundation for developing agent systems capable of navigating the highly variable physical world.* (4) *Code models that can wield mathematical proof language and scientific tools have revolutionized the AI4Science field by advancing towards autonomous scientific discovery, matching human mathematicians in writing proofs, and chemists in analyzing compounds. The current languages used for AI4Science often differ from typical PLs; therefore, adapting code models to novel symbolic languages is essential for their further application in science.*

7 Opportunities and Future Directions

Thus far, we have extensively reviewed and discussed the advancements in code intelligence, encompassing tasks, models, and applications, aiming to provide a comprehensive view and bring interested researchers up to speed with this field. In what follows, we highlight several promising directions for future research that are ripe for contribution.

7.1 Beyond Transformer Architecture

Since the rise of transformer architecture [26], it has maintained a dominant position in NLP [176], and most of the models we discuss in this paper are also based on this. Nowadays, with the emergence of diffusion models for controllable text generation generation [571, 572, 573], it has also gradually been applied in the NL2Code field. CodeFusion [574] has recently pioneered the application of diffusion models in code-related tasks, achieving stunning results on tasks for Python, Bash, and conditional formatting [575], surpassing the performance of LLMs such as StarCoder and CodeT5+ with only a 75M size. As we have discussed in Section 2, utilizing specially tailored architectures or modeling for specific tasks separately remains a path worth exploring.

7.2 Renaissance in Utilizing Code Features

The most notable difference between code and natural language is its structured nature, which was widely leveraged in the pre-training [181], fine-tuning [61, 576], evaluations [577], explainability [62] of models in the Pre-LLM era. Yet, in the current landscape dominated by LLMs, the utilization of structural information has diminished. We believe the main reasons are the incompatibility of most LLM training pipelines with modalities other than text tokens, and the cost required to extract code features (such as data flow graphs and abstract syntax trees) from massive training data. We hold the view that finding appropriate strategies to integrate structural information into the CodeLLM training process is worth exploring. Beyond training, as previously discussed in Section 3.3 we believe that structural information can still help us better understand the behavior of models. Recently we have also witnessed some researchers starting to utilize lexical properties for enhanced code generation [578] or structural information to assess the impact of code complexities on program-aided reasoning [422]. We are convinced that a revitalization of using code structure will significantly contribute to the ongoing advancement of code intelligence.

7.3 Repo-Level Code Understanding and Generation

Following the surge in the popularity of CodeLLMs, their applications have primarily focused on individual functions or files. However, in real-world software engineering, developers often need to consider the relationships between different files and functions within a code repository. When extending the tasks discussed in Section 2.2 to broader scenarios, such as completing or repairing a code snippet within a repository [579], The capability of models to invoke variables and functions from other files remains underexplored. This may necessitate the integration of code understanding, cross-file dependencies understanding, and retrieval-augmented generation [580]. Also, given the potential complexity of code repositories, this may require research into how to expand CodeLLMs' efficient context window [581] and their ability to understand cross-file dependencies. Therefore, repository-level code generation and understanding represent a direction of both practical and research significance, with production environments also serving as a sustainable and challenging testbed for future models.

7.4 Towards Holistic and Reliable Evaluations

Reliable and comprehensive benchmarking is a perennial topic in language model research, and the same applies to evaluations of language modes for code [114, 582]. Current mainstream evaluation methods primarily rely on code execution, during which the security, diversity, and readability of the model-generated code are often overlooked [583]. Additionally, researchers have pointed out that some benchmarks, represented by APPS [354] derived from competition platforms like Codeforces, are likely to have frequently appeared in public repositories [36], consequently leading to models "remembering" the potential solutions to these problems during the pre-training phase. The hand-written HumanEval dataset also faces inevitable data leakage or contamination issues [584] as the training corpus expands. In pursuit of more reliably evaluating CodeLLMs' performance, and to consider the naturalness, robustness, and lexical diversity of the generated code, fair and dynamic comprehensive benchmarking awaits further exploration.

7.5 Interleaved Planning and Code-Driven Reasoning

As discussed in Section 5.1, Program-aided Language Models [399] and Program of Thoughts [400] have demonstrated efficacy in numerical-related tasks, significantly outperforming some NL-centric paradigms [335]. However, the success of such a paradigm hinges on the premise that a problem can be reliably decomposed into multiple lines of code for resolution. Compared to NL-centric reasoning, program-aided strategies are anticipated to push the boundaries of human intelligence. Mirroring the human approach to tackling complex challenges, more demanding problems (*e.g.*, Olympic competition problems [546]) require ongoing planning and reasoning. Each phase in such scenarios necessitates iterative planning, taking into account the problem at hand and the current state, while concurrently generating code to aid in the solution part. This synergistic blend of interleaved planning and program-aided reasoning provides a strategic advantage that is worthy of exploration.

7.6 A Closer Look at Tokenizer Dynamics

Tokenization has historically played a pivotal role in language modeling [585], where the choice of tokenizer can significantly influence a model's downstream performance [586] and multilingual capabilities [587]. Recent studies have also identified its substantial impact on the generalization capabilities of LLMs [588, 589]. Unlike natural languages, code operates under more rigid syntactic rules and structures, with meanings heavily reliant on the precision of these elements. Earlier research has uncovered that tokenization granularity exerts a non-trivial impact on code-related tasks [577]. So, we have reason to doubt that: when processing code with general tokenization methods like BBPE [590], as employed by models such as CodeLLaMA, there is a risk of disrupting this structure due to over-tokenization. Hence, determining an effective approach to tokenization without compromising code structure and semantics poses a challenge. One potential research direction involves developing tokenizers capable of recognizing and preserving code structures (discussed in Section 2.1)–such as keywords, function definitions, and control flow statements—or exploring better strategies that grasp a deeper understanding of code semantics.

7.7 Efficient Methods for CodeLLMs

As the development of CodeLLMs progresses, beyond striving for performance enhancements, the pursuit of model efficiency emerges as another critical direction for in-depth investigation. As for training, whether starting from scratch or conducting additional training on code using general LLMs, the process proves exorbitantly expensive. Although parameter-efficient methods [591] like prefix-based strategies [592, 593, 594] and LoRA [595] significantly reduce resource consumption, applying them directly to language models for code [596] has been shown to noticeably impact the performance of code-related tasks [597]. Thus, identifying new efficient training techniques for code is worth exploring, and it necessitates finding the most suitable strategies and trade-offs between cost and performance across models of different scales [598]. For deployment, due to the prevalent state-of-the-art CodeLLMs being powerful yet cumbersome, researchers have begun to emphasize the application [599] and optimization of CodeLLMs for running within resource-constrained environments. As discussed in Section 4.1, this particularly includes enabling these models to function offline on consumer-level devices without the reliance on GPUs [303, 304]. Additionally, to satisfy more diverse demands, CodeLLMs are faced with the strategic decision of scaling up [600, 601] or scaling out [602]. Alternatively, acceleration can also be achieved through methods such as structured pruning [603] and distillation [604] akin to approaches employed in general LLMs.

7.8 Further Expansion of Multilingual Capability

“Nobody should call themselves a professional if they only knew one language.” — Bjarne Stroustrup. Multilingualism has long been a staple in NLP research, with LLMs proving capable of holding multilingual capabilities [605, 606]. However, in the realm of code intelligence, the exploration of mastering multiple programming languages is a relatively recent development. Although there has been research targeting multilingual scenarios [290], significant performance variations across different programming languages by CodeLLMs are evident from various leaderboards (*e.g.*, Big Code Models Leaderboard[22]). This disparity is largely attributed to the uneven distribution of corpora which is dominated by popular languages like Python and Java. Consequently, collecting more premium data for less prevalent languages (*e.g.*, TypeScript, Kotlin, Scala), exploring data augmentation [607], conducting specialized training, distilling language agnostic representation [608] or even transferring knowledge between different languages [609, 610, 611], presents a worthwhile direction for research. Furthermore, multilingual models have also been shown to be more robust to prompt perturbation and excel in code summarization [355], potentially contributing to an enhanced overall capability of CodeLLMs. It’s also noteworthy that current mainstream multilingual benchmarks [161] are mostly conversions from Python datasets, overlooking the unique characteristics of different languages. Hence, developing new benchmarks that cater to specific language features is another avenue worth exploring.

7.9 Copyright Challenges Faced by Coding Assistants

With the expansion of the open-source community and the rise of coding assistant tools (discussed in Section 6.1), a series of ethical and security concerns regarding the distribution of source code have arisen, such as unauthorized use of copyrighted code, distributions without proper licenses, or exploitation of code for malicious purposes. To circumvent potential legal and copyright disputes, one viable strategy is the watermarking of source code protected by licenses such as GPL. Furthermore, the use of copyrighted content detection in CodeLLMs’ training corpora [612] can also safeguard against the unintended output of such code. Notably, watermarking and detection for code diverge from that for natural language due to the impact of obfuscation on the semantics of variable names, presenting an intriguing area for further investigation. Unlike watermarking [613] and content detection [614] applied to natural language, code presents unique challenges. As previously discussed, the semantics of variable names in code can be significantly affected by obfuscation, making it a more challenging and worthwhile direction for further investigation.

8 Conclusion

In this paper, we present a systematic review of the entire evolutionary trajectory of code intelligence, offering a comprehensive examination from the nascent application of deep neural networks on source code to the breakthroughs made in the LLM era. Throughout this process, we have delved into the interconnections among research across different periods, engaging in detailed discussions and analyses centered on the paradigm shifts in models, tasks, and applications. Moreover, we explore the synergies between code learning and other facets of machine intelligence, along with both long-standing and emergent applications in the real world. Bearing in mind the developmental pathway we have witnessed and insights garnered from our discussions, we also identify several promising directions for future research in code intelligence. Given the concurrent advancements in language models and the evolving needs of software development, we are confident that this domain will continue to flourish in the forthcoming years. It is our aspiration that the literature review, discussions, experiences, and resources provided in this survey paper will boost the community’s future research.

22. https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard

Appendix A Additional Resources

A.1 Reading Lists

In the NCISurvey project related to this paper, we have prepared a curated reading list for our readers, encompassing a diverse array of research areas within the realm of code intelligence.

A.2 Recreating the Figures

We used the template from LLMsPracticalGuide maintained by Yang et al. [615] to construct Figure 3. Our primary criteria for determining the placement of models on specific branches of the tree were based on the release times of the models mentioned in Sections 3 and Section 4, as well as the relationships between the models. Relevant resources will be uploaded to our project for further reference and exploration.

Appendix B More Benchmarks

Given the constraints on space, Table 1 and Table 5 feature only a subset of the representative tasks in code downstream applications and NL2Code. To offer a more thorough overview, Table 6, 7, 8 and 9 systematically present a more exhaustive review on datasets, organized by each specific task category.

Appendix C Detailed of CodePTM Training Objects

Due to space limitations, we adopt abbreviations for the training objectives of CodePTMs in the overview presented in Table 2, section 3.1. Here we provide detailed explanations for them in Table 10.

APPENDIX D RELATED RESEARCH

Before the rise of employing transformer-based models in code intelligence, Han et al. [21] evaluates the performance of eight code embedding models on classic code-related tasks. During the period when pre-trained language models are dominant in NCI research, Wu et al. [616] categorize CodeLMs from the perspectives of code structures. Xu and Zhu [617] revisit the training processes and downstream applications of CodePTMs, and Xu et al. [286] conduct a systematic evaluation of mainstream models in this sphere. Meanwhile, several studies [618, 619] have discussed the application of CodePTMs from the perspective of software engineering. Amidst the burgeoning trend of LLMs, Zan et al. [4] provides an empirical summary of the field's development from the standpoint of NL2Code. Hou et al. [620] and She et al. [621] discuss the application of CodeLLMs in SE, along with the opportunities and challenges. In recent studies, Zhang et al. [5] conduct a retrospective study of language models for coding from the unified viewpoints of NLP and SE. Wan et al. [218] delve into an exploration of benchmarks and toolkits for neural code intelligence. For the synergies between code learning and AI agents, Yang et al. [508] investigate the role of code corpora in augmenting the capabilities of LLM-based agents.

APPENDIX E DATA SOURCES

E.1 Evaluations

The primary data sources for Figure 5 are BigCode's Evaluation Harness [622] and the BigCode Models Leaderboard. Similar evaluation results can also be found at OpenCompass [623]. As for the results reported in the Figure 6, we access GPT-3.5-Turbo and Claude-1 through paid APIs. For open-source LLMs, we utilize CodeLLaMA-Instruct-13B and LLaMA2-Chat-13B under the greedy search setting. Both GSM8K and MATH evaluations are conducted with 3-shot promptings from Gao et al. [399]. WikiSQL [488] and FOLIO [624] are evaluated under the 1-shot prompting, modified from Xu et al. [302].

E.2 Publication Trends Data and Milestones

The construction of Figure 1 was achieved through the use of the ArXiv advanced search[23], with the calculation based on an exact match by querying a group of keywords (*e.g.*, code representation, code generation) in the title or abstract. Detailed information on this is provided within the NCISurvey project.

As for the selection of milestones featured in Figure 2, Qiushi Sun, Zhirui Chen, and Zhangyue Yin each selected what they considered to be representative milestones based on their modeling approaches, release time, the citations of papers/preprints, and their influence within the community. The final selection was made by identifying the intersection of their choices.

23. https://arxiv.org/search/advanced

Beyond statistics, we can observe that the publication venues for research papers on code intelligence have undergone a significant transformation over the past decades. In earlier research, publications were primarily confined to ACM/IEEE Conferences like ASE and ICSE, or journals like IEEE TSE. However, with the rise of the neural approach, research began shifting towards machine learning conferences, such as NeurIPS. Subsequently, with the rapid advances in language modeling, code-related topics have also become frequent subjects at NLP conferences, exemplified by the *ACL Conferences. These changes in publication venues reflect a blurring of the boundaries between previously distinct research areas, demonstrating the interdisciplinary nature of code intelligence research today.

ACKNOWLEDGEMENT

Thanks to everyone who has contributed to this paper. We extend our gratitude to Xuesong Lu for meticulously reviewing our paper and providing helpful feedback, as well as assisting us in identifying and incorporating previously overlooked literature into our study. We warmly welcome the community to utilize all resources provided in this survey for research, educational, knowledge-sharing, and domain background introduction purposes. *The authors of this survey paper retain the copyright of the figures/tables included herein, and any use of these materials for publication purposes requires authorization from the survey authors.*

TABLE 6: A more comprehensive collection of code-related benchmarks extended from Table 1. "# PLs" denotes the number of programming languages each benchmark covers.

Task	Dataset	Date	# PLs	Description	Eval. Metric
Clone Detection	POJ-104 [16] [link]	2014	1	a program classification dataset of 52K C/C++ programs	Acc.
	BigCloneBench [108] [link]	2015	1	a clone detection dataset of 8M Java validated clones	F1 score
	CLCDSA [109] [link]	2019	3	a cross-language clone dataset dataset of more than 78K solutions	F1 score
Defect Detection	CGD [625] [link]	2018	2	a dataset focuses on two types of vulnerabilities in C/C++	F1/Precision
	Draper VDISC [626] [link]	2018	2	a vast dataset of open-source functions in C/C++	F1
	SySeVR [627] [link]	2018	2	a dataset of 126 types of vulnerabilities in C/C++	F1
	Devign [78] [link]	2019	1	a dataset of vulnerable C functions	F1/Acc.
	GREAT [85] [link]	2019	1	a dataset extracted from the ETH Py150 dataset	Acc.
	MVD [628] [link]	2020	2	a dataset that contains 181K pieces of code from 33K C/C++ programs	F1
	ReVeal [629] [link]	2020	1	a vulnerability dataset curated from two real-world projects (Chromium and Debian)	F1
	BigVul [630] [link]	2020	2	a large C/C++ code vulnerability dataset from open-source Github projects	F1
	D2A [631] [link]	2021	2	A dataset built for AI-based vulnerability detection methods	F1
	PyPIBugs [88] [link]	2021	1	A dataset retrieved form the most downloaded packages in the Python package index	F1
	CVEfixes [632] [link]	2021	27	a automate-collected dataset from Common Vulnerabilities and Exposures records	F1
	CrossVul [110] [link]	2021	> 40	a dataset of 27K files containing vulnerabilities	F1/Acc.
	DiverseVul [111] [link]	2023	2	a dataset of 19K vulnerable C/C++ functions and 330K nonvulnerable functions	F1/Acc.
	VulnPatchPairs [633] [link]	2023	1	a dataset contains 26.2K C functions	F1/Acc.
	VulBench [634] [link]	2023	1	a dataset offers a blend of CTF challenges and real-world CVE vulnerabilities	F1
Code Repair	Defects4J [] [link]	2014	1	a database and extensible framework providing real Java bugs	Mean
	ManyBugs [635] [link]	2015	1	a benchmark consisting of 1K defects in 15 C programs	Acc.
	BugAID [636] [link]	2016	1	a benchmark build through mining 105K commits from 134 JavaScript projects	Acc.
	DeepFix [83] [link]	2017	1	a set of 6971 erroneous C programs written by students for 93 programming tasks	Acc.
	Codeflaws [637] [link]	2017	1	a collection of C programs with 3.9K defects	Acc.
	QuixBugs [638] [link]	2017	2	a multi-language benchmark (40 bugs in both Python and Java)	Pass Rate
	Bugs.jar [639] [link]	2018	1	a benchmark consisting of 1.1K bugs and patches	Acc.
	Bears [640] [link]	2019	1	an extensible bug benchmark for automatic repair studies in Java	Acc.
	BugsJS [641] [link]	2019	1	a benchmark of 453 real JavaScript bugs	Acc.
	BugSwarm [642] [link]	2019	2	a benchmark of 3K fail-pass pairs, in Java and Python	Acc.
	ManySStuBs4J [643] [link]	2019	1	a dataset of 153K single statement bugfix changes mined from Java projects	Acc.
	Refactory [644] [link]	2019	1	a dataset consists of almost 1.8K real-life incorrect Python program submissions	Acc.
	Review4Repair [645] [link]	2020	1	a dataset consists of 55K code reviews and associated code changes	Acc.
	BugsInPy [646] [link]	2020	1	a benchmark consists of 493 real bugs from 17 real-world Python programs	Acc.
	TFix [647] [link]	2021	1	a benchmark consists of 52 different error types reported by a popular static analyzer	Acc.
	Megadif [648] [link]	2021	1	A dataset of 600K java source code changes categorized by different size.	Acc.
	SSB/TSSB [649] [link]	2022	1	a collection of over 9M/3M general single statement.	Acc.
	FixJS [650] [link]	2022	1	a dataset containing bug-fixing information of 2M commits.	Acc.
	TypeBugs [651] [link]	2022	1	a benchmark dedicated to repairing type errors in Python.	Precision
	xCodeEval [652] [link]	2023	11	an executable dataset of 450K small buggy/fixed program pairs	Pass Rate
	RunBugRun [653] [link]	2023	8	an executable multilingual benchmark consisting of 25M document-level coding examples	Acc.
	HumanEvalPack [307] [link]	2023	6	expanding the HumanEval benchmark to 3 coding tasks across 6 languages	Acc.
	ErrorCLR [87] [link]	2023	2	a multilingual benchmark of similar buggy programs	Pass Rate

TABLE 7: A more comprehensive collection of code-related benchmarks extended from Table 1. “#PLs.” denotes the number of programming languages each benchmark covers (Cont’d). MRR (Mean Reciprocal Rank) indicates the average rank of correct answer choices. FRank (First Rank) measures the proportion of correct answers ranked first. CR (Compilation Rate) denotes the percentage of code snippets that compile successfully. NIM (Next Identifier Match) is the accuracy in predicting the next identifier or variable name. ISM (Identifier Sequence Match) measures the coherence of generated identifier sequences. PM (Prefix Match) means the alignment of generated code snippets with expected prefixes.

Task	**Dataset**	**Date**	**# PLs**	**Description**	**Eval. Metric**
Code Search	CodeSearchNet [113] [link]	2019	6	a multilingual dataset of 6M functions and query-like natural language	MRR
	AdvTest [114] [link]	2021	1	a Python code search dataset filtered from CodeSearchNet	MRR
	WebQueryTest [114] [link]	2021	1	a test set of Python code search including 1K query-code pairs	Acc.
Code Translation	GeeksforGeeks [101] [link]	2020	3	a test set composed of 852 parallel functions for code translations	Acc./BLEU
	CodeTrans [114] [link]	2021	2	a C#/Java code translation dataset collected from several public repos	Acc./BLEU/CodeBLEU
	Avatar [654] [link]	2021	2	a collection of 9.5K programming problems and solutions written in Java and Python	Acc./BLEU/CodeBLEU
	CoST [115] [link]	2022	7	a multilingual Code Snippet Translation dataset	BLEU/CodeBLEU
	XLCoST [655] [link]	2022	7	a benchmark containing fine-grained parallel data from 7 programming languages.	BLEU/CodeBLEU
	xCodeEval [652] [link]	2023	11	an executable multilingual benchmark consisting of 25M document-level coding examples	Pass Rate
	G-TransEval [656] [link]	2023	5	a benchmark suite of 400 code translation pairs between 5 PLs, categorized into 4 levels	Acc./BLEU/CodeBLEU
	CodeTransOcean [116] [link]	2023	45	a large-scale code translation benchmark with three novel multilingual datasets	EM/BLEU/CodeBLEU
Code Retrieval	StaQC [657] [link]	2018	2	a dataset of around 148K Python and 120K SQL domain question-code pairs	MRR
	DeepCS [146] [link]	2018	1	a large scale codebase collected from GitHub	FRank/SuccessRate/ Precision
	CoNaLa [658] [link]	2018	1	a dataset automatically crawled from Stack Overflow	BLEU
	CodeSearchNet [113] [link]	2019	6	a multilingual dataset of 6M functions and query-like natural language	MRR
	CosBench [659] [link]	2020	1	a dataset that consists of 1K projects and 52 code-independent natural-language queries	Precision/Precision/MRR
	SO-DS [660] [link]	2020	1	a dataset that consists of 12K snippets of Python	MRR/Recall
	FB-Java [661] [link]	2020	1	a dataset consisting of 24K repositories with 4.6M functions in Java	MRR/SuccessRate
	AdvTest [114] [link]	2021	1	a Python code search dataset filtered from CodeSearchNet	MRR
	WebQueryTest [114] [link]	2021	1	a test set of Python code search of 1K query-code pairs	Acc.
	CoSQA [662] [link]	2021	1	a dataset of web queries for code search and question answering.	Acc.
Code Completion	GitHub Java Corpus [2] [link]	2013	1	a giga-token corpus of Java code from a wide variety of domains	EM/Acc./Edit sim
	Py150 [117] [link]	2016	1	a corpus of Python programs from GitHub	EM/Acc./Edit sim
	JS150 [117] [link]	2016	1	a corpus of JavaScript programs from GitHub	EM/Acc./Edit sim
	DotPrompts [663] [link]	2023	1	a dataset of real-world open-source Java projects completion with their environments	CR/NIM/ISM/PM
	LCC [118] [link]	2023	3	a benchmark that focuses on code completion with long code context	EM/Edit Sim
	RepoBench [664] [link]	2023	2	a benchmark tailored for evaluating repository-level code autocompletion systems	EM/Edit Sim
GitHub	unamed [141] [link]	2017	1	a dataset of 509K labeled diff files of Java	Acc.
	CommitGen [121] [link]	2017	4	a multilingual dataset collected from open source projects on Github	BLEU
	NNGen [140] [link]	2018	1	a cleaner version of CommitGen	BLEU
	PtrGNCMsg [665] [link]	2019	1	a dataset of diffs and manual commit messages from Java projects in GitHub	BLEU/ROUGE
	CoDiSum [142] [link]	2019	1	a cleaner version of Jiang and McMillan [141]’s dataset	BLEU/METEOR
	ATOM [666] [link]	2019	1	a dataset dataset crawled from 56 popular Java repositories	BLEU
	CommitBERT [122] [link]	2021	6	a multilingual dataset of code modification with commit messages in Github	BLEU
	MCMD [137] [link]	2021	5	a large-scale, information-rich, and multi-language commit message dataset	B-Moses/B-Norm/B-CC
	CoRec [667] [link]	2021	1	a large-scale dataset crawled from 10K popular Java repositories in Github	BLEU
	ExGroFi [668] [link]	2023	1	a dataset anchored on combining correlated commits and issues	BLEU/ROUGE/CIDEr
	CommitChronicle [669] [link]	2023	20	a dataset containing 10.7M commits across 20 programming languages	B-Norm/Edit Sim/EM
	SWE-bench [123] [link]	2023	1	a dataset of 2.2K software engineering problems with pull requests	Resolve Rate/Recall
	CommitBench [670] [link]	2024	6	a reproducible and privacy- and license-aware benchmark for commit message generation	BLEU/METEOR/ROUGE
	DevBench [671] [link]	2024	4	a benchmark to evaluate LLMs across various stages of the software development lifecycle	pass@k

TABLE 8: A more comprehensive collection of code-related benchmarks extended from Table 1. "# PLs" denotes the number of programming languages each benchmark covers (Cont'd).

Task	Dataset	Date	# PLs	Description	Eval. Metric
Code Summarization	CODE-NN [119] [link]	2016	6	(title, query) pairs from StackOverflow	BLEU
	DeepCom [672] [link]	2018	1	a large-scale Java corpus built from 9.7K open source projects from GitHub	BLEU
	TL-CodeSum [120] [link]	2018	6	a dataset of 69K pairs of code and summary	BLEU
	CodeSearchNet [113] [link]	2019	6	a multilingual dataset of 6M functions and query-like natural language	MRR
	HumanEvalPack [307] [link]	2023	6	a benchmark expanding HumanEval to 3 coding tasks across 6 languages	BLEU
Code Debug	QuixBugs [112] [link]	2017	2	a multi-language benchmark (40 bugs in both Python and Java)	Pass Rate
	EvalGPTFix [673] [link]	2023	1	a benchmark containing 151 pairs of bugs and fixes in Java	Pass Rate
	DebugBench [674] [link]	2024	3	a debugging benchmark containing 4K instances of four major bug categories	Pass Rate
Question Answering	CodeQA [170]	2021	2	a free-form QA dataset for Java/Python code comprehension	BLEU/EM/F1
	CodeQueries [675]	2024	1	a dataset to evaluate models to answer semantic queries over python code	EM/Acc./Recall
	InfiCoder-Eval [676]	2024	1	a large-scale freeform QA benchmark for code collected from Stack Overflow	Customized
Code Editing	EditEval [677]	2023	1	a human-written execution-based benchmark of 194 code editing tasks	Acc.
Text2SQL	ATIS [487] [link]	1994	1	a dataset containing 3K utterances reserved for testing	Acc.
	WikiTQ [489] [link]	2015	1	a dataset of 22K complex questions on Wikipedia tables	Acc.
	WikiSQL [488] [link]	2017	1	a dataset of 80K hand-annotated questions and SQL queries from Wikipedia	Acc./F1
	Spider [164] [link]	2018	1	a dataset consisting of 10K questions and 5.6K unique complex SQL queries	Acc./F1
	SParC [165] [link]	2019	1	a dataset consisting of 4.2K coherent question sequences	Acc.
	Cosql [166] [link]	2019	1	a dataset consisting of 30K+ turns and 10K+ annotated SQL queries	Acc.
	Squall [678] [link]	2020	1	a dataset of 11K English questions with manually created SQL equivalents	Acc.
	KaggleDBQA [679] [link]	2021	1	a cross-domain evaluation dataset of real Web databases	Acc.

TABLE 9: More NL2Code benchmarks extended from Table 5. FGA indicates the F1-score of group accuracy.

Purpose	Dataset	Date	# PLs.	Description	Eval. Metric
Open Domain	CONCODE [159] [link]	2018	1	a dataset with 100K examples consisting of Java classes from online code repositories	EM/BLEU
	ODEX [351] [link]	2023	1	an open-domain execution-based natural language to Python code generation dataset	pass@k
Code Exercise	HumanEval [36] [link]	2021	1	a dataset of 164 handwritten programming problems with unit tests	pass@k
	MBPP [352] [link]	2021	1	a dataset containing 974 short Python programs	pass@k
	MathQA-Python [352] [link]	2021	1	a Python version of the MathQA benchmark contains 23K problems	Acc.
	AixBench [680] [link]	2022	1	a code generation benchmark dataset	Pass Rate
	BIG-Bench [353, 326] [link]	2023	-	a benchmark containing over 12 tasks can be solved by coding	-
	CoderEval [681] [link]	2023	2	a dataset consisting of 230 Python and 230 Java code generation tasks	Pass Rate
	CodeApex [582] [link]	2023	1	a bilingual programming evaluation benchmark for large language models	Acc.
	ClassEval [682] [link]	2023	1	A handcrafted benchmark for evaluating class-level code generation	pass@k
	OOP [683] [link]	2024	1	an OOP-focused benchmark, with 431 Python programs that encompass OOP concepts like classes.	pass@k
Competitions	APPS [354] [link]	2021	1	a benchmark including 10K problems for Python code generation	Acc.
	CodeContests [273] [link]	2022	3	a dataset of competitive programming problems	pass@k
	TACO [684] [link]	2023	1	a dataset on algorithmic code generation, consisting of 25K training problems and 1K testing problem	pass@k
Multilingual	MultiPL-E [161] [link]	2022	18	a parallel, multilanguage benchmark for natural-language-to-code generation	pass@k
	MCoNaLa [685] [link]	2022	1	a benchmark for code generation from multiple natural languages	BLEU
	MBXP [355] [link]	2023	12	a benchmark to evaluate code generation models in over 10 programming languages	pass@k
	MathQA-X [355] [link]	2023	3	a dataset transpiled from the original Python dataset MathQA	Pass Rate/Acc.
	xCodeEval [652] [link]	2023	11	an executable dataset of 450K small buggy/fixed program pairs	Pass Rate
	HumanEval-X [290] [link]	2023	4	a benchmark of 164 code problems for evaluating multilingual models	pass@k
Data Science	JuICe [162] [link]	2018	1	a corpus of 1.5M examples with a curated test set of 3.7K instances	EM/BLEU
	PlotCoder [686] [link]	2021	1	a dataset of plot samples extracted from the original dev and test splits of JuICe	Acc.
	DSP [163] [link]	2022	1	a collection of 1.1K problems curated from 306 pedagogical notebooks	pass@k
	ExeDS [687] [link]	2022	1	a dataset of 534 problems for execution evaluation for data science code generation tasks	(Code)BLEU/EM
	DS-1000 [327] [link]	2023	1	a code generation benchmark with 1K data science problems spanning 7 Python libraries	pass@k
	DAEval [688] [link]	2024	1	a dataset comprising 257 data analysis questions derived from 52 CSV files	Pass Rate
Python Libs	PandasEval [199] [link]	2022	1	a dataset consisting of 101 programming problems on Pandas library	pass@k
	NumpyEval [199] [link]	2022	1	a dataset consisting of 101 programming problems on Numpy library	pass@k
	TorchDataEval [356] [link]	2022	1	a dataset with 50 programming problems using the TorchData library	pass@k
Multi-Turn	MTPB [275] [link]	2023	1	an open benchmark consisting of 115 diverse problem sets that are factorized into multi-turn prompts	Pass Rate
Command Line	NL2Bash [160] [link]	2018	1	a corpus of 9K English-command pairs, covering over 100 unique Bash utilities	Acc.
AI4Science	BioCoder [357] [link]	2023	2	a benchmark developed to evaluate large language models in generating bioinformatics-specific code	pass@k
Education	StudentEval [689] [link]	2023	1	a benchmark containing 1.7K prompts for 48 problems, written by 80 students	pass@1
	COJ2022 [87] [link]	2023	1	a dataset containing 5.9K C programs with semantic errors submitted to 498 different assignments	F1
Log Parsing	LogHub^1 [690] [link]	2019	-	a dataset of logs produced by 16 different systems	Acc.
	LogHub^2[691] [link]	2023	-	a suite of 14 datasets, each averaging 3.6M log lines	FGA
Hardware Design	VerilogEval [692] [link]	2023	1	a dataset consists of 156 problems from the Verilog instructional website HDLBits	BLEU/pass@k

TABLE 10: Detailed Pre-training Objectives of CodePTMs listed in Table 2.

Model	Abbr.	Description
CuBERT [178]	MLM NSP	**M**asked **L**anguage **M**odeling Given two sentences, **P**redict whether one sentence is the **N**ext **S**entence of the other [24].
CodeBERT [28]	MLM RTD	**M**asked **L**anguage **M**odeling **D**etect the **T**okens that are randomly **R**eplaced in a sentence [179].
GraphCodeBERT [181]	MLM Edge Pred. Node Align.	**M**asked **L**anguage **M**odeling (Edge Prediction) Mask direct edges that connect sampled nodes in data flow, and predict these masked edges. (Node Alignment) Mask edges between a variable in data flow and code tokens, and predict which token the variable in data flow is identified from.
SynCoBERT [183]	MMLM IP TEP MCL	**M**ulti-Modal **M**asked **L**anguage **M**odeling which jointly models NL, PL, and AST (CFG). **P**redict the type of all code tokens as either **I**dentifier or non-identifier. Mask edges in the A**ST** and ask the model to **P**redict these **E**dges. **M**ulti-Modal **C**ontrastive **L**earning
CODE-MVP [184]	MCL FGTI MMLM	**M**ulti-Modal **C**ontrastive **L**earning Traverse the AST and use the type checker to **I**nfer **F**ine-**G**rained identifier **T**ypes. **M**ulti-Modal **M**asked **L**anguage **M**odeling
SCodeR [186]	-	Soft-Labeled Contrastive Pre-training that uses relevance scores between different samples as softlabels to learn function-level code representation.
DISCO [185]	MLM NT-MLM CLR	**M**asked **L**anguage **M**odeling Local AST **N**ode **T**ype-MLM which learns to recover the masked AST type. **C**ontrastive **L**earning
PLBART [30]	- - -	Random tokens are sampled and replaced with a mask token from the input sequence. Random tokens are deleted from the input sequence. A number of text spans are sampled and replaced with a single mask token.
CodeT5 [29]	MSP IP MIP -	Randomly mask spans with arbitrary lengths and **P**redict these **M**asked **S**pans combined with some sentinel tokens at the decoder. **P**redict the type of all code tokens as either **I**dentifier or non-identifier. **M**ask all **I**dentifiers in the PL segment and **P**redict them. Bimodal Dual Generation
PyMT5 [193]	MSP	Randomly mask spans with arbitrary lengths and **P**redict these **M**asked **S**pans combined with some sentinel tokens at the decoder.
UniXcoder [194]	MLM ULM MSP MCL CMG	**M**asked **L**anguage **M**odeling **U**nidirectional **L**anguage **M**odeling Randomly mask spans with arbitrary lengths and **P**redict these **M**asked **S**pans combined with some sentinel tokens at the decoder. **M**ulti-modal **C**ontrastive **L**earning (**C**ross-**M**odal **G**eneration) Generate code comment which describes the function of the code.
NatGen [195]	-	Naturalizing pre-training which forces the model to generate correct semantically equivalent natural code that is just what a human originally wrote.
TreeBERT [191]	TMLM NOP	**T**ree **M**asked **L**anguage **M**odeling for masking the nodes in the AST and the tokens in the code snippet. Randomly exchange the positions of some nodes in the AST path, and **P**redict whether the **O**rder of **N**odes in the AST is correct.
ERNIE-Code [196]	SCLM PTLM	(**S**pan-**C**orruption **L**anguage **M**odeling) Corrupts 15% of the original NL/PL input tokens and predict the corrupted span on the target side. (**P**ivot-based **T**ranslation **L**anguage **M**odeling) Concatenate parallel source-target sentences and learn to predict the corrupted target language.
CodeExecutor [197]	- -	Code Execution which improves the model ability to understand and execute code. Curriculum Learning which starts from easy instances and then gradually handles harder ones.
LongCoder [118]	CLM	**C**ausal **L**anguage **M**odeling
GPT-C [198] CodeGPT [114] PyCodeGPT [199]	CLM	**C**ausal **L**anguage **M**odeling

References

[1] B. C. Pierce, *Types and programming languages*. MIT press, 2002. 1

[2] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 207–216. 1, 5, 28

[3] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 81, 2018. 1, 3

[4] D. Zan, B. Chen, F. Zhang, D. Lu, B. Wu, B. Guan, W. Yongji, and J.-G. Lou, "Large language models meet NL2Code: A survey," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 7443–7464. [Online]. Available: https://aclanthology.org/2023.acl-long.411 1, 6, 11, 26

[5] Z. Zhang, C. Chen, B. Liu, C. Liao, Z. Gong, H. Yu, J. Li, and R. Wang, "Unifying the perspectives of nlp and software engineering: A survey on language models for code," 2023. 1, 20, 26

[6] Y. Hu, Q. Xie, V. Jain, J. Francis, J. Patrikar, N. Keetha, S. Kim, Y. Xie, T. Zhang, S. Zhao, Y. Q. Chong, C. Wang, K. Sycara, M. Johnson-Roberson, D. Batra, X. Wang, S. Scherer, Z. Kira, F. Xia, and Y. Bisk, "Toward general-purpose robots via foundation models: A survey and meta-analysis," 2023. 1

[7] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," 2023. 2

[8] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. T. Devanbu, "On the naturalness of software," *Commun. ACM*, vol. 59, no. 5, pp. 122–131, 2016. [Online]. Available: https://doi.org/10.1145/2902362 2, 3

[9] T. Sun, X. Liu, X. Qiu, and X. Huang, "Paradigm shift in natural language processing," *Machine Intelligence Research*, vol. 19, pp. 169–183, 2022. 2

[10] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 532–542. 2

[11] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton, "Learning natural coding conventions," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 281–293. [Online]. Available: https://doi.org/10.1145/2635868.2635883 2

[12] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, "Bimodal modelling of source code and natural language," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2123–2132. [Online]. Available: https://proceedings.mlr.press/v37/allamanis15.html 2

[13] P. Bielik, V. Raychev, and M. Vechev, "Phog: Probabilistic model for code," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2933–2942. [Online]. Available: https://proceedings.mlr.press/v48/bielik16.html 2

[14] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735 2, 3

[15] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. [Online]. Available: https://aclanthology.org/D14-1181 2

[16] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 1287–1293. 2, 3, 5, 27

[17] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 783–794. 2, 4

[18] C. Liang, J. Berant, Q. Le, K. Forbus, and N. Lao, "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 23–33. 2

[19] P. Yin and G. Neubig, "TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 7–12. [Online]. Available: https://aclanthology.org/D18-2002 2

[20] P. Yin, "Learning structured neural semantic parsers," Ph.D. dissertation, Carnegie Mellon University, 2021. 2

[21] S. Han, D. Wang, W. Li, and X. Lu, "A comparison of code embeddings and beyond," *ArXiv*, vol. abs/2109.07173, 2021. 2, 26

[22] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "Code2vec: Learning distributed representations of code," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, jan 2019. 2, 3

[23] U. Alon, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," in *International Conference on Learning Representations*, 2019. 2, 3, 5

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of*

the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423 2, 7, 31

[25] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," *OpenAI blog*, 2018. 2, 8

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. 2, 23

[27] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, sep 2020. 2, 7

[28] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. 2, 7, 9, 31

[29] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 8696–8708. 2, 8, 9, 12, 31

[30] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, Jun. 2021, pp. 2655–2668. 2, 8, 9, 31

[31] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020. 2

[32] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. W. Rae, and L. Sifre, "An empirical analysis of compute-optimal large language model training," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=iBBcRUlOAPR 2

[33] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. 2, 11

[34] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," 2022. 2, 10, 13, 22

[35] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," 2023. 2

[36] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. Ponde, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021. 2, 11, 13, 15, 16, 22, 24, 30

[37] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Comput. Surv.*, vol. 55, no. 9, jan 2023. [Online]. Available: https://doi.org/10.1145/3560815 2

[38] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, L. Li, and Z. Sui, "A survey on in-context learning," 2023. 2, 18

[39] H. K. Dam, T. Tran, and T. Pham, "A deep language model for software code," 2016. 3

[40] H. Liang, Y. Yu, L. Jiang, and Z. Xie, "Seml: A semantic lstm model for software defect prediction," *IEEE Access*, vol. 7, pp. 83 812–83 824, 2019. 3

[41] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "A general path-based representation for predicting program properties," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 404–419. [Online]. Available: https://doi.org/10.1145/3192366.3192412 3

[42] C. Maddison and D. Tarlow, "Structured generative models of natural source code," in *International Conference on Machine Learning*. PMLR, 2014, pp. 649–657. 3

[43] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, "Bimodal modelling of source code and natural language," in *International conference on machine learning*. PMLR, 2015, pp. 2123–2132. 3

[44] W. Wang, G. Li, S. Shen, X. Xia, and Z. Jin, "Modular tree network for source code representation learning," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–23, 2020. 3

[45] C. Cummins, Z. V. Fisches, T. Ben-Nun, T. Hoefler, M. F. P. O'Boyle, and H. Leather, "Programl: A graph-based program representation for data flow analysis and compiler optimizations," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 2244–2253. [Online]. Available: https://proceedings.mlr.press/v139/cummins21a.html 3

[46] F. E. Allen, "Control flow analysis," *ACM Sigplan Notices*, vol. 5, no. 7, pp. 1–19, 1970. 3

[47] L. Li, J. Wang, and H. Quan, "Scalpel: The python static analysis framework," 2022. 3

[48] J. Li, P. He, J. Zhu, and M. R. Lyu, "Software defect prediction via convolutional neural network," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2017, pp. 318–328. 3

[49] N. D. Bui, L. Jiang, and Y. Yu, "Cross-language learning for program classification using bilateral tree-based convolutional neural networks," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 3

[50] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, 2016, pp. 87–98. 3

[51] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *International Conference on Learning Representations*, 2018. 4

[52] Z. Xu, M. Zhou, X. Zhao, Y. Chen, X. Cheng, and H. Zhang, "xastnn: Improved code representations for industrial practice," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1727–1738. [Online]. Available: https://doi.org/10.1145/3611643.3613869 4

[53] N. D. Bui, Y. Yu, and L. Jiang, "Infercode: Self-supervised learning of code representations by predicting subtrees," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1186–1197. 4

[54] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 261–271. 4

[55] Y. Wang and H. Li, "Code completion by modeling flattened abstract syntax trees as graphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 16, pp. 14 015–14 023, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17650 4

[56] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl 4

[57] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ 4

[58] M. Brockschmidt, M. Allamanis, A. L. Gaunt, and O. Polozov, "Generative code modeling with graphs," in *International Conference on Learning Representations*, 2018. 4

[59] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big code!= big vocabulary: open-vocabulary models for source code," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1073–1085. [Online]. Available: https://doi.org/10.1145/3377811.3380342 4

[60] M. Cvitkovic, B. Singh, and A. Anandkumar, "Open vocabulary learning on source code with a graph-structured cache," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1475–1485. [Online]. Available: https://proceedings.mlr.press/v97/cvitkovic19b.html 4

[61] R. Zhu, L. Yuan, X. Li, M. Gao, and W. Cai, "A neural network architecture for program understanding inspired by human behaviors," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 5142–5153. 4, 23

[62] N. Chen, Q. Sun, R. Zhu, X. Li, X. Lu, and M. Gao, "CAT-probing: A metric-based approach to interpret how pre-trained models for programming language attend code structure," in *Findings of the Association for Computational Linguistics: EMNLP 2022*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 4000–4008. 4, 10, 23

[63] H. Wei and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code." in *IJCAI*, 2017, pp. 3034–3040. 4

[64] J. Svajlenko and C. K. Roy, "A survey on the evaluation of clone detection performance and benchmarking," 2020. 4

[65] H. Zhang and K. Sakurai, "A survey of software clone detection from security perspective," *IEEE Access*, vol. 9, pp. 48 157–48 173, 2021. 4

[66] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward! 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 143–153. [Online]. Available: https://doi.org/10.1145/3359591.3359735 4

[67] A. Eghbali and M. Pradel, "Crystalbleu: Precisely and efficiently measuring the similarity of code," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3556903 4, 15

[68] C. K. Roy and J. R. Cordy, "Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *2008 16th IEEE International Conference on Program Comprehension*, 2008, pp. 172–181. 4

[69] M. Chilowicz, E. Duris, and G. Roussel, "Syntax tree fingerprinting for source code similarity detection," in *2009 IEEE 17th International Conference on Program Comprehension*, 2009, pp. 243–247. 4

[70] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: Scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 1157–1168. 4

[71] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," 2007. 4

[72] Y. Wu, D. Zou, S. Dou, S. Yang, W. Yang, F. Cheng, H. Liang, and H. Jin, "Scdetector: Software functional clone detection based on semantic tokens analysis," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 821–833. 4

[73] K. Wang and Z. Su, "Blended, precise semantic program embeddings," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 121–134. 4

[74] J. Krinke and C. Ragkhitwetsagul, "Bigclonebench considered harmful for machine learning," in *2022 IEEE 16th International Workshop on Software Clones (IWSC)*, 2022, pp. 1–7. 4

[75] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, "On the "naturalness" of buggy code," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 428–439. [Online]. Available: https://doi.org/10.1145/2884781.2884848 4

[76] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 297–308. [Online]. Available: https://doi.org/10.1145/2884781.2884804 4

[77] Y. Li, S. Wang, T. N. Nguyen, and S. Van Nguyen, "Improving bug detection via context-based code representation learning and attention-based neural networks," *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, oct 2019. [Online]. Available: https://doi.org/10.1145/3360588 4

[78] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019. 4, 5, 27

[79] L. zhong, P. Minxue, P. Yu, Z. Tian, W. Linzhang, and L. Xuandong, "Empirically revisiting and enhancing automatic classification of bug and non-bug issues," *Frontiers of Computer Science*, vol. 18, no. 5, p. 185207, 2024. 4

[80] Z. Zhou, C. Sha, and X. Peng, "On calibration of pre-trained code models," in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, apr 2024, pp. 861–861. [Online]. Available: https://doi.ieeecomputersociety.org/ 4

[81] C. Spiess, D. Gros, K. S. Pai, M. Pradel, M. R. I. Rabin, A. Alipour, S. Jha, P. Devanbu, and T. Ahmed, "Calibration and correctness of language models for code," 2024. 4

[82] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International conference on software engineering (ICSE)*. IEEE, 2012, pp. 14–24. 4

[83] R. Gupta, S. Pal, A. Kanade, and S. Shevade, "Deepfix: Fixing common c language errors by deep learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10742 4, 5, 27

[84] S. Bhatia, P. Kohli, and R. Singh, "Neuro-symbolic program corrector for introductory programming assignments," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 60–70. 4

[85] V. J. Hellendoorn, C. Sutton, R. Singh, P. Maniatis, and D. Bieber, "Global relational models of source code," in *International conference on learning representations*, 2019. 4, 27

[86] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, "An empirical study on learning bug-fixing patches in the wild via neural machine translation," *ACM Transactions on Software Engineering and Methodology*, vol. 28, no. 4, sep 2019. [Online]. Available: https://doi.org/10.1145/3340544 4

[87] S. Han, Y. Wang, and X. Lu, "Errorclr: Semantic error classification, localization and repair for introductory programming assignments," in *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1345–1354. [Online]. Available: https://doi.org/10.1145/3539618.3591680 4, 27, 30

[88] M. Allamanis, H. Jackson-Flux, and M. Brockschmidt, "Self-supervised bug detection and repair," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 27 865–27 876. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/ea96efc03b9a050d895110db8c4af057-Abstract.html 4, 27

[89] R. Robbes and M. Lanza, "How program history can improve code completion," in *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008, pp. 317–326. 4

[90] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 213–222. [Online]. Available: https://doi.org/10.1145/1595696.1595728

4
[91] S. Han, D. R. Wallace, and R. C. Miller, “Code completion from abbreviated input,” in *2009 IEEE/ACM International Conference on Automated Software Engineering*, 2009, pp. 332–343. 4
[92] D. Hou and D. M. Pletcher, “An evaluation of the strategies of sorting, filtering, and grouping api methods for code completion,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 233–242. 4
[93] A. Svyatkovskiy, S. Lee, A. Hadjitofi, M. Riechert, J. V. Franco, and M. Allamanis, “Fast and memory-efficient neural code completion,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 329–340. 4
[94] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, S. Yih, L. Zettlemoyer, and M. Lewis, “Incoder: A generative model for code infilling and synthesis,” in *The Eleventh International Conference on Learning Representations*, 2023. 4, 11, 12, 13
[95] M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen, “Efficient training of language models to fill in the middle,” 2022. 4, 12
[96] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, “Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices,” in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1214–1224. [Online]. Available: https://doi.org/10.1145/3468264.3473915 4
[97] V. Nitin, S. Asthana, B. Ray, and R. Krishna, “Cargo: Ai-guided dependency analysis for migrating monolithic applications to microservices architecture,” in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3556960 4
[98] R. Pan, A. R. Ibrahimzada, R. Krishna, D. Sankar, L. P. Wassi, M. Merler, B. Sobolev, R. Pavuluri, S. Sinha, and R. Jabbarvand, “Lost in translation: A study of bugs introduced by large language models while translating code,” in *International Conference on Software Engineering*, 2024. 4
[99] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen, “Divide-and-conquer approach for multi-phase statistical migration for source code (t),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 585–596. 4
[100] X. Chen, C. Liu, and D. Song, “Tree-to-tree neural networks for program translation,” *Advances in neural information processing systems*, vol. 31, 2018. 4
[101] B. Rozière, M. Lachaux, L. Chanussot, and G. Lample, “Unsupervised translation of programming languages,” in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/ed23fbf18c2cd35f8c7f8de44f85c08d-Abstract.html 4, 28
[102] Y. Wen, Q. Guo, Q. Fu, X. Li, J. Xu, Y. Tang, Y. Zhao, X. Hu, Z. Du, L. Li, C. Wang, X. Zhou, and Y. Chen, “BabelTower: Learning to auto-parallelized program translation,” in *Proceedings of the 39th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 17–23 Jul 2022, pp. 23 685–23 700. [Online]. Available: https://proceedings.mlr.press/v162/wen22b.html 4
[103] F. Liu, J. Li, and L. Zhang, “Syntax and domain aware model for unsupervised program translation,” in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE ’23. IEEE Press, 2023, p. 755–767. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00072 4
[104] Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura, “Learning to generate pseudo-code from source code using statistical machine translation,” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 574–584. 4
[105] S. Kulal, P. Pasupat, K. Chandra, M. Lee, O. Padon, A. Aiken, and P. S. Liang, “Spoc: Search-based pseudocode to code,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/7298332f04ac004a0ca44cc69ecf6f6b-Paper.pdf 4, 15
[106] C.-Y. Lin, “ROUGE: A package for automatic evaluation of summaries,” in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013 4, 15
[107] M. Denkowski and A. Lavie, “Meteor universal: Language specific translation evaluation for any target language,” in *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, Jun. 2014, pp. 376–380. [Online]. Available: https://aclanthology.org/W14-3348 4, 15
[108] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, “Towards a big data curated benchmark of inter-project code clones,” in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 476–480. 5, 27
[109] K. W. Nafi, T. S. Kar, B. Roy, C. K. Roy, and K. A. Schneider, “Clcdsa: Cross language code clone detection using syntactical features and api documentation,” in *2019 34th IEEE/ACM International Conference*

on Automated Software Engineering (ASE), 2019, pp. 1026–1037. 5, 27

[110] G. Nikitopoulos, K. Dritsa, P. Louridas, and D. Mitropoulos, "Crossvul: A cross-language vulnerability dataset with commit data," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1565–1569. [Online]. Available: https://doi.org/10.1145/3468264.3473122 5, 27

[111] Y. Chen, Z. Ding, L. Alowain, X. Chen, and D. Wagner, "Diversevul: A new vulnerable source code dataset for deep learning based vulnerability detection," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 654–668. [Online]. Available: https://doi.org/10.1145/3607199.3607242 5, 27

[112] H. Ye, M. Martinez, T. Durieux, and M. Monperrus, "A comprehensive study of automatic program repair on the quixbugs benchmark," *Journal of Systems and Software*, vol. 171, p. 110825, Jan. 2021. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2020.110825 5, 29

[113] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019. 5, 7, 11, 28, 29

[114] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. GONG, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. LIU, "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: https://openreview.net/forum?id=6lE4dQXaUcb 5, 6, 8, 9, 15, 24, 28, 31

[115] M. Zhu, K. Suresh, and C. K. Reddy, "Multilingual code snippets training for program translation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, pp. 11 783–11 790, Jun. 2022. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/21434 5, 28

[116] W. Yan, Y. Tian, Y. Li, Q. Chen, and W. Wang, "CodeTransOcean: A comprehensive multilingual benchmark for code translation," in *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5067–5089. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.337 5, 28

[117] V. Raychev, P. Bielik, and M. Vechev, "Probabilistic model for code with decision trees," *SIGPLAN Not.*, vol. 51, no. 10, p. 731–747, oct 2016. [Online]. Available: https://doi.org/10.1145/3022671.2984041 5, 28

[118] D. Guo, C. Xu, N. Duan, J. Yin, and J. Mcauley, "LongCoder: A long-range pre-trained language model for code completion," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 23–29 Jul 2023, pp. 12 098–12 107. [Online]. Available: https://proceedings.mlr.press/v202/guo23j.html 5, 9, 13, 28, 31

[119] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "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)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 2073–2083. [Online]. Available: https://aclanthology.org/P16-1195 5, 29

[120] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing source code with transferred api knowledge," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 2269–2275. [Online]. Available: https://doi.org/10.24963/ijcai.2018/314 5, 29

[121] P. Loyola, E. Marrese-Taylor, and Y. Matsuo, "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)*, R. Barzilay and M.-Y. Kan, Eds. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 287–292. [Online]. Available: https://aclanthology.org/P17-2045 5, 28

[122] T. H. Jung, "CommitBERT: Commit message generation using pre-trained programming language model," in *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 26–33. [Online]. Available: https://aclanthology.org/2021.nlp4prog-1.3 5, 28

[123] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "Swe-bench: Can language models resolve real-world github issues?" in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=VTF8yNQM66 5, 28

[124] Y. Zhu and M. Pan, "Automatic code summarization: A systematic literature review," 2019. 5

[125] Z. H. Yamin HU, Hao JIANG, "Measuring code maintainability with deep neural networks," *Frontiers of Computer Science*, vol. 17, no. 6, p. 176214, 2023. 5

[126] P. Fernandes, M. Allamanis, and M. Brockschmidt, "Structured neural summarization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1ersoRqtm 5

[127] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, "Improving automatic source code summarization via deep reinforcement learning," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '18. New York, NY, USA: Association for Computing

Machinery, 2018, p. 397–407. [Online]. Available: https://doi.org/10.1145/3238147.3238206 5

[128] A. LeClair, S. Jiang, and C. McMillan, "A neural model for generating natural language summaries of program subroutines," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 795–806. 5

[129] M. Allamanis, H. Peng, and C. Sutton, "A convolutional attention network for extreme summarization of source code," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 2091–2100. [Online]. Available: https://proceedings.mlr.press/v48/allamanis16.html 5

[130] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 4998–5007. 5

[131] H. Peng, G. Li, W. Wang, Y. Zhao, and Z. Jin, "Integrating tree path in transformer for code representation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 9343–9354, 2021. 5

[132] W. Pian, H. Peng, X. Tang, T. Sun, H. Tian, A. Habib, J. Klein, and T. F. Bissyandé, "Metatptrans: A meta learning approach for multilingual code representation learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 5239–5247. 5

[133] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proceedings of the 2010 17th Working Conference on Reverse Engineering*, ser. WCRE '10. USA: IEEE Computer Society, 2010, p. 35–44. [Online]. Available: https://doi.org/10.1109/WCRE.2010.13 5

[134] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using ir methods for labeling source code artifacts: Is it worthwhile?" in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, 2012, pp. 193–202. 5

[135] E. Wong, T. Liu, and L. Tan, "Clocom: Mining existing source code for automatic comment generation," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2015, pp. 380–389. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SANER.2015.7081848 5

[136] P. Yin, G. Neubig, M. Allamanis, M. Brockschmidt, and A. L. Gaunt, "Learning to represent edits," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=BJl6AjC5F7 5

[137] W. Tao, Y. Wang, E. Shi, L. Du, S. Han, H. Zhang, D. Zhang, and W. Zhang, "On the evaluation of commit message generation models: An experimental study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 126–136. 5, 28

[138] ——, "A large-scale empirical study of commit message generation: Models, datasets and evaluation," *Empirical Softw. Engg.*, vol. 27, no. 7, dec 2022. 5

[139] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*, 2014, pp. 275–284. 5

[140] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, M. Huchard, C. Kästner, and G. Fraser, Eds. ACM, 2018, pp. 373–384. [Online]. Available: https://doi.org/10.1145/3238147.3238190 5, 28

[141] S. Jiang and C. McMillan, "Towards automatic generation of short summaries of commits," in *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE Computer Society, 2017, pp. 320–323. [Online]. Available: https://doi.org/10.1109/ICPC.2017.12 5, 28

[142] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, S. Kraus, Ed. ijcai.org, 2019, pp. 3975–3981. [Online]. Available: https://doi.org/10.24963/ijcai.2019/552 5, 28

[143] E. Shi, Y. Wang, W. Tao, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, "RACE: Retrieval-augmented commit message generation," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 5520–5530. 5

[144] M. R. Parvez, W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Retrieval augmented code generation and summarization," in *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2719–2734. [Online]. Available: https://aclanthology.org/2021.findings-emnlp.232 5, 9

[145] Y. Xie, J. Lin, H. Dong, L. Zhang, and Z. Wu, "Survey of code search based on deep learning," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 2, p. 1–42, Dec. 2023. [Online]. Available: http://dx.doi.org/10.1145/3628161 5

[146] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 933–944. [Online]. Available: https://doi.org/10.1145/3180155.3180167

5, 28

[147] W. Wang, Y. Zhang, Z. Zeng, and G. Xu, "Trans$\hat{3}$: A transformer-based framework for unifying code summarization and code search," 2020. 5

[148] L. Di Grazia and M. Pradel, "Code search: A survey of techniques for finding code," *ACM Comput. Surv.*, vol. 55, no. 11, feb 2023. [Online]. Available: https://doi.org/10.1145/3565971 5

[149] K. Kim, S. Ghatpande, D. Kim, X. Zhou, K. Liu, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Big code search: A bibliography," *ACM Comput. Surv.*, vol. 56, no. 1, aug 2023. [Online]. Available: https://doi.org/10.1145/3604905 5

[150] S. A. Hayati, R. Olivier, P. Avvaru, P. Yin, A. Tomasic, and G. Neubig, "Retrieval-based neural code generation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 925–930. 5

[151] R. Balzer, "A 15 year perspective on automatic programming," *IEEE Trans. Softw. Eng.*, vol. 11, no. 11, p. 1257–1268, nov 1985. [Online]. Available: https://doi.org/10.1109/TSE.1985.231877 5

[152] T. Ben-Nun, A. S. Jakobovits, and T. Hoefler, "Neural code comprehension: A learnable representation of code semantics," *Advances in Neural Information Processing Systems*, vol. 31, 2018. 5

[153] Z. Sun, Q. Zhu, L. Mou, Y. Xiong, G. Li, and L. Zhang, "A grammar-based structural cnn decoder for code generation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 7055–7062. 5

[154] E. C. Shin, M. Allamanis, M. Brockschmidt, and A. Polozov, "Program synthesis and semantic parsing with learned code idioms," *Advances in Neural Information Processing Systems*, vol. 32, 2019. 5

[155] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 440–450. [Online]. Available: https://aclanthology.org/P17-1041 5

[156] X. V. Lin, C. Wang, D. Pang, K. Vu, L. Zettlemoyer, and M. D. Ernst, "Program synthesis from natural language using recurrent neural networks," *University of Washington Department of Computer Science and Engineering, Seattle, WA, USA, Tech. Rep. UW-CSE-17-03-01*, 2017. 5

[157] Z. Sun, Q. Zhu, L. Mou, Y. Xiong, G. Li, and L. Zhang, "A grammar-based structural cnn decoder for code generation," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33017055 5

[158] W. Ling, P. Blunsom, E. Grefenstette, K. M. Hermann, T. Kočiský, F. Wang, and A. Senior, "Latent predictor networks for code generation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 599–609. 6

[159] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Mapping language to code in programmatic context," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 1643–1652. [Online]. Available: https://aclanthology.org/D18-1192 6, 16, 30

[160] X. V. Lin, C. Wang, L. Zettlemoyer, and M. D. Ernst, "NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://aclanthology.org/L18-1491 6, 16, 30

[161] F. Cassano, J. Gouwar, D. Nguyen, S. Nguyen, L. Phipps-Costin, D. Pinckney, M.-H. Yee, Y. Zi, C. J. Anderson, M. Q. Feldman, A. Guha, M. Greenberg, and A. Jangda, "Multipl-e: A scalable and polyglot approach to benchmarking neural code generation," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, pp. 3675–3691, 2023. 6, 14, 15, 16, 25, 30

[162] R. Agashe, S. Iyer, and L. Zettlemoyer, "JuICe: A large scale distantly supervised dataset for open domain context-based code generation," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 5436–5446. [Online]. Available: https://aclanthology.org/D19-1546 6, 16, 22, 30

[163] S. Chandel, C. B. Clement, G. Serrato, and N. Sundaresan, "Training and evaluating a jupyter notebook data science assistant," 2022. 6, 8, 16, 21, 22, 30

[164] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3911–3921. [Online]. Available: https://aclanthology.org/D18-1425 6, 14, 21, 29

[165] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, J. Kraft, V. Zhang, C. Xiong, R. Socher, and D. R. Radev, "Sparc: Cross-domain semantic parsing in context," in *Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL)*, 2019, pp. 4511–4523. [Online]. Available: https://doi.org/10.18653/v1/p19-1443 6, 21, 29

[166] T. Yu, R. Zhang, H. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. R. Fabbri, Z. Li, L. Chen,

Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. S. Lasecki, and D. R. Radev, "Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 1962–1979. [Online]. Available: https://doi.org/10.18653/v1/D19-1204 6, 21, 29

[167] B. Qin, B. Hui, L. Wang, M. Yang, J. Li, B. Li, R. Geng, R. Cao, J. Sun, L. Si, F. Huang, and Y. Li, "A survey on text-to-sql parsing: Concepts, methods, and future directions," 2022. 6

[168] N. Deng, Y. Chen, and Y. Zhang, "Recent advances in text-to-SQL: A survey of what we have and what we expect," in *Proceedings of the 29th International Conference on Computational Linguistics*. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 2166–2187. [Online]. Available: https://aclanthology.org/2022.coling-1.190 6

[169] G. Katsogiannis-Meimarakis and G. Koutrika, "A survey on deep learning approaches for text-to-sql," *The VLDB Journal*, vol. 32, no. 4, p. 905–936, jan 2023. [Online]. Available: https://doi.org/10.1007/s00778-022-00776-8 6

[170] C. Liu and X. Wan, "CodeQA: A question answering dataset for source code comprehension," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2618–2632. [Online]. Available: https://aclanthology.org/2021.findings-emnlp.223 6, 29

[171] X. Yu, Q. Huang, Z. Wang, Y. Feng, and D. Zhao, "Towards context-aware code comment generation," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 3938–3947. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.350 6

[172] R. B. Yadav, P. S. Kumar, and S. V. Dhavale, "A survey on log anomaly detection using deep learning," in *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, 2020, pp. 1215–1220. 6

[173] D. Miao, Y. Dong, and X. Lu, "Pipe: Predicting logical programming errors in programming exercises." *International Educational Data Mining Society*, 2020. 6

[174] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker, V. Thost, L. Buratti, S. Pujar, S. Ramji, U. Finkler, S. Malaika, and F. Reiss, "Codenet: A large-scale AI for code dataset for learning a diversity of coding tasks," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. [Online]. Available: https://openreview.net/forum?id=6vZVBkCDrHT 6

[175] R. Zhu, D. Zhang, C. Han, M. Gaol, X. Lu, W. Qian, and A. Zhou, "Programming knowledge tracing: A comprehensive dataset and a new model," in *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2022, pp. 298–307. 6

[176] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, vol. 3, pp. 111–132, 2022. 7, 23

[177] L. Buratti, S. Pujar, M. Bornea, S. McCarley, Y. Zheng, G. Rossiello, A. Morari, J. Laredo, V. Thost, Y. Zhuang, and G. Domeniconi, "Exploring software naturalness through neural language models," 2020. 7

[178] A. Kanade, P. Maniatis, G. Balakrishnan, and K. Shi, "Learning and evaluating contextual embedding of source code," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 5110–5121. 7, 9, 31

[179] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1xMH1BtvB 7, 31

[180] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019. 7

[181] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. LIU, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "GraphCodeBERT: Pre-training code representations with data flow," in *International Conference on Learning Representations*, 2021. 7, 9, 23, 31

[182] A. Neelakantan, T. Xu, R. Puri, A. Radford, J. M. Han, J. Tworek, Q. Yuan, N. Tezak, J. W. Kim, C. Hallacy, J. Heidecke, P. Shyam, B. Power, T. E. Nekoul, G. Sastry, G. Krueger, D. Schnurr, F. P. Such, K. Hsu, M. Thompson, T. Khan, T. Sherbakov, J. Jang, P. Welinder, and L. Weng, "Text and code embeddings by contrastive pre-training," 2022. 7

[183] X. Wang, Y. Wang, F. Mi, P. Zhou, Y. Wan, X. Liu, L. Li, H. Wu, J. Liu, and X. Jiang, "SynCoBERT: Syntax-guided multi-modal contrastive pre-training for code representation," in *arXiv*, 2021. 7, 9, 31

[184] X. Wang, Y. Wang, Y. Wan, J. Wang, P. Zhou, L. Li, H. Wu, and J. Liu, "CODE-MVP: Learning to represent source code from multiple views with contrastive pre-training," in *Findings of the Association for Computational Linguistics: NAACL 2022*. Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 1066–1077. [Online]. Available: https://aclanthology.org/2022.findings-naacl.80 7, 9, 31

[185] Y. Ding, L. Buratti, S. Pujar, A. Morari, B. Ray, and S. Chakraborty, "Towards learning (dis)-similarity of source code from program contrasts," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 6300–6312. 7, 9, 31

[186] X. Li, D. Guo, Y. Gong, Y. Lin, Y. Shen, X. Qiu, D. Jiang, W. Chen, and N. Duan, "Soft-labeled contrastive pre-training for function-level code representation," in *Findings of the Association for Computational Linguistics: EMNLP 2022*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 118–129. 7, 9, 31

[187] P. Jain, A. Jain, T. Zhang, P. Abbeel, J. Gonzalez, and I. Stoica, "Contrastive code representation learning," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 5954–5971. [Online]. Available: https://aclanthology.org/2021.emnlp-main.482 7

[188] N. D. Q. Bui, Y. Yu, and L. Jiang, "Self-supervised contrastive learning for code retrieval and summarization via semantic-preserving transformations," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 511–521. [Online]. Available: https://doi.org/10.1145/3404835.3462840 7

[189] M.-A. Lachaux, B. Roziere, M. Szafraniec, and G. Lample, "Dobf: A deobfuscation pre-training objective for programming languages," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 14 967–14 979. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/7d6548bdc0082aacc950ed35e91fcccb-Paper.pdf 7

[190] A. Elnaggar, W. Ding, L. Jones, T. Gibbs, T. Feher, C. Angerer, S. Severini, F. Matthes, and B. Rost, "Codetrans: Towards cracking the language of silicon's code through self-supervised deep learning and high performance computing," 2021. 8

[191] X. Jiang, Z. Zheng, C. Lyu, L. Li, and L. Lyu, "Treebert: A tree-based pre-trained model for programming language," in *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, ser. Proceedings of Machine Learning Research, C. de Campos and M. H. Maathuis, Eds., vol. 161. PMLR, 27–30 Jul 2021, pp. 54–63. [Online]. Available: https://proceedings.mlr.press/v161/jiang21a.html 8, 9, 31

[192] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Jul. 2020, pp. 7871–7880. 8

[193] C. B. Clement, D. Drain, J. Timcheck, A. Svyatkovskiy, and N. Sundaresan, "Pymt5: multi-mode translation of natural language and python code with transformers," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, 2020, pp. 9052–9065. [Online]. Available: https://doi.org/10.18653/v1/2020.emnlp-main.728 8, 9, 31

[194] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7212–7225. 8, 9, 12, 31

[195] S. Chakraborty, T. Ahmed, Y. Ding, P. T. Devanbu, and B. Ray, "Natgen: Generative pre-training by "naturalizing" source code," in *ESEC/FSE*. Association for Computing Machinery, 2022, p. 18–30. 8, 9, 31

[196] Y. Chai, S. Wang, C. Pang, Y. Sun, H. Tian, and H. Wu, "ERNIE-code: Beyond English-centric cross-lingual pretraining for programming languages," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 10 628–10 650. [Online]. Available: https://aclanthology.org/2023.findings-acl.676 9, 31

[197] C. Liu, S. Lu, W. Chen, D. Jiang, A. Svyatkovskiy, S. Fu, N. Sundaresan, and N. Duan, "Code execution with pre-trained language models," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4984–4999. [Online]. Available: https://aclanthology.org/2023.findings-acl.308 9, 31

[198] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, 2020, p. 1433–1443. 8, 9, 20, 31

[199] D. Zan, B. Chen, D. Yang, Z. Lin, M. Kim, B. Guan, Y. Wang, W. Chen, and J.-G. Lou, "Cert: Continual pre-training on sketches for library-oriented code generation," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, L. D. Raedt, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 2369–2375, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2022/329 8, 9, 16, 21, 30, 31

[200] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. 8

[201] A. Mastropaolo, S. Scalabrino, N. Cooper, D. Nader-Palacio, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Studying the usage of text-to-text transfer transformer to support code-related tasks," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 336–347. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00041 8

[202] H. Le, Y. Wang, A. D. Gotmare, S. Savarese,

and S. Hoi, “CodeRL: Mastering code generation through pretrained models and deep reinforcement learning,” in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=WaGvb7OzySA 8, 15

[203] C. Niu, C. Li, V. Ng, J. Ge, L. Huang, and B. Luo, “Spt-code: Sequence-to-sequence pre-training for learning source code representations,” in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2022, pp. 1–13. [Online]. Available: https://doi.ieeecomputersociety.org/10.1145/3510003.3510096 8

[204] S. Tipirneni, M. Zhu, and C. K. Reddy, “Structcoder: Structure-aware transformer for code generation,” 2024. 8

[205] Y. Wang, H. Le, A. D. Gotmare, J. Li, and S. Hoi, “Codet5mix: A pretrained mixture of encoder-decoder transformers for code understanding and generation,” 2023. [Online]. Available: https://openreview.net/forum?id=VPCi3STZcaO 8

[206] L. Gong, M. Elhoushi, and A. Cheung, “Ast-t5: Structure-aware pretraining for code generation and understanding,” 2024. 8

[207] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H. Hon, “Unified language model pre-training for natural language understanding and generation,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019, pp. 13 042–13 054. 8

[208] H. Bao, L. Dong, F. Wei, W. Wang, N. Yang, X. Liu, Y. Wang, J. Gao, S. Piao, M. Zhou, and H.-W. Hon, “UniLMv2: Pseudo-masked language models for unified language model pre-training,” in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 13–18 Jul 2020, pp. 642–652. [Online]. Available: https://proceedings.mlr.press/v119/bao20a.html 8

[209] F. Liu, G. Li, Y. Zhao, and Z. Jin, “Multi-task learning based pre-trained language model for code completion,” in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’20. New York, NY, USA: Association for Computing Machinery, 2021, p. 473–485. [Online]. Available: https://doi.org/10.1145/3324884.3416591 8, 12

[210] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, 2019. 8, 21

[211] CodedotAl, “GPT Code Clippy: The Open Source version of GitHub Copilot,” 2021, https://github.com/CodedotAl/gpt-code-clippy. 8

[212] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, “GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow,” Mar. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5297715 8

[213] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, “The pile: An 800gb dataset of diverse text for language modeling,” 2020. 8, 10

[214] C.-Y. Su, A. Bansal, V. Jain, S. Ghanavati, and C. McMillan, “A language model of java methods with train/test deduplication,” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 2152–2156. [Online]. Available: https://doi.org/10.1145/3611643.3613090 8

[215] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, “Transformers: State-of-the-art natural language processing,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6 9

[216] J. Wang, N. Chen, Q. Sun, W. Huang, C. Wang, and M. Gao, “HugNLP: A unified and comprehensive library for natural language processing,” in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM ’23)*, ser. Demo Papers. Association for Computing Machinery, 2023. [Online]. Available: https://arxiv.org/abs/2302.14286 9

[217] N. D. Q. Bui, H. Le, Y. Wang, J. Li, A. D. Gotmare, and S. C. H. Hoi, “Codetf: One-stop transformer library for state-of-the-art code llm,” 2023. 9, 15

[218] Y. Wan, Y. He, Z. Bi, J. Zhang, H. Zhang, Y. Sui, G. Xu, H. Jin, and P. S. Yu, “Deep learning for code intelligence: Survey, benchmark and toolkit,” 2023. 9, 26

[219] X. Li, Y. Gong, Y. Shen, X. Qiu, H. Zhang, B. Yao, W. Qi, D. Jiang, W. Chen, and N. Duan, “CodeRetriever: A large scale contrastive pre-training method for code search,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 2898–2910. 9

[220] S. Lu, N. Duan, H. Han, D. Guo, S.-w. Hwang, and A. Svyatkovskiy, “ReACC: A retrieval-augmented code completion framework,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 6227–6240. [Online]. Available: https://aclanthology.org/2022.acl-long.431 9

[221] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, and N. Sundaresan, “Automating code review activities by large-scale pre-training,” in *ESEC/FSE*. Singapore: Association for Computing Machinery, November 2022, p. 1035–1047. 9

[222] D. Zügner, T. Kirschstein, M. Catasta, J. Leskovec, and

S. Günnemann, "Language-agnostic representation learning of source code from structure and context," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=Xh5eMZVONGF 9

[223] D. Guo, A. Svyatkovskiy, J. Yin, N. Duan, M. Brockschmidt, and M. Allamanis, "Learning to complete code with sketches," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=q79uMSC6ZBT 9

[224] C. Clement, S. Lu, X. Liu, M. Tufano, D. Drain, N. Duan, N. Sundaresan, and A. Svyatkovskiy, "Long-range modeling of source code files with eWASH: Extended window access by syntax hierarchy," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 4713–4722. [Online]. Available: https://aclanthology.org/2021.emnlp-main.387 9

[225] A. Raganato and J. Tiedemann, "An analysis of encoder representations in transformer-based machine translation," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 287–297. [Online]. Available: https://aclanthology.org/W18-5431 10

[226] S. Serrano and N. A. Smith, "Is attention interpretable?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2931–2951. 10

[227] S. Wiegreffe and Y. Pinter, "Attention is not not explanation," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 11–20. 10

[228] J. Hewitt and C. D. Manning, "A structural probe for finding syntax in word representations," 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4129–4138. [Online]. Available: https://aclanthology.org/N19-1419 10

[229] A. Karmakar and R. Robbes, "What do pre-trained code models know about code?" in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1332–1336. 10

[230] ——, "Inspect: Intrinsic and systematic probing evaluation for code transformers," 2023. 10

[231] S. Troshin and N. Chirkova, "Probing pretrained models of source codes," in *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, J. Bastings, Y. Belinkov, Y. Elazar, D. Hupkes, N. Saphra, and S. Wiegreffe, Eds. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, Dec. 2022, pp. 371–383. [Online]. Available: https://aclanthology.org/2022.blackboxnlp-1.31 10

[232] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? an analysis of BERT's attention," in *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 276–286. 10

[233] G. Jawahar, B. Sagot, and D. Seddah, "What does BERT learn about the structure of language?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3651–3657. [Online]. Available: https://aclanthology.org/P19-1356 10

[234] D. Yenicelik, F. Schmidt, and Y. Kilcher, "How does BERT capture semantics? a closer look at polysemous words," in *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Online: Association for Computational Linguistics, Nov. 2020, pp. 156–162. [Online]. Available: https://aclanthology.org/2020.blackboxnlp-1.15 10

[235] J. Vig, A. Madani, L. R. Varshney, C. Xiong, richard socher, and N. Rajani, "{BERT}ology meets biology: Interpreting attention in protein language models," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YWtLZvLmud7 10

[236] Y. Wan, W. Zhao, H. Zhang, Y. Sui, G. Xu, and H. Jin, "What do they capture? a structural analysis of pre-trained language models for source code," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2377–2388. 10

[237] J. A. Hernández López, M. Weyssow, J. S. Cuadrado, and H. Sahraoui, *AST-Probe: Recovering Abstract Syntax Trees from Hidden Representations of Pre-Trained Language Models*. New York, NY, USA: Association for Computing Machinery, 2023. 10

[238] Z. Zhang, H. Zhang, B. Shen, and X. Gu, "Diet code is healthy: simplifying programs for pre-trained models of code," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1073–1084. [Online]. Available: https://doi.org/10.1145/3540250.3549094 10

[239] E. Shi, Y. Wang, H. Zhang, L. Du, S. Han, D. Zhang, and H. Sun, "Towards efficient fine-tuning of pre-trained code models: An experimental study and beyond," in *ISSTA*, 2023. 10

[240] X. Wang, H. Wang, and D. Yang, "Measure and improve robustness in NLP models: A survey," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, Jul. 2022, pp. 4569–4586. [Online]. Available: https:

//aclanthology.org/2022.naacl-main.339 10
[241] Q. Sun, N. Chen, J. Wang, and X. Li, "Rethinking the role of structural information: How it enhances code representation learning?" in *International Joint Conference on Neural Networks, IJCNN 2024, Yokohama, Japan, June 30 - July 5, 2024*. IEEE, 2024. 10
[242] N. Yefet, U. Alon, and E. Yahav, "Adversarial examples for models of code," *Proc. ACM Program. Lang.*, vol. 4, no. OOPSLA, nov 2020. [Online]. Available: https://doi.org/10.1145/3428230 10
[243] J. Henkel, G. Ramakrishnan, Z. Wang, A. Albarghouthi, S. Jha, and T. Reps, "Semantic robustness of models of source code," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2022, pp. 526–537. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SANER53432.2022.00070 10
[244] Z. Yang, J. Shi, J. He, and D. Lo, "Natural attack for pre-trained models of code," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1482–1493. [Online]. Available: https://doi.org/10.1145/3510003.3510146 10
[245] A. Jha and C. K. Reddy, "Codeattack: Code-based adversarial attacks for pre-trained programming language models," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. 10
[246] L. Li, R. Ma, Q. Guo, X. Xue, and X. Qiu, "BERT-ATTACK: Adversarial attack against BERT using BERT," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6193–6202. [Online]. Available: https://aclanthology.org/2020.emnlp-main.500 10
[247] Z. Zhao, M. M. Das, and F. H. Fard, "Studying vulnerable code entities in r," 2024. 10
[248] J. Zhang, W. Ma, Q. Hu, S. Liu, X. Xie, Y. Le Traon, and Y. Liu, "A black-box attack on code models via representation nearest neighbor search," in *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9706–9716. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.649 10
[249] N. Chen, Q. Sun, J. Wang, M. Gao, X. Li, and X. Li, "Evaluating and enhancing the robustness of code pre-trained models through structure-aware adversarial samples generation," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14 857–14 873. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.991 10
[250] R. Schuster, C. Song, E. Tromer, and V. Shmatikov, "You autocomplete me: Poisoning vulnerabilities in neural code completion," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1559–1575. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/schuster 10
[251] J. Li, Z. Li, H. Zhang, G. Li, Z. Jin, X. Hu, and X. Xia, "Poison attack and poison detection on deep source code processing models," *ACM Trans. Softw. Eng. Methodol.*, nov 2023, just Accepted. [Online]. Available: https://doi.org/10.1145/3630008 10
[252] Z. Yang, B. Xu, J. M. Zhang, H. J. Kang, J. Shi, J. He, and D. Lo, "Stealthy backdoor attack for code models," 2023. 10
[253] Y. Li, S. Liu, K. Chen, X. Xie, T. Zhang, and Y. Liu, "Multi-target backdoor attacks for code pre-trained models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 7236–7254. [Online]. Available: https://aclanthology.org/2023.acl-long.399 10
[254] S. Wang, Z. Li, H. Qian, C. Yang, Z. Wang, M. Shang, V. Kumar, S. Tan, B. Ray, P. Bhatia, R. Nallapati, M. K. Ramanathan, D. Roth, and B. Xiang, "ReCode: Robustness evaluation of code generation models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 13 818–13 843. [Online]. Available: https://aclanthology.org/2023.acl-long.773 10
[255] F. Wu, X. Liu, and C. Xiao, "Deceptprompt: Exploiting llm-driven code generation via adversarial natural language instructions," 2023. 10
[256] OpenAI, "GPT-4 technical report," 2023. 10
[257] G. Team, "Gemini: A family of highly capable multimodal models," 2023. 10
[258] D. Groeneveld, I. Beltagy, P. Walsh, A. Bhagia, R. Kinney, O. Tafjord, A. H. Jha, H. Ivison, I. Magnusson, Y. Wang, S. Arora, D. Atkinson, R. Authur, K. R. Chandu, A. Cohan, J. Dumas, Y. Elazar, Y. Gu, J. Hessel, T. Khot, W. Merrill, J. Morrison, N. Muennighoff, A. Naik, C. Nam, M. E. Peters, V. Pyatkin, A. Ravichander, D. Schwenk, S. Shah, W. Smith, E. Strubell, N. Subramani, M. Wortsman, P. Dasigi, N. Lambert, K. Richardson, L. Zettlemoyer, J. Dodge, K. Lo, L. Soldaini, N. A. Smith, and H. Hajishirzi, "Olmo: Accelerating the science of language models," 2024. 10
[259] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg *et al.*, "On the opportunities and risks of foundation models," *ArXiv*, 2021. [Online]. Available: https://crfm.stanford.edu/assets/report.pdf 10
[260] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022. 10
[261] A. D. Cohen, A. Roberts, A. Molina, A. Butryna, A. Jin, A. Kulshreshtha, B. Hutchinson, B. Zevenbergen, B. H. Aguera-Arcas, C. ching Chang, C. Cui, C. Du, D. D. F. Adiwardana, D. Chen, D. D. Lepikhin, E. H. Chi, E. Hoffman-John, H.-T. Cheng, H. Lee, I. Krivokon, J. Qin, J. Hall, J. Fenton, J. Soraker, K. Meier-Hellstern,

K. Olson, L. M. Aroyo, M. P. Bosma, M. J. Pickett, M. A. Menegali, M. Croak, M. Díaz, M. Lamm, M. Krikun, M. R. Morris, N. Shazeer, Q. V. Le, R. Bernstein, R. Rajakumar, R. Kurzweil, R. Thoppilan, S. Zheng, T. Bos, T. Duke, T. Doshi, V. Y. Zhao, V. Prabhakaran, W. Rusch, Y. Li, Y. Huang, Y. Zhou, Y. Xu, and Z. Chen, "Lamda: Language models for dialog applications," in *arXiv*, 2022. 10

[262] B. Workshop, "Bloom: A 176b-parameter open-access multilingual language model," 2023. 10

[263] H. Laurençon, L. Saulnier, T. Wang, C. Akiki, A. V. del Moral, T. L. Scao, L. V. Werra, C. Mou, E. G. Ponferrada, H. Nguyen, J. Frohberg, M. Šaško, Q. Lhoest, A. McMillan-Major, G. Dupont, S. Biderman, A. Rogers, L. B. allal, F. D. Toni, G. Pistilli, O. Nguyen, S. Nikpoor, M. Masoud, P. Colombo, J. de la Rosa, P. Villegas, T. Thrush, S. Longpre, S. Nagel, L. Weber, M. R. Muñoz, J. Zhu, D. V. Strien, Z. Alyafeai, K. Almubarak, V. M. Chien, I. Gonzalez-Dios, A. Soroa, K. Lo, M. Dey, P. O. Suarez, A. Gokaslan, S. Bose, D. I. Adelani, L. Phan, H. Tran, I. Yu, S. Pai, J. Chim, V. Lepercq, S. Ilic, M. Mitchell, S. Luccioni, and Y. Jernite, "The bigscience ROOTS corpus: A 1.6TB composite multilingual dataset," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. [Online]. Available: https://openreview.net/forum?id=UoEw6KigkUn 10

[264] O. Press, N. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=R8sQPpGCv0 10

[265] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," 2023. 10

[266] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019. 10, 12

[267] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022. 10

[268] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 4895–4901. [Online]. Available: https://aclanthology.org/2023.emnlp-main.298 10

[269] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022. 10

[270] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *CoRR*, vol. abs/2302.13971, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.13971 10

[271] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," March 2023. [Online]. Available: https://lmsys.org/blog/2023-03-30-vicuna/ 10

[272] I. Team, "Internlm: A multilingual language model with progressively enhanced capabilities," https://github.com/InternLM/InternLM-techreport, 2023. 10

[273] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022. 11, 12, 13, 16, 17, 30

[274] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, and S. C. H. Hoi, "Codet5+: Open code large language models for code understanding and generation," *arXiv preprint*, 2023. 11, 12, 13

[275] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," in *The Eleventh International Conference on Learning Representations*, 2023. 11, 13, 16, 30

[276] D. Kocetkov, R. Li, L. B. Allal, J. Li, C. Mou, C. M. Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf, D. Bahdanau, L. von Werra, and H. de Vries, "The stack: 3 tb of permissively licensed source code," 2022. 11

[277] R. Li, L. B. allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. LI, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, J. Lamy-Poirier, J. Monteiro, N. Gontier, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. T. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, U. Bhattacharyya, W. Yu, S. Luccioni, P. Villegas, F. Zhdanov, T. Lee, N. Timor, J. Ding, C. S. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. V. Werra, and H. de Vries, "Starcoder: may the source be with you!" *Transactions on Machine Learning Research*, 2023, reproducibility Certification. [Online]. Available: https://openreview.net/forum?id=KoFOg41haE 11, 12, 13, 14

[278] A. Lozhkov, R. Li, L. B. Allal, F. Cassano, J. Lamy-Poirier, N. Tazi, A. Tang, D. Pykhtar, J. Liu, Y. Wei, T. Liu, M. Tian, D. Kocetkov, A. Zucker, Y. Belkada, Z. Wang, Q. Liu, D. Abulkhanov, I. Paul, Z. Li, W.-D. Li, M. Risdal, J. Li, J. Zhu, T. Y. Zhuo, E. Zheltonozhskii, N. O. O. Dade, W. Yu, L. Krauß, N. Jain, Y. Su, X. He, M. Dey, E. Abati, Y. Chai, N. Muennighoff, X. Tang, M. Oblokulov, C. Akiki, M. Marone, C. Mou, M. Mishra, A. Gu, B. Hui, T. Dao, A. Zebaze, O. Dehaene, N. Patry, C. Xu, J. McAuley, H. Hu, T. Scholak,

S. Paquet, J. Robinson, C. J. Anderson, N. Chapados, M. Patwary, N. Tajbakhsh, Y. Jernite, C. M. Ferrandis, L. Zhang, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder 2 and the stack v2: The next generation," 2024. 11, 12, 13

[279] W. Sun, C. Fang, Y. You, Y. Chen, Y. Liu, C. Wang, J. Zhang, Q. Zhang, H. Qian, W. Zhao, Y. Liu, and Z. Chen, "A prompt learning framework for source code summarization," 2023. 11

[280] H. Joshi, J. C. Sanchez, S. Gulwani, V. Le, I. Radiček, and G. Verbruggen, "Repair is nearly generation: multilingual program repair with llms," in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i4.25642 11

[281] H. Fu, Yao; Peng and T. Khot, "How does gpt obtain its ability? tracing emergent abilities of language models to their sources," *Yao Fu's Notion*, Dec 2022. 11, 17, 19

[282] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, "Training language models to follow instructions with human feedback," 2022. 11, 15

[283] E. Nijkamp, H. Hayashi, C. Xiong, S. Savarese, and Y. Zhou, "Codegen2: Lessons for training llms on programming and natural languages," 2023. 12, 13

[284] T. Sun, X. Zhang, Z. He, P. Li, Q. Cheng, H. Yan, X. Liu, Y. Shao, Q. Tang, X. Zhao, K. Chen, Y. Zheng, Z. Zhou, R. Li, J. Zhan, Y. Zhou, L. Li, X. Yang, L. Wu, Z. Yin, X. Huang, and X. Qiu, "Moss: Training conversational language models from synthetic data," 2023. 12

[285] CodeParrot, "Codeparrot." [Online]. Available: https://huggingface.co/codeparrot/codeparrot 13

[286] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–10. 13, 26

[287] F. Christopoulou, G. Lampouras, M. Gritta, G. Zhang, Y. Guo, Z. Li, Q. Zhang, M. Xiao, B. Shen, L. Li, H. Yu, L. Yan, P. Zhou, X. Wang, Y. Ma, I. Iacobacci, Y. Wang, G. Liang, J. Wei, X. Jiang, Q. Wang, and Q. Liu, "Pangu-coder: Program synthesis with function-level language modeling," 2022. 13

[288] L. B. Allal, R. Li, D. Kocetkov, C. Mou, C. Akiki, C. M. Ferrandis, N. Muennighoff, M. Mishra, A. Gu, M. Dey, L. K. Umapathi, C. J. Anderson, Y. Zi, J. L. Poirier, H. Schoelkopf, S. Troshin, D. Abulkhanov, M. Romero, M. Lappert, F. D. Toni, B. G. del Río, Q. Liu, S. Bose, U. Bhattacharyya, T. Y. Zhuo, I. Yu, P. Villegas, M. Zocca, S. Mangrulkar, D. Lansky, H. Nguyen, D. Contractor, L. Villa, J. Li, D. Bahdanau, Y. Jernite, S. Hughes, D. Fried, A. Guha, H. de Vries, and L. von Werra, "Santacoder: don't reach for the stars!" 2023. 12, 13

[289] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. D. Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, and Y. Li, "Textbooks are all you need," 2023. 13, 14

[290] Q. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, Z. Wang, L. Shen, A. Wang, Y. Li, T. Su, Z. Yang, and J. Tang, "Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x," in *KDD*, 2023. 12, 13, 14, 16, 20, 25, 30

[291] S. Chaudhary, "Code alpaca: An instruction-following llama model for code generation," https://github.com/sahil280114/codealpaca, 2023. 13, 14

[292] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," 2023. 13, 14

[293] BAAI, "Aquilacode," 2023. [Online]. Available: https://huggingface.co/BAAI/AquilaCode-multi 13

[294] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023. 13, 14, 19, 21

[295] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, M. Huang, N. Duan, and W. Chen, "Tora: A tool-integrated reasoning agent for mathematical problem solving," 2023. 13, 14

[296] X. Yue, X. Qu, G. Zhang, Y. Fu, W. Huang, H. Sun, Y. Su, and W. Chen, "MAmmoTH: Building math generalist models through hybrid instruction tuning," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=yLClGs770I 13, 14

[297] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu, "Qwen technical report," 2023. 13, 14

[298] B. Liu, C. Chen, C. Liao, Z. Gong, H. Wang, Z. Lei, M. Liang, D. Chen, M. Shen, H. Zhou, H. Yu, and J. Li, "Mftcoder: Boosting code llms with multitask fine-tuning," 2023. 13, 15

[299] WisdomShell, "Aquilacode," 2023. [Online]. Available: https://huggingface.co/WisdomShell/CodeShell-7B 13

[300] Y. Xu, H. Su, C. Xing, B. Mi, Q. Liu, W. Shi, B. Hui, F. Zhou, Y. Liu, T. Xie, Z. Cheng, S. Zhao, L. Kong, B. Wang, C. Xiong, and T. Yu, "Lemur: Harmonizing natural language and code for language agents," 2023. 13, 14, 22

[301] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming – the rise of code intelligence," 2024. 13, 19, 21

[302] F. Xu, Z. Wu, Q. Sun, S. Ren, F. Yuan, S. Yuan, Q. Lin, Y. Qiao, and J. Liu, "Symbol-llm: Towards foundational symbol-centric interface for large language models," 2023. 13, 14, 26

[303] N. Pinnaparaju, R. Adithyan, D. Phung, J. Tow, J. Baicoianu, , and N. Cooper, "Stable code 3b." [Online]. Available: https://huggingface.co/stabilityai/stable-code-3b 13, 14, 24

[304] D. R. Team, "Decicoder," 2023. [Online]. Available: https://huggingface.co/deci/decicoder-1b 13, 14, 24

[305] CodeGemma Team, "Codegemma: Open code models based on gemma," 2024. 13, 14

[306] L. Tunstall, N. Lambert, N. Rajani, E. Beeching, T. Le Scao, L. von Werra, S. Han, P. Schmid, and A. Rush, "Creating a coding assistant with starcoder," *Hugging Face Blog*, 2023, https://huggingface.co/blog/starchat-alpha. 12, 20

[307] N. Muennighoff, Q. Liu, A. Zebaze, Q. Zheng, B. Hui, T. Y. Zhuo, S. Singh, X. Tang, L. von Werra, and S. Longpre, "Octopack: Instruction tuning code large language models," 2023. 12, 27, 29

[308] C. Lattner and V. Adve, "Llvm: a compilation framework for lifelong program analysis & transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75–86. 12

[309] S. Soltan, S. Ananthakrishnan, J. FitzGerald, R. Gupta, W. Hamza, H. Khan, C. Peris, S. Rawls, A. Rosenbaum, A. Rumshisky, C. S. Prakash, M. Sridhar, F. Triefenbach, A. Verma, G. Tur, and P. Natarajan, "Alexatm 20b: Few-shot learning using a large-scale multilingual seq2seq model," 2022. 12

[310] Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, J. Wei, X. Wang, H. W. Chung, D. Bahri, T. Schuster, S. Zheng, D. Zhou, N. Houlsby, and D. Metzler, "UL2: Unifying language learning paradigms," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=6ruVLB727MC 12

[311] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023. 13, 14

[312] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-instruct: Aligning language models with self-generated instructions," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 13 484–13 508. [Online]. Available: https://aclanthology.org/2023.acl-long.754 13

[313] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. 13

[314] S. Chen, S. Wong, L. Chen, and Y. Tian, "Extending context window of large language models via positional interpolation," 2023. 13, 14

[315] Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck, "Llemma: An open language model for mathematics," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=4WnqRR915j 13, 19

[316] A. Liu, X. Han, Y. Wang, Y. Tsvetkov, Y. Choi, and N. A. Smith, "Tuning language models by proxy," 2024. 13

[317] D. Song, H. Guo, Y. Zhou, S. Xing, Y. Wang, Z. Song, W. Zhang, Q. Guo, H. Yan, X. Qiu, and D. Lin, "Code needs comments: Enhancing code llms with comment augmentation," 2024. 13

[318] S. Lin, J. Hilton, and O. Evans, "TruthfulQA: Measuring how models mimic human falsehoods," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3214–3252. [Online]. Available: https://aclanthology.org/2022.acl-long.229 13

[319] T. Hartvigsen, S. Gabriel, H. Palangi, M. Sap, D. Ray, and E. Kamar, "ToxiGen: A large-scale machine-generated dataset for adversarial and implicit hate speech detection," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3309–3326. [Online]. Available: https://aclanthology.org/2022.acl-long.234 13

[320] J. Dhamala, T. Sun, V. Kumar, S. Krishna, Y. Pruksachatkun, K.-W. Chang, and R. Gupta, "Bold: Dataset and metrics for measuring biases in open-ended language generation," in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 862–872. [Online]. Available: https://doi.org/10.1145/3442188.3445924 13

[321] DeepSeek-AI, "Deepseek llm: Scaling open-source language models with longtermism," 2024. 14

[322] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," 2024. 14, 19

[323] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147. 14, 22

[324] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022. 14, 22

[325] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring

massive multitask language understanding," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=d7KBjmI3GmQ 14

[326] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. Le, E. Chi, D. Zhou, and J. Wei, "Challenging BIG-bench tasks and whether chain-of-thought can solve them," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 13 003–13 051. [Online]. Available: https://aclanthology.org/2023.findings-acl.824 14, 16, 18, 30

[327] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W.-t. Yih, D. Fried, S. Wang, and T. Yu, "Ds-1000: A natural and reliable benchmark for data science code generation," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023. 14, 16, 22, 30

[328] X. Wang, Z. Wang, J. Liu, Y. Chen, L. Yuan, H. Peng, and H. Ji, "MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=jp3gWrMuIZ 14

[329] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang, "GLM: General language model pretraining with autoregressive blank infilling," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 320–335. [Online]. Available: https://aclanthology.org/2022.acl-long.26 14

[330] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia, W. L. Tam, Z. Ma, Y. Xue, J. Zhai, W. Chen, Z. Liu, P. Zhang, Y. Dong, and J. Tang, "GLM-130b: An open bilingual pre-trained model," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=-Aw0rrrPUF 14

[331] Gemma Team, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024. 14

[332] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang, "Wizardlm: Empowering large language models to follow complex instructions," 2023. 14

[333] Z. Yu, X. Zhang, N. Shang, Y. Huang, C. Xu, Y. Zhao, W. Hu, and Q. Yin, "Wavecoder: Widespread and versatile enhanced instruction tuning with refined data generation," 2023. 14

[334] Y. Wang, K. He, G. Dong, P. Wang, W. Zeng, M. Diao, Y. Mou, M. Zhang, J. Wang, X. Cai, and W. Xu, "Dolphcoder: Echo-locating code large language models with diverse and multi-objective instruction tuning," 2024. 14

[335] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=_VjQlMeSB_J 14, 18, 19, 24

[336] J. Li, P. Chen, and J. Jia, "Motcoder: Elevating large language models with modular of thought for challenging programming tasks," 2024. 14

[337] T. Cai, X. Wang, T. Ma, X. Chen, and D. Zhou, "Large language models as tool makers," 2023. 14

[338] Z. Z. Wang, Z. Cheng, H. Zhu, D. Fried, and G. Neubig, "What are tools anyway? a survey from the language model perspective," 2024. 14

[339] N. Jain, T. Zhang, W.-L. Chiang, J. E. Gonzalez, K. Sen, and I. Stoica, "Llm-assisted code cleaning for training accurate code generators," 2023. 14

[340] J. Tow, M. Bellagente, D. Mahan, and C. Riquelme, "Stablelm 3b 4e1t." [Online]. Available: https://huggingface.co/stabilityai/stablelm-3b-4e1t 14

[341] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. 15

[342] Y. Bai, A. Jones, K. Ndousse, A. Askell, A. Chen, N. DasSarma, D. Drain, S. Fort, D. Ganguli, T. Henighan, N. Joseph, S. Kadavath, J. Kernion, T. Conerly, S. El-Showk, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, T. Hume, S. Johnston, S. Kravec, L. Lovitt, N. Nanda, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, B. Mann, and J. Kaplan, "Training a helpful and harmless assistant with reinforcement learning from human feedback," 2022. 15

[343] X. Wang, Y. Wang, Y. Wan, F. Mi, Y. Li, P. Zhou, J. Liu, H. Wu, X. Jiang, and Q. Liu, "Compilable neural code generation with compiler feedback," in *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 9–19. [Online]. Available: https://aclanthology.org/2022.findings-acl.2 15

[344] P. Shojaee, A. Jain, S. Tipirneni, and C. K. Reddy, "Execution-based code generation using deep reinforcement learning," *Transactions on Machine Learning Research*, 2023. [Online]. Available: https://openreview.net/forum?id=0XBuaxqEcG 15

[345] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. 15

[346] J. Liu, Y. Zhu, K. Xiao, Q. FU, X. Han, Y. Wei, and D. Ye, "RLTF: Reinforcement learning from unit test feedback," *Transactions on Machine Learning Research*, 2023. [Online]. Available: https://openreview.net/forum?id=hjYmsV6nXZ 15

[347] A. Jain, C. Adiole, S. Chaudhuri, T. Reps, and C. Jermaine, "Coarse-tuning models of code with reinforcement learning feedback," 2023. 15

[348] B. Shen, J. Zhang, T. Chen, D. Zan, B. Geng, A. Fu, M. Zeng, A. Yu, J. Ji, J. Zhao, Y. Guo, and Q. Wang, "Pangu-coder2: Boosting large language models for

code with ranking feedback," 2023. 15

[349] K. Shi, J. Hong, Y. Deng, P. Yin, M. Zaheer, and C. Sutton, "Exedec: Execution decomposition for compositional generalization in neural program synthesis," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=oTRwljRgiv 15

[350] S. Dou, Y. Liu, H. Jia, L. Xiong, E. Zhou, W. Shen, J. Shan, C. Huang, X. Wang, X. Fan, Z. Xi, Y. Zhou, T. Ji, R. Zheng, Q. Zhang, X. Huang, and T. Gui, "Stepcoder: Improve code generation with reinforcement learning from compiler feedback," 2024. 15

[351] Z. Wang, S. Zhou, D. Fried, and G. Neubig, "Execution-based evaluation for open-domain code generation," in *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1271–1290. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.89 16, 30

[352] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton, "Program synthesis with large language models," 2021. 15, 16, 17, 30

[353] B. bench authors, "Beyond the imitation game: Quantifying and extrapolating the capabilities of language models," *Transactions on Machine Learning Research*, 2023. [Online]. Available: https://openreview.net/forum?id=uyTL5Bvosj 16, 17, 18, 30

[354] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, and J. Steinhardt, "Measuring coding challenge competence with APPS," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. [Online]. Available: https://openreview.net/forum?id=sD93GOzH3i5 16, 17, 24, 30

[355] B. Athiwaratkun, S. K. Gouda, Z. Wang, X. Li, Y. Tian, M. Tan, W. U. Ahmad, S. Wang, Q. Sun, M. Shang, S. K. Gonugondla, H. Ding, V. Kumar, N. Fulton, A. Farahani, S. Jain, R. Giaquinto, H. Qian, M. K. Ramanathan, R. Nallapati, B. Ray, P. Bhatia, S. Sengupta, D. Roth, and B. Xiang, "Multi-lingual evaluation of code generation models," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=Bo7eeXm6An8 16, 25, 30

[356] D. Zan, B. Chen, Z. Lin, B. Guan, W. Yongji, and J.-G. Lou, "When language model meets private library," in *Findings of the Association for Computational Linguistics: EMNLP 2022*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 277–288. [Online]. Available: https://aclanthology.org/2022.findings-emnlp.21 16, 30

[357] X. Tang, B. Qian, R. Gao, J. Chen, X. Chen, and M. Gerstein, "Biocoder: A benchmark for bioinformatics code generation with contextual pragmatic knowledge," 2023. 16, 23, 30

[358] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. 15

[359] C.-Y. Lin and F. J. Och, "ORANGE: a method for evaluating automatic evaluation metrics for machine translation," in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. Geneva, Switzerland: COLING, aug 23–aug 27 2004, pp. 501–507. 15

[360] M. Post, "A call for clarity in reporting BLEU scores," in *Proceedings of the Third Conference on Machine Translation: Research Papers*. Belgium, Brussels: Association for Computational Linguistics, Oct. 2018, pp. 186–191. [Online]. Available: https://www.aclweb.org/anthology/W18-6319 15

[361] A. B. Sai, A. K. Mohankumar, and M. M. Khapra, "A survey of evaluation metrics used for nlg systems," *ACM Comput. Surv.*, vol. 55, no. 2, jan 2022. [Online]. Available: https://doi.org/10.1145/3485766 15

[362] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, "Out of the bleu: How should we assess quality of the code generation models?" *J. Syst. Softw.*, vol. 203, no. C, sep 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111741 15

[363] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, M. Zhou, A. Blanco, and S. Ma, "Codebleu: a method for automatic evaluation of code synthesis," *arXiv preprint arXiv:2009.10297*, 2020. 15

[364] M. Popović, "chrF: character n-gram F-score for automatic MT evaluation," in *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 392–395. [Online]. Available: https://aclanthology.org/W15-3049 15

[365] N. Tran, H. Tran, S. Nguyen, H. Nguyen, and T. N. Nguyen, "Does bleu score work for code migration?" in *Proceedings of the 27th International Conference on Program Comprehension*, ser. ICPC '19. IEEE Press, 2019, p. 165–176. [Online]. Available: https://doi.org/10.1109/ICPC.2019.00034 15

[366] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," 2023. 16

[367] A. Ni, P. Yin, Y. Zhao, M. Riddell, T. Feng, R. Shen, S. Yin, Y. Liu, S. Yavuz, C. Xiong, S. Joty, Y. Zhou, D. Radev, and A. Cohan, "L2ceval: Evaluating language-to-code generation capabilities of large language models," 2023. 17

[368] Z. Lin, Y. Yao, and Y. Yuan, "Catcode: A comprehensive evaluation framework for llms on the mixture of code and text," 2024. 17

[369] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov, E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, and J. Wei, "Scaling instruction-finetuned language

models," 2022. 17

[370] S. Zhang, Z. Chen, Y. Shen, M. Ding, J. B. Tenenbaum, and C. Gan, "Planning with large language models for code generation," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=Lr8cOOtYbfL 17

[371] X. Jiang, Y. Dong, L. Wang, Z. Fang, Q. Shang, G. Li, Z. Jin, and W. Jiao, "Self-planning code generation with large language models," 2023. 17

[372] L. Zheng, J. Yuan, Z. Zhang, H. Yang, and L. Kong, "Self-infilling code generation," 2023. 17

[373] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Y. Zhuo, and T. Chen, "Chain-of-thought in neural code generation: From and for lightweight language models," 2023. 17

[374] J. Li, Y. Zhao, Y. Li, G. Li, and Z. Jin, "Acecoder: Utilizing existing code to enhance code generation," 2023. 17

[375] J. Li, G. Li, Y. Li, and Z. Jin, "Structured chain-of-thought prompting for code generation," 2023. 17

[376] E. Arteca, S. Harner, M. Pradel, and F. Tip, "Nessie: Automatically testing javascript apis with asynchronous callbacks," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1494–1505. 17

[377] F. Shi, D. Fried, M. Ghazvininejad, L. Zettlemoyer, and S. I. Wang, "Natural language to code translation with execution," 2022. 17

[378] P. Bareiß, B. Souza, M. d'Amorim, and M. Pradel, "Code generation tools (almost) for free? a study of few-shot, pre-trained language models on code," 2022. 17

[379] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J.-G. Lou, and W. Chen, "Codet: Code generation with generated tests," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=ktrw68Cmu9c 17

[380] S. K. Lahiri, S. Fakhoury, A. Naik, G. Sakkas, S. Chakraborty, M. Musuvathi, P. Choudhury, C. von Veh, J. P. Inala, C. Wang, and J. Gao, "Interactive code generation via test-driven user-intent formalization," 2023. 17

[381] A. Ni, S. Iyer, D. Radev, V. Stoyanov, W.-t. Yih, S. I. Wang, and X. V. Lin, "Lever: Learning to verify language-to-code generation with execution," in *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, 2023. 17

[382] K. Zhang, D. Wang, J. Xia, W. Y. Wang, and L. Li, "ALGO: Synthesizing algorithmic programs with generated oracle verifiers," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=JolrEmMim6 17

[383] B. Roziere, J. Zhang, F. Charton, M. Harman, G. Synnaeve, and G. Lample, "Leveraging automated unit tests for unsupervised code translation," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=cmt-6KtR4c4 17

[384] H. Han, M. Kim, S.-w. Hwang, N. Duan, and S. Lu, "Intervention-based alignment of code search with execution feedback," in *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 2241–2263. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.148 17

[385] T. Zhang, T. Yu, T. B. Hashimoto, M. Lewis, W.-t. Yih, D. Fried, and S. I. Wang, "Coder reviewer reranking for code generation," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023. 17

[386] S. Zhou, U. Alon, F. F. Xu, Z. Jiang, and G. Neubig, "Docprompting: Generating code by retrieving the docs," in *International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, May 2023. 17

[387] T. Ridnik, D. Kredo, and I. Friedman, "Code generation with alphacodium: From prompt engineering to flow engineering," 2024. 17

[388] J. Sun, C. Zheng, E. Xie, Z. Liu, R. Chu, J. Qiu, J. Xu, M. Ding, H. Li, M. Geng, Y. Wu, W. Wang, J. Chen, Z. Yin, X. Ren, J. Fu, J. He, W. Yuan, Q. Liu, X. Liu, Y. Li, H. Dong, Y. Cheng, M. Zhang, P. A. Heng, J. Dai, P. Luo, J. Wang, J.-R. Wen, X. Qiu, Y. Guo, H. Xiong, Q. Liu, and Z. Li, "A survey of reasoning with foundation models," 2023. 18

[389] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena, "Show your work: Scratchpads for intermediate computation with language models," in *Deep Learning for Code Workshop*, 2022. [Online]. Available: https://openreview.net/forum?id=HBlx2idbkbq 18

[390] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=e2TBb5y0yFf 18

[391] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=1PL1NIMMrw 18

[392] Y. Fu, H. Peng, A. Sabharwal, P. Clark, and T. Khot, "Complexity-based prompting for multi-step reasoning," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=yf1icZHC-l9 18, 19

[393] J. Wang, Q. Sun, X. Li, and M. Gao, "Boosting language models reasoning with chain-of-knowledge prompting," 2023. 18

[394] Z. Chu, J. Chen, Q. Chen, W. Yu, T. He, H. Wang, W. Peng, M. Liu, B. Qin, and T. Liu, "A survey of chain of thought reasoning: Advances, frontiers and future," 2023. 18

[395] Z. Yin, Q. Sun, Q. Guo, J. Wu, X. Qiu, and X. Huang, "Do large language models know what they don't know?" in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto,

Canada: Association for Computational Linguistics, Jul. 2023, pp. 8653–8665. [Online]. Available: https://aclanthology.org/2023.findings-acl.551 18

[396] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, mar 2023. [Online]. Available: https://doi.org/10.1145/3571730 18

[397] R. Nogueira, Z. Jiang, and J. Lin, "Investigating the limitations of transformers with simple arithmetic tasks," 2021. 18

[398] J. Qian, H. Wang, Z. Li, S. Li, and X. Yan, "Limitations of language models in arithmetic and symbolic induction," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 9285–9298. [Online]. Available: https://aclanthology.org/2023.acl-long.516 18

[399] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "PAL: Program-aided language models," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 10 764–10 799. [Online]. Available: https://proceedings.mlr.press/v202/gao23f.html 18, 24, 26

[400] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," 2022. 18, 24

[401] A. Zhou, K. Wang, Z. Lu, W. Shi, S. Luo, Z. Qin, S. Lu, A. Jia, L. Song, M. Zhan, and H. Li, "Solving challenging math word problems using gpt-4 code interpreter with code-based self-verification," 2023. 18

[402] M. Schubotz, P. Scharpf, K. Dudhat, Y. Nagar, F. Hamborg, and B. Gipp, "Introducing mathqa: a math-aware question answering system," *Information Discovery and Delivery*, vol. 46, no. 4, p. 214–224, Nov. 2018. [Online]. Available: http://dx.doi.org/10.1108/IDD-06-2018-0022 18

[403] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," 2021. 18

[404] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the MATH dataset," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. [Online]. Available: https://openreview.net/forum?id=7Bywt2mQsCe 18, 19

[405] Z. Chen, W. Chen, C. Smiley, S. Shah, I. Borova, D. Langdon, R. Moussa, M. Beane, T.-H. Huang, B. Routledge, and W. Y. Wang, "FinQA: A dataset of numerical reasoning over financial data," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3697–3711. [Online]. Available: https://aclanthology.org/2021.emnlp-main.300 18

[406] Z. Chen, S. Li, C. Smiley, Z. Ma, S. Shah, and W. Y. Wang, "ConvFinQA: Exploring the chain of numerical reasoning in conversational finance question answering," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 6279–6292. [Online]. Available: https://aclanthology.org/2022.emnlp-main.421 18

[407] K. Wang, H. Ren, A. Zhou, Z. Lu, S. Luo, W. Shi, R. Zhang, L. Song, M. Zhan, and H. Li, "Mathcoder: Seamless code integration in LLMs for enhanced mathematical reasoning," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=z8TW0ttBPp 18

[408] J. Zhao, Y. Xie, K. Kawaguchi, J. He, and M. Xie, "Automatic model selection with large language models for reasoning," in *Findings of the Association for Computational Linguistics: EMNLP 2023*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 758–783. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.55 18

[409] Y. Xie, K. Kawaguchi, Y. Zhao, X. Zhao, M.-Y. Kan, J. He, and Q. Xie, "Self-evaluation guided beam search for reasoning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=Bw82hwg5Q3 18

[410] P. Lu, B. Peng, H. Cheng, M. Galley, K.-W. Chang, Y. N. Wu, S.-C. Zhu, and J. Gao, "Chameleon: Plug-and-play compositional reasoning with large language models," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=HtqnVSCj3q 18

[411] Z. Yin, Q. Sun, C. Chang, Q. Guo, J. Dai, X. Huang, and X. Qiu, "Exchange-of-thought: Enhancing large language model capabilities through cross-model communication," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 15 135–15 153. [Online]. Available: https://aclanthology.org/2023.emnlp-main.936 19

[412] Q. Sun, Z. Yin, X. Li, Z. Wu, X. Qiu, and L. Kong, "Corex: Pushing the boundaries of complex reasoning through multi-model collaboration," 2023. 19

[413] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta programming for multi-agent collaborative framework," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=VtmBAGCN7o 19

[414] W. Chen, C. Yuan, J. Yuan, Y. Su, C. Qian, C. Yang, R. Xie, Z. Liu, and M. Sun, "Beyond natural language: Llms leveraging alternative formats for enhanced reasoning and communication," 2024. 19

[415] J. He-Yueya, G. Poesia, R. E. Wang, and N. D. Goodman, "Solving math word problems by combining language models with symbolic solvers," 2023. 19

[416] F. Xu, Q. Lin, J. Han, T. Zhao, J. Liu, and E. Cambria, "Are large language models really good logical reasoners? A comprehensive evaluation from deductive, inductive and abductive views," *CoRR*, vol. abs/2306.09841, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.09841 19

[417] C. Li, J. Liang, A. Zeng, X. Chen, K. Hausman, D. Sadigh, S. Levine, L. Fei-Fei, F. Xia, and B. Ichter, "Chain of code: Reasoning with a language model-augmented code emulator," 2023. 19

[418] Q. Lyu, S. Havaldar, A. Stein, L. Zhang, D. Rao, E. Wong, M. Apidianaki, and C. Callison-Burch, "Faithful chain-of-thought reasoning," in *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. Nusa Dua, Bali: Association for Computational Linguistics, Nov. 2023, pp. 305–329. [Online]. Available: https://aclanthology.org/2023.ijcnlp-main.20 19

[419] J. Ye, C. Li, L. Kong, and T. Yu, "Generating data for symbolic language with large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 8418–8443. [Online]. Available: https://aclanthology.org/2023.emnlp-main.523 19

[420] E. L. Malfa, C. Weinhuber, O. Torre, F. Lin, A. Cohn, N. Shadbolt, and M. Wooldridge, "Code simulation challenges for large language models," 2024. 19

[421] M.-C. Dinu, C. Leoveanu-Condrei, M. Holzleitner, W. Zellinger, and S. Hochreiter, "Symbolicai: A framework for logic-based approaches combining generative models and solvers," 2024. 19

[422] Z. Bi, N. Zhang, Y. Jiang, S. Deng, G. Zheng, and H. Chen, "When do program-of-thoughts work for reasoning?" 2023. 19, 24

[423] H. Luo, Q. Sun, C. Xu, P. Zhao, J. Lou, C. Tao, X. Geng, Q. Lin, S. Chen, and D. Zhang, "Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct," 2023. 19

[424] J. Ahn, R. Verma, R. Lou, D. Liu, R. Zhang, and W. Yin, "Large language models for mathematical reasoning: Progresses and challenges," 2024. 19

[425] I. Drori and N. Verma, "Solving linear algebra by program synthesis," *ArXiv*, vol. abs/2111.08171, 2021. 19

[426] L. Tang, E. Ke, N. Singh, N. Verma, and I. Drori, "Solving probability and statistics problems by program synthesis," 2021. 19

[427] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang, "Is ChatGPT a general-purpose natural language processing task solver?" in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1339–1384. [Online]. Available: https://aclanthology.org/2023.emnlp-main.85 19

[428] Y. Ma, Y. Liu, Y. Yu, Y. Zhang, Y. Jiang, C. Wang, and S. Li, "At which training stage does code data help llms reasoning?" 2023. 19

[429] S. Toshniwal, I. Moshkov, S. Narenthiran, D. Gitman, F. Jia, and I. Gitman, "Openmathinstruct-1: A 1.8 million math instruction tuning dataset," 2024. 19

[430] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari, and V. Misra, "Solving quantitative reasoning problems with language models," 2022. 19

[431] H. Ying, S. Zhang, L. Li, Z. Zhou, Y. Shao, Z. Fei, Y. Ma, J. Hong, K. Liu, Z. Wang, Y. Wang, Z. Wu, S. Li, F. Zhou, H. Liu, S. Zhang, W. Zhang, H. Yan, X. Qiu, J. Wang, K. Chen, and D. Lin, "Internlm-math: Open math large language models toward verifiable reasoning," 2024. 19

[432] Y. Lu, Q. Liu, D. Dai, X. Xiao, H. Lin, X. Han, L. Sun, and H. Wu, "Unified structure generation for universal information extraction," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 5755–5772. [Online]. Available: https://aclanthology.org/2022.acl-long.395 19

[433] D. Xu, W. Chen, W. Peng, C. Zhang, T. Xu, X. Zhao, X. Wu, Y. Zheng, and E. Chen, "Large language models for generative information extraction: A survey," 2023. 19

[434] X. Wang, S. Li, and H. Ji, "Code4Struct: Code generation for few-shot event structure prediction," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 3640–3663. [Online]. Available: https://aclanthology.org/2023.acl-long.202 19

[435] P. Li, T. Sun, Q. Tang, H. Yan, Y. Wu, X. Huang, and X. Qiu, "CodeIE: Large code generation models are better few-shot information extractors," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 15 339–15 353. [Online]. Available: https://aclanthology.org/2023.acl-long.855 19

[436] Z. Bi, J. Chen, Y. Jiang, F. Xiong, W. Guo, H. Chen, and N. Zhang, "Codekgc: Code language model for generative knowledge graph construction," *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, feb 2024. [Online]. Available: https://doi.org/10.1145/3641850 19

[437] Y. Guo, Z. Li, X. Jin, Y. Liu, Y. Zeng, W. Liu, X. Li, P. Yang, L. Bai, J. Guo, and X. Cheng, "Retrieval-augmented code generation for universal information extraction," 2023. 19

[438] O. Sainz, I. García-Ferrero, R. Agerri, O. L. de Lacalle, G. Rigau, and E. Agirre, "GoLLIE: Annotation guidelines improve zero-shot information-extraction," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https:

//openreview.net/forum?id=Y3wpuxd7u9 20

[439] T. Xie, C. H. Wu, P. Shi, R. Zhong, T. Scholak, M. Yasunaga, C.-S. Wu, M. Zhong, P. Yin, S. I. Wang, V. Zhong, B. Wang, C. Li, C. Boyle, A. Ni, Z. Yao, D. Radev, C. Xiong, L. Kong, R. Zhang, N. A. Smith, L. Zettlemoyer, and T. Yu, "UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 602–631. [Online]. Available: https://aclanthology.org/2022.emnlp-main.39 20

[440] A. Madaan, S. Zhou, U. Alon, Y. Yang, and G. Neubig, "Language models of code are few-shot commonsense learners," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 1384–1403. 20

[441] L. Logeswaran, S. Sohn, Y. Lyu, A. Z. Liu, D.-K. Kim, D. Shim, M. Lee, and H. Lee, "Code models are zero-shot precondition reasoners," 2023. 20

[442] Y. Jiang, F. Ilievski, and K. Ma, "Transferring procedural knowledge across commonsense tasks," 2023. 20

[443] B. Bogin, S. Gupta, P. Clark, and A. Sabharwal, "Leveraging code to improve in-context learning for semantic parsing," 2023. 20

[444] Y. Dong, L. Martin, and C. Callison-Burch, "CoRRPUS: Code-based structured prompting for neurosymbolic story understanding," in *Findings of the Association for Computational Linguistics: ACL 2023*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 13 152–13 168. [Online]. Available: https://aclanthology.org/2023.findings-acl.832 20

[445] J. Wang, J. Wu, Y. Hou, Y. Liu, M. Gao, and J. McAuley, "Instructgraph: Boosting large language models via graph-centric instruction tuning and preference alignment," 2024. 20

[446] T. Gupta and A. Kembhavi, "Visual programming: Compositional visual reasoning without training," in *Computer Vision and Pattern Recognition*, 2022. 20

[447] D. Surís, S. Menon, and C. Vondrick, "Vipergpt: Visual inference via python execution for reasoning," in *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*. IEEE, 2023, pp. 11 854–11 864. [Online]. Available: https://doi.org/10.1109/ICCV51070.2023.01092 20, 22

[448] Y. Chen, X. Wang, M. Li, D. Hoiem, and H. Ji, "ViStruct: Visual structural knowledge extraction via curriculum guided code-vision representation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 13 342–13 357. [Online]. Available: https://aclanthology.org/2023.emnlp-main.824 20

[449] P. Sharma, T. R. Shaham, M. Baradad, S. Fu, A. Rodriguez-Munoz, S. Duggal, P. Isola, and A. Torralba, "A vision check-up for language models," 2024. 20

[450] T. Yoneda, J. Fang, P. Li, H. Zhang, T. Jiang, S. Lin, B. Picker, D. Yunis, H. Mei, and M. R. Walter, "Statler: State-maintaining language models for embodied reasoning," 2023. 20, 22

[451] G. Copilot, "Github copilot." [Online]. Available: https://github.com/features/copilot/ 20

[452] M. Tabachnyk and S. Nikolov, "ML-enhanced code completion improves developer productivity," 2022. 20

[453] CodeWhisperer, "Amazon codewhisperer." [Online]. Available: https://aws.amazon.com/codewhisperer/ 20

[454] Tabnine, "The ai coding assistant that you control." [Online]. Available: https://www.tabnine.com/ 20

[455] M. L. Siddiq, A. Samee, S. R. Azgor, M. A. Haider, S. I. Sawraz, and J. C. S. Santos, "Zero-shot prompting for code complexity prediction using github copilot," in *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, 2023, pp. 56–59. 20

[456] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 21–29. [Online]. Available: https://doi.org/10.1145/3520312.3534864 20

[457] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," 2023. 20

[458] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, 2023. 20

[459] N. Nguyen and S. Nadi, "An empirical evaluation of github copilot's code suggestions," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 1–5. 20

[460] I. D. Fagadau, L. Mariani, D. Micucci, and O. Riganelli, "Analyzing prompt influence on automated method generation: An empirical study with copilot," *arXiv preprint arXiv:2402.08430*, 2024. 20

[461] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, "On the robustness of code generation techniques: An empirical study on github copilot," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 2149–2160. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00181 20

[462] V. Corso, L. Mariani, D. Micucci, and O. Riganelli, "Generating java methods: An empirical assessment of four ai-based code assistants," *arXiv preprint arXiv:2402.08431*, 2024. 20

[463] K. Koyanagi, D. Wang, K. Noguchi, M. Kondo, A. Serebrenik, Y. Kamei, and N. Ubayashi, "Exploring the effect of multiple natural languages on

code suggestion using github copilot," *arXiv preprint arXiv:2402.01438*, 2024. 20

[464] B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün, "Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt," *arXiv preprint arXiv:2304.10778*, 2023. 21

[465] Y. Fu, P. Liang, A. Tahir, Z. Li, M. Shahin, and J. Yu, "Security weaknesses of copilot generated code in github," *arXiv preprint arXiv:2310.02059*, 2023. 21

[466] C. Qian, X. Cong, C. Yang, W. Chen, Y. Su, J. Xu, Z. Liu, and M. Sun, "Communicative agents for software development," 2023. 21

[467] D. Huang, Q. Bu, J. M. Zhang, M. Luck, and H. Cui, "Agentcoder: Multi-agent-based code generation with iterative testing and optimisation," 2023. 21

[468] K. Zhang, Z. Li, J. Li, G. Li, and Z. Jin, "Self-edit: Fault-aware code editor for code generation," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 769–787. [Online]. Available: https://aclanthology.org/2023.acl-long.45 21

[469] T. X. Olausson, J. P. Inala, C. Wang, J. Gao, and A. Solar-Lezama, "Demystifying gpt self-repair for code generation," 2023. 21

[470] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," 2023. 21

[471] C. Qian, Y. Dang, J. Li, W. Liu, W. Chen, C. Yang, Z. Liu, and M. Sun, "Experiential co-learning of software-developing agents," 2023. 21

[472] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," 2017. 21

[473] C. Si, Y. Zhang, Z. Yang, R. Liu, and D. Yang, "Design2code: How far are we from automating front-end engineering?" 2024. 21

[474] K. Zhang, J. Li, G. Li, X. Shi, and Z. Jin, "Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges," 2024. 21

[475] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=Yacmpz84TH 21

[476] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun, "Toolllm: Facilitating large language models to master 16000+ real-world apis," 2023. 21

[477] B. Lyu, X. Cong, H. Yu, P. Yang, Y. Qin, Y. Ye, Y. Lu, Z. Zhang, Y. Yan, Y. Lin, Z. Liu, and M. Sun, "Gitagent: Facilitating autonomous agent with github by tool extension," 2023. 21

[478] J. Y. Khan and G. Uddin, "Automatic code documentation generation using gpt-3," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3559548 21

[479] Q. Luo, Y. Ye, S. Liang, Z. Zhang, Y. Qin, Y. Lu, Y. Wu, X. Cong, Y. Lin, Y. Zhang, X. Che, Z. Liu, and M. Sun, "Repoagent: An llm-powered open-source framework for repository-level code documentation generation," 2024. 21

[480] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," 2024. 21

[481] A. Frömmgen, J. Austin, P. Choy, N. Ghelani, L. Kharatyan, G. Surita, E. Khrapko, P. Lamblin, P.-A. Manzagol, M. Revaj, M. Tabachnyk, D. Tarlow, K. Villela, D. Zheng, S. Chandra, and P. Maniatis, "Resolving code review comments with machine learning," in *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2024. 21

[482] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 4870–4888. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.438 21

[483] Z. Cheng, T. Xie, P. Shi, C. Li, R. Nadkarni, Y. Hu, C. Xiong, D. Radev, M. Ostendorf, L. Zettlemoyer, N. A. Smith, and T. Yu, "Binding language models in symbolic languages," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=lH1PV42cbF 21

[484] R. Sun, S. O. Arik, R. Sinha, H. Nakhost, H. Dai, P. Yin, and T. Pfister, "Sqlprompt: In-context text-to-sql with minimal labeled data," 2023. 21

[485] R. Sun, S. O. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, and T. Pfister, "Sql-palm: Improved large language model adaptation for text-to-sql," 2023. 21

[486] B. Zhang, Y. Ye, G. Du, X. Hu, Z. Li, S. Yang, C. H. Liu, R. Zhao, Z. Li, and H. Mao, "Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation," 2024. 21

[487] D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, C. Pao, A. Rudnicky, and E. Shriberg, "Expanding the scope of the ATIS task: The ATIS-3 corpus," in *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994. [Online]. Available: https://aclanthology.org/H94-1010 21, 29

[488] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017. 21, 26, 29

[489] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-structured tables," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong and M. Strube, Eds. Beijing, China: Association for Computational

Linguistics, Jul. 2015, pp. 1470–1480. [Online]. Available: https://aclanthology.org/P15-1142 21, 29

[490] J. Li, B. Hui, G. QU, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. Chang, F. Huang, R. Cheng, and Y. Li, "Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: https://openreview.net/forum?id=dI4wzAE6uV 21

[491] J. Yang, A. Prabhakar, K. Narasimhan, and S. Yao, "Intercode: Standardizing and benchmarking interactive coding with execution feedback," 2023. 21

[492] D. Wang, J. Andres, J. D. Weisz, E. Oduor, and C. Dugan, "Autods: Towards human-centered automation of data science," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445526 21

[493] D. Donoho, "50 years of data science," Oct 2018. [Online]. Available: https://qubeshub.org/publications/912/1 21

[494] P. Yin, Z. Lu, H. Li, and K. Ben, "Neural enquirer: Learning to query tables in natural language," in *Proceedings of the Workshop on Human-Computer Question Answering*. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 29–35. [Online]. Available: https://aclanthology.org/W16-0105 21

[495] J. Perkel, "Reactive, reproducible, collaborative: computational notebooks evolve," *Nature*, vol. 593, 2021. 21

[496] P. Yin, W.-D. Li, K. Xiao, A. Rao, Y. Wen, K. Shi, J. Howland, P. Bailey, M. Catasta, H. Michalewski, A. Polozov, and C. Sutton, "Natural language to code generation in interactive data science notebooks," 2022. 21, 22

[497] Y. Wen, P. Yin, K. Shi, H. Michalewski, S. Chaudhuri, and A. Polozov, "Grounding code generation with input-output specifications," in *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*, 2023. [Online]. Available: https://openreview.net/forum?id=wq4OaU8tfE 22

[498] Y. Cao, S. Chen, R. Liu, Z. Wang, and D. Fried, "API-assisted code generation for question answering on varied table structures," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14 536–14 548. [Online]. Available: https://aclanthology.org/2023.emnlp-main.897 22

[499] G. Li, X. Wang, G. Aodeng, S. Zheng, Y. Zhang, C. Ou, S. Wang, and C. H. Liu, "Visualization generation with large language models: An evaluation," 2024. 22

[500] Z. Yang, Z. Zhou, S. Wang, X. Cong, X. Han, Y. Yan, Z. Liu, Z. Tan, P. Liu, D. Yu, Z. Liu, X. Shi, and M. Sun, "Matplotagent: Method and evaluation for llm-based agentic scientific data visualization," 2024. 22

[501] B. Chopra, A. Singha, A. Fariha, S. Gulwani, C. Parnin, A. Tiwari, and A. Z. Henley, "Conversational challenges in ai-powered data science: Obstacles, needs, and design opportunities," 2023. 22

[502] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J.-R. Wen, "A survey on large language model based autonomous agents," 2024. 22

[503] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023. 22

[504] Y. Hu, Q. Xie, V. Jain, J. Francis, J. Patrikar, N. Keetha, S. Kim, Y. Xie, T. Zhang, Z. Zhao *et al.*, "Toward general-purpose robots via foundation models: A survey and meta-analysis," *arXiv preprint arXiv:2312.08782*, 2023. 22

[505] Z. Xi, W. Chen, X. Guo, W. He, Y. Ding, B. Hong, M. Zhang, J. Wang, S. Jin, E. Zhou, R. Zheng, X. Fan, X. Wang, L. Xiong, Y. Zhou, W. Wang, C. Jiang, Y. Zou, X. Liu, Z. Yin, S. Dou, R. Weng, W. Cheng, Q. Zhang, W. Qin, Y. Zheng, X. Qiu, X. Huang, and T. Gui, "The rise and potential of large language model based agents: A survey," 2023. 22

[506] T. Sumers, S. Yao, K. Narasimhan, and T. Griffiths, "Cognitive architectures for language agents," *Transactions on Machine Learning Research*, 2024, survey Certification. [Online]. Available: https://openreview.net/forum?id=1i6ZCvflQJ 22

[507] Z. Durante, Q. Huang, N. Wake, R. Gong, J. S. Park, B. Sarkar, R. Taori, Y. Noda, D. Terzopoulos, Y. Choi *et al.*, "Agent ai: Surveying the horizons of multimodal interaction," *arXiv preprint arXiv:2401.03568*, 2024. 22

[508] K. Yang, J. Liu, J. Wu, C. Yang, Y. R. Fung, S. Li, Z. Huang, X. Cao, X. Wang, Y. Wang, H. Ji, and C. Zhai, "If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents," 2024. 22, 26

[509] C. Ma, J. Zhang, Z. Zhu, C. Yang, Y. Yang, Y. Jin, Z. Lan, L. Kong, and J. He, "Agentboard: An analytical evaluation board of multi-turn llm agents," 2024. 22

[510] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 9493–9500. 22

[511] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023. 22

[512] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530. 22

[513] M. G. Arenas, T. Xiao, S. Singh, V. Jain, A. Z. Ren, Q. Vuong, J. Varley, A. Herzog, I. Leal, S. Kirmani *et al.*, "How to prompt your robot: A promptbook for manipulation skills with code as policies," in *Towards Generalist Robots: Learning Paradigms for Scalable Skill Acquisition@ CoRL2023*, 2023. 22

[514] Y. Jin, D. Li, J. Shi, P. Hao, F. Sun, J. Zhang, B. Fang *et al.*, "Robotgpt: Robot manipulation learning from chatgpt," *arXiv preprint arXiv:2312.01421*, 2023. 22

[515] M. Parakh, A. Fong, A. Simeonov, T. Chen, A. Gupta, and P. Agrawal, "Lifelong robot learning with human assisted language planners," *arXiv e-prints*, pp. arXiv–2309, 2023. 22

[516] S. Huang, Z. Jiang, H. Dong, Y. Qiao, P. Gao, and H. Li, "Instruct2act: Mapping multi-modality instructions to robotic actions with large language model," *arXiv preprint arXiv:2305.11176*, 2023. 22

[517] J. Yang, Y. Dong, S. Liu, B. Li, Z. Wang, C. Jiang, H. Tan, J. Kang, Y. Zhang, K. Zhou *et al.*, "Octopus: Embodied vision-language programmer from environmental feedback," *arXiv preprint arXiv:2310.08588*, 2023. 22

[518] Y. Mu, J. Chen, Q. Zhang, S. Chen, Q. Yu, C. Ge, R. Chen, Z. Liang, M. Hu, C. Tao, P. Sun, H. Yu, C. Yang, W. Shao, W. Wang, J. Dai, Y. Qiao, M. Ding, and P. Luo, "Robocodex: Multimodal code generation for robotic behavior synthesis," 2024. 22

[519] H. Ha, P. Florence, and S. Song, "Scaling up and distilling down: Language-guided robot skill acquisition," in *Conference on Robot Learning*. PMLR, 2023, pp. 3766–3777. 22

[520] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang, "Gensim: Generating robotic simulation tasks via large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=OI3RoHoWAN 22

[521] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," 2023. 22

[522] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023. 22

[523] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2reward: Automated dense reward function generation for reinforcement learning," *arXiv preprint arXiv:2309.11489*, 2023. 22

[524] L. Zheng, R. Wang, X. Wang, and B. An, "Synapse: Trajectory-as-exemplar prompting with memory for computer control," in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. 22

[525] I. Gur, H. Furuta, A. Huang, M. Safdari, Y. Matsuo, D. Eck, and A. Faust, "A real-world webagent with planning, long context understanding, and program synthesis," *arXiv preprint arXiv:2307.12856*, 2023. 22

[526] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better llm agents," 2024. 22

[527] G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun, and T. Scialom, "Gaia: a benchmark for general ai assistants," 2023. 22

[528] Z. Wu, C. Han, Z. Ding, Z. Weng, Z. Liu, S. Yao, T. Yu, and L. Kong, "Os-copilot: Towards generalist computer agents with self-improvement," 2024. 22

[529] Z. Zhao, W. Chai, X. Wang, L. Boyi, S. Hao, S. Cao, T. Ye, J.-N. Hwang, and G. Wang, "See and think: Embodied agent in virtual environment," *arXiv preprint arXiv:2311.15209*, 2023. 22

[530] W. Tan, Z. Ding, W. Zhang, B. Li, B. Zhou, J. Yue, H. Xia, J. Jiang, L. Zheng, X. Xu *et al.*, "Towards general computer control: A multimodal agent for red dead redemption ii as a case study," *arXiv preprint arXiv:2403.03186*, 2024. 22

[531] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," 2023. 22

[532] J. Mao, J. Ye, Y. Qian, M. Pavone, and Y. Wang, "A language agent for autonomous driving," *arXiv preprint arXiv:2311.10813*, 2023. 22

[533] Y. Ma, C. Cui, X. Cao, W. Ye, P. Liu, J. Lu, A. Abdelraouf, R. Gupta, K. Han, A. Bera *et al.*, "Lampilot: An open benchmark dataset for autonomous driving with language model programs," *arXiv preprint arXiv:2312.04372*, 2023. 22

[534] S. Ishida, G. Corrado, G. Fedoseev, H. Yeo, L. Russell, J. Shotton, J. F. Henriques, and A. Hu, "Langprop: A code optimization framework using language models applied to driving," 2024. 22

[535] B. Qiao, L. Li, X. Zhang, S. He, Y. Kang, C. Zhang, F. Yang, H. Dong, J. Zhang, L. Wang, M. Ma, P. Zhao, S. Qin, X. Qin, C. Du, Y. Xu, Q. Lin, S. Rajmohan, and D. Zhang, "Taskweaver: A code-first agent framework," 2023. 22

[536] W. Shi, R. Xu, Y. Zhuang, Y. Yu, J. Zhang, H. Wu, Y. Zhu, J. Ho, C. Yang, and M. D. Wang, "Ehragent: Code empowers large language models for complex tabular reasoning on electronic health records," *arXiv preprint arXiv:2401.07128*, 2024. 22

[537] T. Gupta and A. Kembhavi, "Visual programming: Compositional visual reasoning without training," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 14 953–14 962. 22

[538] L. C. Paulson, *Isabelle: A generic theorem prover*. Springer, 1994. 22

[539] L. de Moura, S. Kong, J. Avigad, F. Van Doorn, and J. von Raumer, "The lean theorem prover (system description)," in *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*. Springer, 2015, pp. 378–388. 22

[540] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *arXiv preprint arXiv:2009.03393*, 2020. 23

[541] A. Q. Jiang, W. Li, J. M. Han, and Y. Wu, "Lisa: Language models of isabelle proofs," in *6th Conference on Artificial Intelligence and Theorem Proving*, 2021, pp. 378–392. 23

[542] S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever, "Formal mathematics statement curriculum learning," *arXiv preprint arXiv:2202.01344*, 2022. 23

[543] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song,

S. Yu, S. Godil, R. Prenger, and A. Anandkumar, "Leandojo: Theorem proving with retrieval-augmented language models," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. [Online]. Available: https://openreview.net/forum?id=g7OX2sOJtn 23

[544] G. Lample, T. Lacroix, M.-A. Lachaux, A. Rodriguez, A. Hayat, T. Lavril, G. Ebner, and X. Martinet, "Hypertree proof search for neural theorem proving," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 337–26 349, 2022. 23

[545] H. Wang, Y. Yuan, Z. Liu, J. Shen, Y. Yin, J. Xiong, E. Xie, H. Shi, Y. Li, L. Li *et al.*, "Dt-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 12 632–12 646. 23

[546] T. Trinh, Y. Wu, Q. Le, H. He, and T. Luong, "Solving olympiad geometry without human demonstrations," *Nature*, 2024. [Online]. Available: https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-system-for-geometry/ 23, 24

[547] T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong, "Solving olympiad geometry without human demonstrations," *Nature*, vol. 625, no. 7995, pp. 476–482, 2024. 23

[548] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Miłoś, Y. Wu, and M. Jamnik, "Thor: Wielding hammers to integrate language models and automated theorem provers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8360–8373, 2022. 23

[549] A. Q. Jiang, S. Welleck, J. P. Zhou, T. Lacroix, J. Liu, W. Li, M. Jamnik, G. Lample, and Y. Wu, "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=SMa9EAovKMC 23

[550] X. Zhao, W. Li, and L. Kong, "Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving," *arXiv preprint arXiv:2305.16366*, 2023. 23

[551] E. First, M. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1229–1241. 23

[552] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022. 23

[553] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi, "Mathematical discoveries from program search with large language models," *Nature*, vol. 625, no. 7995, pp. 468–475, Jan 2024. [Online]. Available: https://doi.org/10.1038/s41586-023-06924-6 23

[554] G. M. Hocky and A. D. White, "Natural language processing models that automate programming will transform chemistry research and teaching," *Digital Discovery*, vol. 1, no. 2, p. 79–83, 2022. [Online]. Available: http://dx.doi.org/10.1039/D1DD00009H 23

[555] G. Landrum *et al.*, "Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling," *Greg Landrum*, vol. 8, p. 31, 2013. 23

[556] J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, pp. 583 – 589, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235959867 23

[557] B. M, D. F, A. I, D. J, O. S, L. Gr, W. J, C. Q, K. Ln, S. Rd, M. C, P. H, A. C, G. Cr, D. A, P. Jh, R. Av, A. van Dijk, E. Ac, O. Dj, S. T, B. C, P.-K. T, R. Mk, D. U, Y. Ck, B. Je, G. Kc, G. Nv, A. Pd, R. Rj, and B. D, "Accurate prediction of protein structures and interactions using a 3-track neural network," *Science (New York, N.Y.)*, vol. 373, pp. 871 – 876, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236141270 23

[558] R. Apweiler, A. Bairoch, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh, "Uniprot: the universal protein knowledgebase," *Nucleic acids research*, vol. 32 Database issue, pp. D115–9, 2004. [Online]. Available: https://api.semanticscholar.org/CorpusID:5163716 23

[559] R. D. Finn, P. C. Coggill, R. Y. Eberhardt, S. R. Eddy, J. Mistry, A. L. Mitchell, S. C. Potter, M. Punta, M. Qureshi, A. Sangrador-Vegas, G. A. Salazar, J. G. Tate, and A. Bateman, "The pfam protein families database: towards a more sustainable future," *Nucleic Acids Research*, vol. 44, pp. D279 – D285, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:9574721 23

[560] S. Kim, P. A. Thiessen, E. E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, J. Wang, B. Yu, J. Zhang, and S. H. Bryant, "Pubchem substance and compound databases," *Nucleic Acids Research*, vol. 44, pp. D1202 – D1213, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:9567253 23

[561] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider, "The swiss-prot protein knowledgebase and its supplement trembl in 2003," *Nucleic acids research*,

vol. 31 1, pp. 365–70, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:12651390 23

[562] Z. Zhu, C. Shi, Z. Zhang, S. Liu, M. Xu, X. Yuan, Y. Zhang, J. Chen, H. Cai, J. Lu, C. Ma, R. Liu, L.-P. Xhonneux, M. Qu, and J. Tang, "Torchdrug: A powerful and flexible machine learning platform for drug discovery," *ArXiv*, vol. abs/2202.08320, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246904686 23

[563] A. L. Dias and T. Rodrigues, "Large language models direct automated chemistry laboratory," 2023. 23

[564] A. M. Bran, S. Cox, A. D. White, and P. Schwaller, "Chemcrow: Augmenting large-language models with chemistry tools," *arXiv preprint arXiv:2304.05376*, 2023. 23

[565] C. Ma, H. Zhao, L. Zheng, J. Xin, Q. Li, L. Wu, Z. Deng, Y. Lu, Q. Liu, and L. Kong, "Retrieved sequence augmentation for protein representation learning," *bioRxiv*, pp. 2023–02, 2023. 23

[566] G. Ye, X. Cai, H. Lai, X. Wang, J. Huang, L. Wang, W. Liu, and X. Zeng, "Drugassist: A large language model for molecule optimization," *arXiv preprint arXiv:2401.10334*, 2023. 23

[567] S. Liu, J. Wang, Y. Yang, C. Wang, L. Liu, H. Guo, and C. Xiao, "ChatGPT-powered conversational drug editing using retrieval and domain feedback," in *1st Workshop on the Synergy of Scientific and Machine Learning Modeling @ ICML2023*, 2023. [Online]. Available: https://openreview.net/forum?id=HhqJtragcp 23

[568] Z. Zheng, O. Zhang, H. L. Nguyen, N. Rampal, A. H. Alawadhi, Z. Rong, T. Head-Gordon, C. Borgs, J. T. Chayes, and O. M. Yaghi, "Chatgpt research group for optimizing the crystallinity of mofs and cofs," *ACS Central Science*, vol. 9, no. 11, pp. 2161–2170, 2023. 23

[569] S. Steiner, J. Wolf, S. Glatzel, A. Andreou, J. M. Granda, G. Keenan, T. Hinkley, G. Aragon-Camarasa, P. J. Kitson, D. Angelone, and L. Cronin, "Organic synthesis in a modular robotic system driven by a chemical programming language," *Science*, vol. 363, no. 6423, p. eaav2211, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aav2211 23

[570] R. Rauschen, M. Guy, J. E. Hein, and L. Cronin, "Universal chemical programming language for robotic synthesis repeatability," *Nature Synthesis*, pp. 1–9, 2024. 23

[571] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. Hashimoto, "Diffusion-LM improves controllable text generation," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=3s9IrEsjLyk 23

[572] S. Gong, M. Li, J. Feng, Z. Wu, and L. Kong, "Diffuseq: Sequence to sequence text generation with diffusion models," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=jQj-_rLVXsj 23

[573] Z. He, T. Sun, Q. Tang, K. Wang, X. Huang, and X. Qiu, "DiffusionBERT: Improving generative masked language models with diffusion models," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 4521–4534. [Online]. Available: https://aclanthology.org/2023.acl-long.248 23

[574] M. Singh, J. Cambronero, S. Gulwani, V. Le, C. Negreanu, and G. Verbruggen, "CodeFusion: A pre-trained diffusion model for code generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 11 697–11 708. [Online]. Available: https://aclanthology.org/2023.emnlp-main.716 23

[575] M. Singh, J. Cambronero, S. Gulwani, V. Le, C. Negreanu, M. Raza, and G. Verbruggen, "Cornet: Learning table formatting rules by example," 2022. 23

[576] N. Chen, Q. Sun, J. Wang, X. Li, and M. Gao, "Pass-tuning: Towards structure-aware parameter-efficient tuning for code representation learning," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 577–591. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.42 23

[577] N. Chirkova and S. Troshin, "CodeBPE: Investigating subtokenization options for large language model pre-training on source code," in *Deep Learning for Code Workshop*, 2022. 23, 24

[578] N. Jain, T. Zhang, W.-L. Chiang, J. E. Gonzalez, K. Sen, and I. Stoica, "Improving code style for accurate code generation," in *NeurIPS 2023 Workshop on Synthetic Data Generation with Generative AI*, 2023. [Online]. Available: https://openreview.net/forum?id=uc9sYHRX6O 24

[579] F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J.-G. Lou, and W. Chen, "RepoCoder: Repository-level code completion through iterative retrieval and generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 2471–2484. [Online]. Available: https://aclanthology.org/2023.emnlp-main.151 24

[580] H. Su, S. Jiang, Y. Lai, H. Wu, B. Shi, C. Liu, Q. Liu, and T. Yu, "Arks: Active retrieval in knowledge soup for code generation," 2024. 24

[581] C. An, S. Gong, M. Zhong, M. Li, J. Zhang, L. Kong, and X. Qiu, "L-eval: Instituting standardized evaluation for long context language models," 2023. 24

[582] L. Fu, H. Chai, S. Luo, K. Du, W. Zhang, L. Fan, J. Lei, R. Rui, J. Lin, Y. Fang, Y. Liu, J. Wang, S. Qi, K. Zhang, W. Zhang, and Y. Yu, "Codeapex: A bilingual programming evaluation benchmark for large language models," 2023. 24, 30

[583] S. Zhou, U. Alon, S. Agarwal, and G. Neubig, "CodeBERTScore: Evaluating code generation with pretrained models of code," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 13 921–

13 937. [Online]. Available: https://aclanthology.org/2023.emnlp-main.859 24

[584] M. Riddell, A. Ni, and A. Cohan, "Quantifying contamination in evaluating code generation capabilities of language models," 2024. 24

[585] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. 24

[586] M. Ali, M. Fromm, K. Thellmann, R. Rutmann, M. Lübbering, J. Leveling, K. Klug, J. Ebert, N. Doll, J. S. Buschhoff, C. Jain, A. A. Weber, L. Jurkschat, H. Abdelwahab, C. John, P. O. Suarez, M. Ostendorff, S. Weinbach, R. Sifa, S. Kesselheim, and N. Flores-Herr, "Tokenizer choice for llm training: Negligible or crucial?" 2023. 24

[587] A. Petrov, E. L. Malfa, P. Torr, and A. Bibi, "Language model tokenizers introduce unfairness between languages," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available: https://openreview.net/forum?id=78yDLKi95p 24

[588] Y. Li, Y. Guo, F. Guerin, and C. Lin, "Evaluating large language models for generalization and robustness via data compression," 2024. 24

[589] G. Dagan, G. Synnaeve, and B. Rozière, "Getting the most out of your tokenizer for pre-training and domain adaptation," 2024. 24

[590] C. Wang, K. Cho, and J. Gu, "Neural machine translation with byte-level subwords," 2019. 24

[591] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, J. Yi, W. Zhao, X. Wang, Z. Liu, H.-T. Zheng, J. Chen, Y. Liu, J. Tang, J. Li, and M. Sun, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, Mar 2023. 24

[592] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4582–4597. 24

[593] X. Liu, K. Ji, Y. Fu, W. Tam, Z. Du, Z. Yang, and J. Tang, "P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 61–68. 24

[594] J. Wang, Q. Sun, N. Chen, C. Wang, J. Huang, M. Gao, and X. Li, "Uncertainty-aware parameter-efficient self-training for semi-supervised language understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7873–7884. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.528 24

[595] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. 24

[596] S. Ayupov and N. Chirkova, "Parameter-efficient fine-tuning of transformers for source code," 2022. 24

[597] W. Zou, Q. Li, J. Ge, C. Li, X. Shen, L. Huang, and B. Luo, "A comprehensive evaluation of parameter-efficient fine-tuning on software engineering tasks," 2023. 24

[598] T. Y. Zhuo, A. Zebaze, N. Suppattarachai, L. von Werra, H. de Vries, Q. Liu, and N. Muennighoff, "Astraios: Parameter-efficient instruction tuning code large language models," *https://arxiv.org/abs/2401.00788*, 2024. 24

[599] W. Wang, Y. Wang, S. Hoi, and S. Joty, "Towards low-resource automatic program repair with meta-learning and pretrained language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 6954–6968. [Online]. Available: https://aclanthology.org/2023.emnlp-main.430 24

[600] C. Wu, Y. Gan, Y. Ge, Z. Lu, J. Wang, Y. Feng, P. Luo, and Y. Shan, "Llama pro: Progressive llama with block expansion," 2024. 24

[601] J. Lin, H. Dong, Y. Xie, and L. Zhang, "Scaling laws behind code understanding model," 2024. 24

[602] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of experts," 2024. 24

[603] M. Xia, T. Gao, Z. Zeng, and D. Chen, "Sheared llama: Accelerating language model pre-training via structured pruning," 2023. 24

[604] H. Chen, A. Saha, S. Hoi, and S. Joty, "Personalized distillation: Empowering open-sourced LLMs with adaptive learning for code generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 6737–6749. [Online]. Available: https://aclanthology.org/2023.emnlp-main.417 24

[605] T. Pires, E. Schlinger, and D. Garrette, "How multilingual is multilingual BERT?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019. [Online]. Available: https://aclanthology.org/P19-1493 25

[606] F. Yuan, S. Yuan, Z. Wu, and L. Li, "How multilingual is multilingual llm?" 2023. 25

[607] T. Y. Zhuo, Z. Yang, Z. Sun, Y. Wang, L. Li, X. Du, Z. Xing, and D. Lo, "Source code data augmentation for deep learning: A survey," 2023. 25

[608] Y. Huang, M. Qi, Y. Yao, M. Wang, B. Gu, C. Clement, and N. Sundaresan, "Program translation via code distillation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10 903–10 914. [Online]. Available: https://aclanthology.org/2023.emnlp-main.672 25

[609] Q. Sun, N. Chen, J. Wang, X. Li, and M. Gao, "Transcoder: Towards unified transferable code representation learning inspired by human skills," 2023. 25

[610] F. Cassano, J. Gouwar, F. Lucchetti, C. Schlesinger, A. Freeman, C. J. Anderson, M. Q. Feldman, M. Greenberg, A. Jangda, and A. Guha, "Knowledge transfer from high-resource to low-resource programming languages for code llms," 2024. 25

[611] A. Mastropaolo, N. Cooper, D. N. Palacio, S. Scalabrino, D. Poshyvanyk, R. Oliveto, and G. Bavota, "Using transfer learning for code-related tasks," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1580–1598, 2023. 25

[612] Z. Sun, X. Du, F. Song, M. Ni, and L. Li, "Coprotector: Protect open-source code against unauthorized training usage with data poisoning," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 652–660. [Online]. Available: https://doi.org/10.1145/3485447.3512225 25

[613] B. Yang, W. Li, L. Xiang, and B. Li, "Srcmarker: Dual-channel source code watermarking via scalable code transformations," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 97–97. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00097 25

[614] A. V. Duarte, X. Zhao, A. L. Oliveira, and L. Li, "De-cop: Detecting copyrighted content in language models training data," 2024. 25

[615] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," 2023. 25

[616] R. Wu, Y. Zhang, Q. Peng, L. Chen, and Z. Zheng, "A survey of deep learning models for structural code understanding," 2022. 26

[617] Y. Xu and Y. Zhu, "A survey on pretrained language models for neural code intelligence," 2022. 26

[618] C. Niu, C. Li, B. Luo, and V. Ng, "Deep learning meets software engineering: A survey on pre-trained models of source code," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, L. D. Raedt, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 5546–5555, survey Track. [Online]. Available: https://doi.org/10.24963/ijcai.2022/775 26

[619] C. Niu, C. Li, V. Ng, D. Chen, J. Ge, and B. Luo, "An empirical comparison of pre-trained models of source code," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 2136–2148. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICSE48619.2023.00180 26

[620] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," 2023. 26

[621] X. She, Y. Liu, Y. Zhao, Y. He, L. Li, C. Tantithamthavorn, Z. Qin, and H. Wang, "Pitfalls in language models for code intelligence: A taxonomy and survey," 2023. 26

[622] L. Ben Allal, N. Muennighoff, L. Kumar Umapathi, B. Lipkin, and L. von Werra, "A framework for the evaluation of code generation models," https://github.com/bigcode-project/bigcode-evaluation-harness, 2022. 26

[623] OpenCompass Contributors, "Opencompass: A universal evaluation platform for foundation models," https://github.com/open-compass/opencompass, 2023. 26

[624] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, L. Benson, L. Sun, E. Zubova, Y. Qiao, M. Burtell, D. Peng, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, S. Joty, A. R. Fabbri, W. Kryscinski, X. V. Lin, C. Xiong, and D. Radev, "Folio: Natural language reasoning with first-order logic," 2022. 26

[625] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_03A-2_Li_paper.pdf 27

[626] R. L. Russell, L. Y. Kim, L. H. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. M. Ellingwood, and M. W. McConley, "Automated vulnerability detection in source code using deep representation learning," in *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018*. IEEE, 2018, pp. 757–762. [Online]. Available: https://doi.org/10.1109/ICMLA.2018.00120 27

[627] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2244–2258, 2021. 27

[628] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "μ vuldeepecker: A deep learning-based system for multiclass vulnerability detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2224–2236, 2019. 27

[629] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Transactions on Software Engineering*, 2021. 27

[630] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "Ac/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 508–512. 27

[631] Y. Zheng, S. Pujar, B. L. Lewis, L. Buratti, E. A. Epstein, B. Yang, J. Laredo, A. Morari, and Z. Su, "D2A: A dataset built for ai-based vulnerability detection methods using differential analysis," in

43rd IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2021, Madrid, Spain, May 25-28, 2021. IEEE, 2021, pp. 111–120. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP52600.2021.00020 27

[632] G. P. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *PROMISE '21: 17th International Conference on Predictive Models and Data Analytics in Software Engineering, Athens Greece, August 19-20, 2021*, S. McIntosh, X. Xia, and S. Amasaki, Eds. ACM, 2021, pp. 30–39. [Online]. Available: https://doi.org/10.1145/3475960.3475985 27

[633] N. Risse and M. Böhme, "Limits of machine learning for automatic vulnerability detection," *arXiv preprint arXiv:2306.17193*, 2023. 27

[634] Z. Gao, H. Wang, Y. Zhou, W. Zhu, and C. Zhang, "How far have we gone in vulnerability detection using large language models," *arXiv preprint arXiv:2311.12420*, 2023. 27

[635] C. L. Goues, N. J. Holtschulte, E. K. Smith, Y. Brun, P. T. Devanbu, S. Forrest, and W. Weimer, "The manybugs and introclass benchmarks for automated repair of C programs," *IEEE Trans. Software Eng.*, vol. 41, no. 12, pp. 1236–1256, 2015. [Online]. Available: https://doi.org/10.1109/TSE.2015.2454513 27

[636] Q. Hanam, F. S. D. M. Brito, and A. Mesbah, "Discovering bug patterns in c," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, T. Zimmermann, J. Cleland-Huang, and Z. Su, Eds. ACM, 2016, pp. 144–156. [Online]. Available: https://doi.org/10.1145/2950290.2950308 27

[637] S. H. Tan, J. Yi, Yulis, S. Mechtaev, and A. Roychoudhury, "Codeflaws: a programming competition benchmark for evaluating automated program repair tools," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume*, S. Uchitel, A. Orso, and M. P. Robillard, Eds. IEEE Computer Society, 2017, pp. 180–182. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.76 27

[638] J. A. Prenner, H. Babii, and R. Robbes, "Can openai's codex fix bugs? an evaluation on quixbugs," in *Proceedings of the Third International Workshop on Automated Program Repair*, 2022, pp. 69–75. 27

[639] R. K. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. R. Prasad, "Bugs.jar: a large-scale, diverse dataset of real-world java bugs," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, A. Zaidman, Y. Kamei, and E. Hill, Eds. ACM, 2018, pp. 10–13. [Online]. Available: https://doi.org/10.1145/3196398.3196473 27

[640] F. Madeiral, S. Urli, M. de Almeida Maia, and M. Monperrus, "BEARS: an extensible java bug benchmark for automatic program repair studies," in *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, X. Wang, D. Lo, and E. Shihab, Eds. IEEE, 2019, pp. 468–478. [Online]. Available: https://doi.org/10.1109/SANER.2019.8667991 27

[641] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, Á. Beszédes, R. Ferenc, and A. Mesbah, "Bugsjs: a benchmark of javascript bugs," in *12th IEEE Conference on Software Testing, Validation and Verification, ICST 2019, Xi'an, China, April 22-27, 2019*. IEEE, 2019, pp. 90–101. [Online]. Available: https://doi.org/10.1109/ICST.2019.00019 27

[642] D. A. Tomassi, N. Dmeiri, Y. Wang, A. Bhowmick, Y. Liu, P. T. Devanbu, B. Vasilescu, and C. Rubio-González, "Bugswarm: mining and continuously growing a dataset of reproducible failures and fixes," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 339–349. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00048 27

[643] R.-M. Karampatsis and C. Sutton, "How often do single-statement bugs occur? the manysstubs4j dataset," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 573–577. 27

[644] Y. Hu, U. Z. Ahmed, S. Mechtaev, B. Leong, and A. Roychoudhury, "Re-factoring based program repair applied to programming assignments," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 388–398. 27

[645] F. Huq, M. Hasan, M. M. A. Haque, S. Mahbub, A. Iqbal, and T. Ahmed, "Review4repair: Code review aided automatic program repairing," *Inf. Softw. Technol.*, vol. 143, p. 106765, 2022. [Online]. Available: https://doi.org/10.1016/j.infsof.2021.106765 27

[646] R. Widyasari, S. Q. Sim, C. Lok, H. Qi, J. Phan, Q. Tay, C. Tan, F. Wee, J. E. Tan, Y. Yieh, B. Goh, F. Thung, H. J. Kang, T. Hoang, D. Lo, and E. L. Ouh, "Bugsinpy: a database of existing bugs in python programs to enable controlled testing and debugging studies," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 1556–1560. [Online]. Available: https://doi.org/10.1145/3368089.3417943 27

[647] B. Berabi, J. He, V. Raychev, and M. Vechev, "Tfix: Learning to fix coding errors with a text-to-text transformer," in *International Conference on Machine Learning*. PMLR, 2021, pp. 780–791. 27

[648] M. Monperrus, M. Martinez, H. Ye, F. Madeiral, T. Durieux, and Z. Yu, "Megadiff: A dataset of 600k java source code changes categorized by diff size," *arXiv preprint arXiv:2108.04631*, 2021. 27

[649] C. Richter and H. Wehrheim, "TSSB-3M: mining single statement bugs at massive scale," in *19th IEEE/ACM International Conference on Mining Software*

Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022. ACM, 2022, pp. 418–422. [Online]. Available: https://doi.org/10.1145/3524842.3528505 27

[650] V. Csuvik and L. Vidács, "Fixjs: a dataset of bug-fixing javascript commits," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 712–716. 27

[651] W. Oh and H. Oh, "Pyter: effective program repair for python type errors," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 922–934. 27

[652] M. A. M. Khan, M. S. Bari, X. L. Do, W. Wang, M. R. Parvez, and S. Joty, "xcodeeval: A large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval," 2023. 27, 28, 30

[653] J. A. Prenner and R. Robbes, "Runbugrun–an executable dataset for automated program repair," *arXiv preprint arXiv:2304.01102*, 2023. 27

[654] W. U. Ahmad, M. G. R. Tushar, S. Chakraborty, and K. Chang, "AVATAR: A parallel corpus for java-python program translation," in *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, A. Rogers, J. L. Boyd-Graber, and N. Okazaki, Eds. Association for Computational Linguistics, 2023, pp. 2268–2281. [Online]. Available: https://doi.org/10.18653/v1/2023.findings-acl.143 28

[655] M. Zhu, A. Jain, K. Suresh, R. Ravindran, S. Tipirneni, and C. K. Reddy, "Xlcost: A benchmark dataset for cross-lingual code intelligence," *CoRR*, vol. abs/2206.08474, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2206.08474 28

[656] M. Jiao, T. Yu, X. Li, G. Qiu, X. Gu, and B. Shen, "On the evaluation of neural code translation: Taxonomy and benchmark," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023.* IEEE, 2023, pp. 1529–1541. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00114 28

[657] Z. Yao, D. S. Weld, W. Chen, and H. Sun, "Staqc: A systematically mined question-code dataset from stack overflow," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, P. Champin, F. Gandon, M. Lalmas, and P. G. Ipeirotis, Eds. ACM, 2018, pp. 1693–1703. [Online]. Available: https://doi.org/10.1145/3178876.3186081 28

[658] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig, "Learning to mine aligned code and natural language pairs from stack overflow," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 476–486. [Online]. Available: https://doi.org/10.1145/3196398.3196408 28

[659] S. Yan, H. Yu, Y. Chen, B. Shen, and L. Jiang, "Are the code snippets what we are searching for? A benchmark and an empirical study on code search with natural-language queries," in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M. Fokaefs, and M. Zhou, Eds. IEEE, 2020, pp. 344–354. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054840 28

[660] G. Heyman and T. V. Cutsem, "Neural code search revisited: Enhancing code snippet retrieval through natural language intent," *CoRR*, vol. abs/2008.12193, 2020. [Online]. Available: https://arxiv.org/abs/2008.12193 28

[661] X. Ling, L. Wu, S. Wang, G. Pan, T. Ma, F. Xu, A. X. Liu, C. Wu, and S. Ji, "Deep graph matching and searching for semantic code retrieval," *ACM Trans. Knowl. Discov. Data*, vol. 15, no. 5, pp. 88:1–88:21, 2021. [Online]. Available: https://doi.org/10.1145/3447571 28

[662] J. Huang, D. Tang, L. Shou, M. Gong, K. Xu, D. Jiang, M. Zhou, and N. Duan, "Cosqa: 20, 000+ web queries for code search and question answering," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, 2021, pp. 5690–5700. [Online]. Available: https://doi.org/10.18653/v1/2021.acl-long.442 28

[663] L. A. Agrawal, A. Kanade, N. Goyal, S. K. Lahiri, and S. K. Rajamani, "Guiding language models of code with global context using monitors," *CoRR*, vol. abs/2306.10763, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.10763 28

[664] T. Liu, C. Xu, and J. McAuley, "Repobench: Benchmarking repository-level code auto-completion systems," 2023. 28

[665] Q. Liu, Z. Liu, H. Zhu, H. Fan, B. Du, and Y. Qian, "Generating commit messages from diffs using pointer-generator network," in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, M. D. Storey, B. Adams, and S. Haiduc, Eds. IEEE / ACM, 2019, pp. 299–309. [Online]. Available: https://doi.org/10.1109/MSR.2019.00056 28

[666] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "ATOM: commit message generation based on abstract syntax tree and hybrid ranking," *IEEE Trans. Software Eng.*, vol. 48, no. 5, pp. 1800–1817, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2020.3038681 28

[667] H. Wang, X. Xia, D. Lo, Q. He, X. Wang, and J. Grundy, "Context-aware retrieval-based deep commit message generation," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 4, pp. 56:1–56:30, 2021. [Online]. Available: https://doi.org/10.1145/3464689 28

[668] L. Wang, X. Tang, Y. He, C. Ren, S. Shi, C. Yan, and Z. Li, "Delving into commit-issue correlation to enhance commit message generation models," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023.* IEEE, 2023, pp. 710–722. [Online]. Available:

https://doi.org/10.1109/ASE56229.2023.00050 28

[669] A. Eliseeva, Y. Sokolov, E. Bogomolov, Y. Golubev, D. Dig, and T. Bryksin, "From commit message generation to history-aware commit message completion," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 723–735. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00078 28

[670] M. Schall, T. Czinczoll, and G. de Melo, "Commitbench: A benchmark for commit message generation," 2024. 28

[671] B. Li, W. Wu, Z. Tang, L. Shi, J. Yang, J. Li, S. Yao, C. Qian, B. Hui, Q. Zhang, Z. Yu, H. Du, P. Yang, D. Lin, C. Peng, and K. Chen, "Devbench: A comprehensive benchmark for software development," 2024. 28

[672] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th conference on program comprehension*, 2018, pp. 200–210. 29

[673] Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, "A critical review of large language model on software engineering: An example from chatgpt and automated program repair," *arXiv preprint arXiv:2310.08879*, 2023. 29

[674] R. Tian, Y. Ye, Y. Qin, X. Cong, Y. Lin, Y. Pan, Y. Wu, Z. Liu, and M. Sun, "Debugbench: Evaluating debugging capability of large language models," 2024. 29

[675] S. P. Sahu, M. Mandal, S. Bharadwaj, A. Kanade, P. Maniatis, and S. Shevade, "Codequeries: A dataset of semantic queries over code," in *Proceedings of the 17th Innovations in Software Engineering Conference*, ser. ISEC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3641399.3641408 29

[676] L. Li, S. Geng, Z. Li, Y. He, H. Yu, Z. Hua, G. Ning, S. Wang, T. Xie, and H. Yang, "Inficoder-eval: Systematically evaluating the question-answering capabilities of code large language models," 2024. 29

[677] Q. Hu, K. Li, X. Zhao, Y. Xie, T. Liu, H. Chen, Q. Xie, and J. He, "Instructcoder: Empowering language models for code editing," 2023. 29

[678] T. Shi, C. Zhao, J. L. Boyd-Graber, H. D. III, and L. Lee, "On the potential of lexico-logical alignments for semantic parsing to SQL queries," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1849–1864. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.167 29

[679] C. Lee, O. Polozov, and M. Richardson, "Kaggledbqa: Realistic evaluation of text-to-sql parsers," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, 2021, pp. 2261–2273. [Online]. Available: https://doi.org/10.18653/v1/2021.acl-long.176 29

[680] Y. Hao, G. Li, Y. Liu, X. Miao, H. Zong, S. Jiang, Y. Liu, and H. Wei, "Aixbench: A code generation benchmark dataset," *CoRR*, vol. abs/2206.13179, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2206.13179 30

[681] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, T. Xie, and Q. Wang, "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," *CoRR*, vol. abs/2302.00288, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.00288 30

[682] X. Du, M. Liu, K. Wang, H. Wang, J. Liu, Y. Chen, J. Feng, C. Sha, X. Peng, and Y. Lou, "Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation," 2023. 30

[683] S. Wang, L. Ding, L. Shen, Y. Luo, B. Du, and D. Tao, "Oop: Object-oriented programming evaluation benchmark for large language models," 2024. 30

[684] R. Li, J. Fu, B.-W. Zhang, T. Huang, Z. Sun, C. Lyu, G. Liu, Z. Jin, and G. Li, "Taco: Topics in algorithmic code generation dataset," *arXiv preprint arXiv:2312.14852*, 2023. 30

[685] Z. Wang, G. Cuenca, S. Zhou, F. F. Xu, and G. Neubig, "Mconala: A benchmark for code generation from multiple natural languages," in *Findings of the Association for Computational Linguistics: EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, A. Vlachos and I. Augenstein, Eds. Association for Computational Linguistics, 2023, pp. 265–273. [Online]. Available: https://doi.org/10.18653/v1/2023.findings-eacl.20 30

[686] X. Chen, L. Gong, A. Cheung, and D. Song, "Plotcoder: Hierarchical decoding for synthesizing visualization code in programmatic context," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021. 30

[687] J. Huang, C. Wang, J. Zhang, C. Yan, H. Cui, J. P. Inala, C. Clement, and N. Duan, "Execution-based evaluation for data science code generation models," in *Proceedings of the Fourth Workshop on Data Science with Human-in-the-Loop (Language Advances)*. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, Dec. 2022, pp. 28–36. [Online]. Available: https://aclanthology.org/2022.dash-1.5 30

[688] X. Hu, Z. Zhao, S. Wei, Z. Chai, Q. Ma, G. Wang, X. Wang, J. Su, J. Xu, M. Zhu, Y. Cheng, J. Yuan, J. Li, K. Kuang, Y. Yang, H. Yang, and F. Wu, "Infiagent-dabench: Evaluating agents on data analysis tasks," 2024. 30

[689] H. M. Babe, S. Nguyen, Y. Zi, A. Guha, M. Q. Feldman, and C. J. Anderson, "Studenteval: A benchmark of student-written prompts for large language models of code," *CoRR*, vol. abs/2306.04556, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.04556 30

[690] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the*

41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019. IEEE / ACM, 2019, pp. 121–130. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00021 30

[691] Z. Jiang, J. Liu, J. Huang, Y. Li, Y. Huo, J. Gu, Z. Chen, J. Zhu, and M. R. Lyu, "A large-scale benchmark for log parsing," *CoRR*, vol. abs/2308.10828, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.10828 30

[692] M. Liu, N. R. Pinckney, B. Khailany, and H. Ren, "Invited paper: Verilogeval: Evaluating large language models for verilog code generation," in *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*. IEEE, 2023, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICCAD57390.2023.10323812 30