Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about

Online ISSN 3007-3197
Print ISSN 3007-3189

https://doi.org/10.5281/zenodo.18091368

Scaling Down Code: Balancing Data Volume and Intelligence in Code Generation Models

Zainab Ishtiaq
Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan Email: zaenbishtiaq53@gmail.com

Muhammad Waseem
Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan Email: m.wasi17@gmail.com

Muhammad Kamran
Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan Email: muhammadkamran5862@gmail.com

Muhammad Usman Ghani Khan
Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan Email: usman.ghani@uet.edu.pk

ABSTRACT

The performance of models that perform code generation primarily depends on both the number of data points they are trained on and how well those data points represent functionally correct input code. This research looks at the interactions between various sizes of CodeSearchNet-Python datasets and transformer-based models that have been trained using fine-tuning methods using cross-entropy loss functions. The study also looks at the method used to tokenize the training set and the characteristics and performance of the models' decoders, and whether or not these models have the same type of performance for varying numbers of data extracted from CodeSearchNet-Python. All evaluations used HumanEval data as the performance criterion and reported performance using metrics such as ROUGE and BLEU which measure both functionally-correct and coherent outputs. The results show that as the number of records grows in increments across different dataset sizes, the performance levels of the models fluctuate as a result of the growing volume of usable data; thus producing critical insights into opportunities and risks associated with training data size for optimally efficient trainable resources. Additionally, they show evidence that the effort expended on ensuring that there is a sufficient degree of quality in the preparation (distillation) of training data has a significant impact on addressing the issue of scale.

Keywords: Code Generation, Transformer Models, Dataset Scaling, CodeSearchNet, HumanEval, CodeGen, StarCoder, CodeT5, Model Performance, Cross-Entropy Loss, Tokenization, BLEU Score, ROUGE Score, Fine- Tuning, Data Efficiency, SFT

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

Introduction

As foundation models become more central to modern machine learning workflows [1], their use in code generation is transforming how developers work with software [2]. Similarly, the fast- paced development of Large Language Models (LLMs) [3], including OpenAI's GPT, Google's Gemini, Meta's LLaMA, and High-Flyer's Deepseek, has transformed the space of automated code generation [4] [5] [6] [7] [8]. In addition, tools like GitHub Copilot [9] have demonstrated the transformative potential of AI-powered code assistants in both industry and education. These models, which have been trained on enormous collections of publicly accessible code (e.g.,GitHub, Stack Overflow), show incredible ability to comprehend programming syntax, semantics, and even top-level software design patterns.

Moreover, these code generative models help developers write, debug, and optimize code more efficiently and conveniently than before. Recent LLM-based systems produce context-aware, syntactically correct, and often functionally correct code snippets from natural language input based on deep learning algorithms, saving time and reducing the efforts involved in writing code. Behind these tools lie complex trade-offs between model architecture, training data volume, and performance. Subsequently, these Code Generative LLM models are trained on billions of datasets from various sources [10]. Still, the question is, do we need this large amount of data [11], or does there come a point when increasing the amount of data does not affect the performance of code generative models? Performance has also been observed to increase with model size [12], but what is the actual pattern of increase in performance? While state-of-the-art models like GPT-4 and CodeLLaMA require massive computational resources, there is a growing interest in smaller, more efficient models that can still deliver high-quality code synthesis when properly fine-tuned [13] [14].

All these questions frequently arise in the minds of researchers and machine learning experts. These questions are often overlooked, even though a lot of work has been done on these large language generative models [15]. Many people believe that the quantity of data affects model performance to a large extent. Mainly, there is this thought that the larger the dataset used to train the model, the better the model will perform [16]. The primary objective of this study is to address the questions outlined above in the most comprehensive manner possible. This research primarily focuses on determining how the dataset impacts the performance of different code generative models.

The main contributions of this paper are provided below:

- Provides a thorough comparative analysis of the CodeGen, CodeT5, and TinyStarCoder models using different portions of the CodeSearchNet dataset and demonstrates how performance is influenced by data quantity.
- Discovered that model architecture made a significant impact on data sensitivity, specifically that the encoder-decoder models (i.e., CodeT5), even with fewer parameters, were more robust to data reduction than decoder-only models (i.e., TinyStarCoder).

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

LITERATURE REVIEW

N. Jain et al. [17] investigated data quality for code and found that making the code more structured leads to improved code generation performance by building a novel data-cleaning pipeline. However, reducing the number of unique solutions per task may also limit how well these models generalize to different coding problems. Furthermore, N. Christakis et al. [18] introduced the INFINITE methodology, a novel inference index designed to systematically evaluate the performance of LLMs in code generation. But the approach is tested on a small range of coding tasks and models, so its effectiveness in broader scenarios remains uncertain.

Additionally, Y. Han et al. [19] propose a multistage guided code generation strategy that aims to gradually shorten the transformation distance between the problem description and the correct code. However, the reliance on synthetic benchmarks, used in this study, may limit real-world generalizability. Moreover, V. Iyer et al. [20] explored what it would take to adapt LLMs to the low-resource setting. It shows that parallel data is critical during both pre-training and Supervised Fine-tuning (SFT), and diversity tends to cause interference instead of transfer. Its reliance on synthetic benchmarks may limit the applicability of findings to real-world translation scenarios. In addition, J. Jiang et al. [10] provided a comprehensive survey of LLMs tailored for code generation, addressing a notable gap in the existing literature. However, depending heavily on current benchmarks may restrict its relevance to real-world use cases and new challenges in code generation. S. Zhou et al. [21] have also created CodeBERTScore as a way to evaluate how well a trained model generates code. However, an important limitation of CodeBERTScore is that it may fail to identify some syntactical errors and all compile-time/failure errors, as shown in Table 1.

C. Song et al. [22], in their study, which focused on using a new dataset called DoIT, looked at the effect of different factors on instruction tuning of Chinese language LLMs. It is important to note that the results may only be applicable to Chinese LLM models so they cannot be considered generalisable to other language LLMs. In a separate study, Nijkamp et al. [23] created CodeGen, which consists of a large number of models (16.1B parameters maximum) whose purpose is to generate code from natural languages. Furthermore, the MTPB (Multi-Turn Programming Benchmark) was established by Nijkamp et al. [23] to track the performance of these models for multi-step interactive code generation. The models that were created are mainly evaluated based on benchmarks created from simulated environments and may not equate to real-world coding complexity levels.

In another study, M. Evtikhiev et al. [24] examined the efficacy of various evaluation metrics that are used to assess the performance of code generation models. Evtikhiev's work particularly points to the weaknesses of BLEU when evaluating code generation models for their semantic correctness. However, the Evtikhiev et al. research is restricted to a limited number of programming languages and development tasks, resulting in reduced generalisability of this study. J. Austin et al. [12] assessed different transformer-based models for generating Python code from natural language queries. This research also produced two different types of benchmarks: one named the MBPP, with 974 basic programming problems and the MathQA- Python benchmark, which has 23,914 math-related problem sets. It has been demonstrated that semantic knowledge remains a challenge for instructions and many modern code

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

generation models, as these models often require multiple attempts before successfully predicting a program output.

METHODOLOGY

Dataset and Data Preprocessing

We trained our model on the CodeSearchNet [25], a high-quality, large-scale dataset of code, aggregated from open-source repositories, collaboratively released by GitHub and Weights & Biases.

TABLE I: Comprehensive Comparative analysis of code generation Large Language Models as reported in recent Literature.

No.	Year	Paper	Model	Dataset	Results	Strengths	Limitations
1	2025	Jain et al. [17]	CodeLlama-7B	MBPP HumanEv al APPS	• 30% improvement • 15% cleaned & 100% original • Beats AlphaCode	• Cleaning pipeline • Data quality focus Small-scale efficiency	• Limited evaluation • LLM-dependence
2	2025	Christakis et al. [18]	• GPT-4o (GPT) • OpenAI-o1 pro (OAI1) • OpenAI-o3 mini- high	CSV numeric columns Meteorological data Variables: - Temperature - Humidity - Wind Speed Time Series	• Simple Tasks Scores: - GPT: 1.0 - OAI3: 1.0 - OAI1: 0.83 • Complex Tasks InI: - GPT: 0.76 - OAI3: 0.75 - OAI1: 0.62	• INFINITE method • Controlled eval • Supervised validation	• OpenAI-only • Less automation
3	2025	Han et al. [19]	• ChatGPT • text-davinci-003	CodeC (208 0 problems) APPS, MBPP HumanEval(X)	• 34.7% Pass@1 im-provement • Multi-language suc-cess	• 3-stage method • New CodeC dataset • Problem-code gap reduction	• Cascading error risk Needs powerful LLMs
4	2025	Iyer et al. [20]	• Mistral • LLaMA3 (0.7B each)	Indigenous American and NE Indian lan- guages	• Parallel data helps • Diverse SFT data can hurt • Opposite of	• Low-resource focus • Data scale/diversit	• Language-specific • May not generalize

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

					high- resource	y study	
5	2024	Jiang et al. [10]	• Codex • CodeGen • Other code LLMs	HumanEval MBPP BigCodeBench	• Performance improve- ment across benchmarks	• Comprehensive tax- onomy • Historical overview • Ethics coverage	• High-level only Lacks technical depth
6	2023	Zhou et al. [21]	CodeBERT	• CoNaLa • CodeXGLUE	CodeBERTScore outperforms BLEU and CodeBLEU in correlating with human judgment.	Captures semantic similarity aligned with functional correctness	May not detect syntac- tic or compilation er- rors in generated code
7	2023	Song et al. [22]	• Chinese-LLaMA • Baichuan2 • Qwen1. 5 (7B-33B)	DoIT (40k instances)	• Ability growth varies Human data >syn- thetic • Good generalization	• Ability-specific anal- ysis • DoIT dataset contri- bution • Multilingual focus	• Chinese-only • May not generalize
8	2023	Nijkamp et al. [23]	CodeGen (16B)	HumanEval Multi-Turn Benchmark	• Strong zero-shot • Multi-turn is better than single	• Open access • Multi-Turn Bench- mark • Scalability	• Real-world general- ization? • Benchmark limita- tions
9	2023	Evtikhiev et al. [24]	• CoNaLa baseline • Codex (DaVinci) • TranX variants • NL2Code, GCNN	CoNaLa Hearthston e	• ChrF/ROUGE-L best • BLEU/CodeBL EU misleading • Codex excluded	• Human evaluation Statistical resampling	• Only 2 datasets • No snippet analysis
10	2021	Austin et al. [12]	LLMs (244 M- 137B params)	MBPP MathQA-Python	• Few-Shot: 59.6% on MBPP • Fine-Tuning: +10% MathQA:	• Scalability • Human interaction • Versatility	• Semantic challenges • Program complexity issues

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

					83.8% accuracy Human feedback halved errors		

The dataset comprises millions of function-level code snippets across multiple programming languages, each accompanied by natural language documentation strings or function signatures. The data is grouped by language, and for each language, it contains cleaned and parsed code examples with metadata such as repository name, file path, license type, function name, and more. It has structured fields in each entry with the code, docstring (natural language description of the code), programming language, function name, original string (code together with the docstring), commit hash (SHA), repository URL, and an ID.
CodeSearchNet supports six popular programming languages, namely Python, Java, JavaScript, Go, Ruby, and PHP. The data is split into train, valid, and test splits in JSONL format, with processing done at the function level. Overall, it contains about 1.3 million code/doc pairs, with more than 457k of those being examples written in Python. This dataset preserves high-quality correspondence between code and documentation pairs, hence being highly applicable for the task of code generation [26] [27] [28].
For our experiment, we utilized the Python subset specifically to fine-tune the model. To get the CodeSearchNet dataset ready for fine-tuning, we used a preprocessing function designed specifically for code generation tasks. This function converts each data point into a prompt- response format, suitable for autoregressive models, where a natural language instruction is followed by its corresponding code. The idea was to make the inputs more consistent, keeping the sequence lengths manageable, and structuring the data in a way that helps the model learn more effectively during supervised training.
For model testing, we used the OpenAI HumanEval dataset, a widely used benchmark to measure the functional correctness of AI model-generated code [29]. HumanEval is a set of 164 Python programming exercises, with each containing an action signature, a docstring describing the function, and a set of concealed unit tests for verification. In contrast to token-level measures, HumanEval distinguishes between functional correctness and emphasizes a consistent measure of the model's capability to produce syntactically accurate and semantically sound code [2]. In our fine-tuning process, the HumanEval dataset helped us see how well our model could turn natural language prompts into working code. For each example, the model was given a description of a coding task and asked to generate a solution. It rendered this dataset an easy but solid benchmark for measuring code generation performance.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

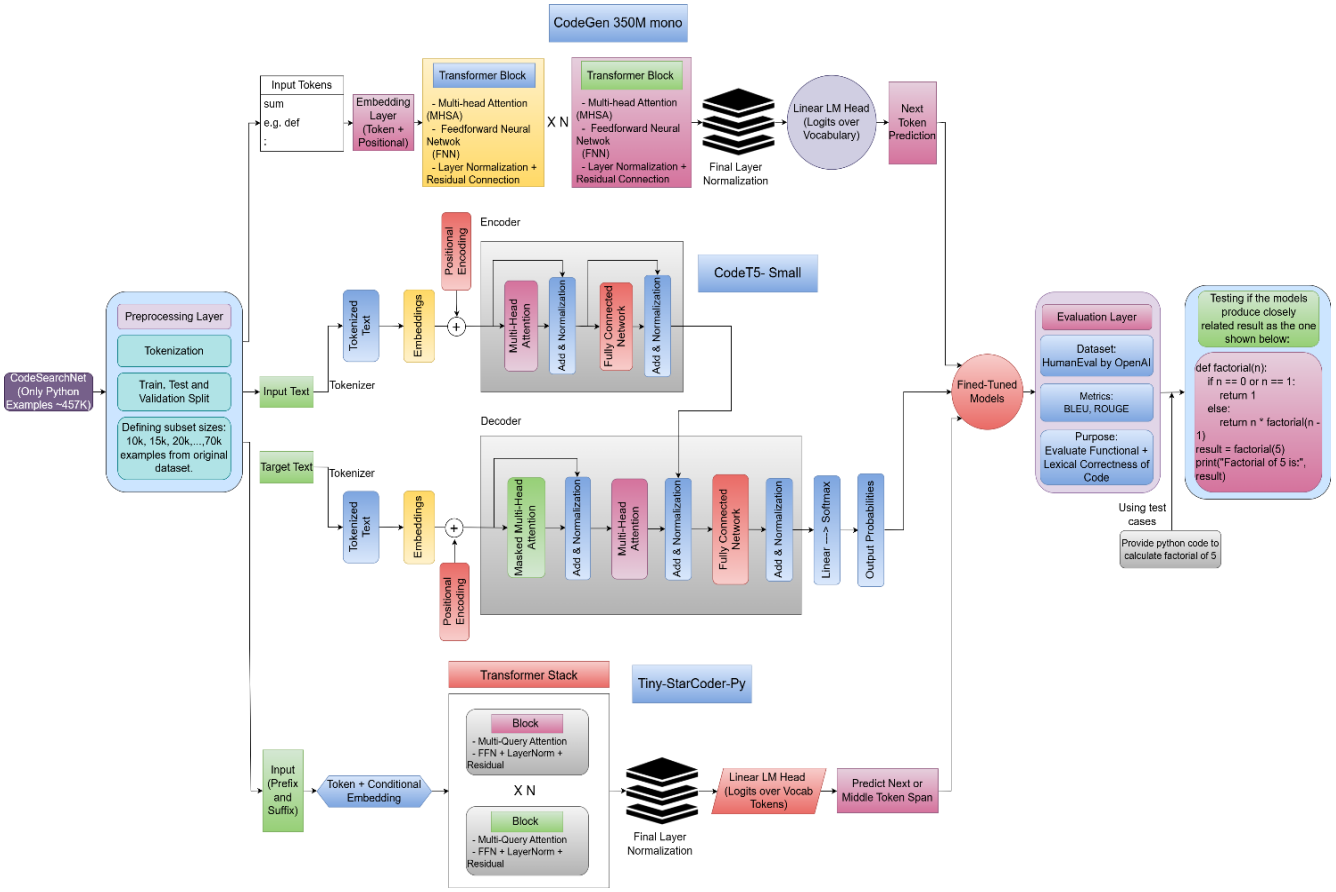


Figure 1: Architecture diagram of the fine-tuning pipeline for code-generation models (CodeGen, TinyStarCoder, CodeT5) on subsets of the CodeSearchNet-Python dataset. The workflow includes preprocessing (tokenization, dataset splitting), model-specific components (cross-entropy loss, decoder layers), and evaluation using BLEU/ROUGE metrics. Subsets (10k, 15k, 20k,..., 70k) of the original dataset are used to analyze the impact of dataset quantity on performance.

Experimental Setup

After preprocessing the CodeSearchNet dataset, the size of the subset of the dataset was determined to fine-tune the model (ranging from 10k to 70k in our experiment). Both the input and target sequences were tokenized to a fixed maximum length of 256 tokens, with padding applied. During tokenization of target labels, all padding token IDs were replaced with -100, so they would be ignored in the loss calculation. After this, the models are imported for fine-tuning.

We utilized three models in our experiments, as shown in the architecture diagram Fig. 1. One such model used was CodeT5-small, a light-weight encoder-decoder model from Salesforce Research that belongs to the CodeT5 family [30]. CodeT5-small is based on the T5 architecture and is specifically designed for a range of software engineering tasks, including code summarization, generation, translation, and refinement. With about 60 million parameters, it is a computationally light variant of its larger versions: CodeT5-base and CodeT5-large. CodeT5- small is particularly good at sequence-to-sequence (Seq2Seq) learning tasks. It can generate

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

executable code from natural language descriptions and natural language summaries or cross- language translations of code snippets [31]. The model utilizes a SentencePiece tokenizer, as in the original T5 architecture.The second model employed in our experiment was CodeGen-350M-mono, a small autoregressive language model designed by Salesforce and tailored to Python code generation tasks specifically [32]. Being the smallest of the CodeGen family, the model contains roughly 350 million parameters. Architecturally, CodeGen-350M-mono utilizes a transformer-based architecture and is trained with cross-entropy loss to capture the probability of the next token with the context before it [23].

The last model used in the experiment was TinyStarCoderPy, which is a Python-specialized language model of the BigCode project [33]. With approximately 159 million parameters, it is the most compact member in the StarCoder family, designed specifically to offer quick and efficient code generation features in environment-limited settings [34]. Even with its lower number of parameters, TinyStarCoderPy maintains major architectural advances from its larger siblings, such as Multi-Query Attention (MQA) [35] and Fill-in-the-Middle (FIM) [36] code completion support, making it capable of generating code in random gaps of a given codebase, a crucial capability for most contemporary development processes.

The same environment was provided to each model during each subset of data to evaluate them in an equally effective environment. Training was performed over 3 epochs with a batch size of 8, using a learning rate of 2e-5 and a weight decay of 0.01. After the fine-tuning process is done, the model is evaluated using the code snippets from the HumanEval Dataset using evaluation metrics like BLEU (evaluates how closely the model's generated code matches the expected answer by comparing overlapping sequences of tokens) and ROUGE (measures how much of the reference is successfully recalled in the model's output). Then, the BLEU and ROUGE scores are noted down.

Algorithm 1 depicts the complete experimental setup.

Algorithm 1 Code Generation Model Fine-Tuning and Evaluation

Input:	CodeSearchNet (Python subset), model checkpoints {CodeGen, CodeT5, TinyStarCoder}, dataset splits {10k, 15k, 20k,..., 70k}
Output:	Fine-tuned models with BLEU and ROUGE evaluation scores
1:	**Preprocessing and Tokenization**
2:	**for** each sample s in dataset **do**
3:	Extract documentation d and code c
4:	x ←tokenizer("Generate Python code: "+d)

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

5: y ←tokenizer(c)

6: Replace padding token IDs in y with −100

7: **end for**

8: **Model Fine-Tuning**

9: **for** each model M ∈ {CodeGen, CodeT5, TinyStarCoder} **do**

10: Load pre-trained checkpoint

11: Set training parameters: learning rate, batch size, epochs

12: Train M using the processed dataset

13: **end for**

14: **Evaluation on HumanEval**

15: **for** each model M **do**

16: for each prompt p in HumanEval test set **do**

17: Generate prediction ˆc ← M(p)

18: Compare ˆc with ground truth using BLEU, ROUGE

19: **end for**

20: Compute average metrics for M

21: **end for**

RESULTS

The fine-tuned performance of each model was evaluated using both the BLEU and ROUGE scoring systems on different sized datasets, along with noting how much time was required for each model to fine-tune. These evaluations provided insights on how each fine-tuned model is affected by the degree of scaling and what the best measurements were for each model. The graphs in Fig. 2 provide visualisation of the differences in BLEU and ROUGE scores for each of the models relative to the size of the respective datasets used.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

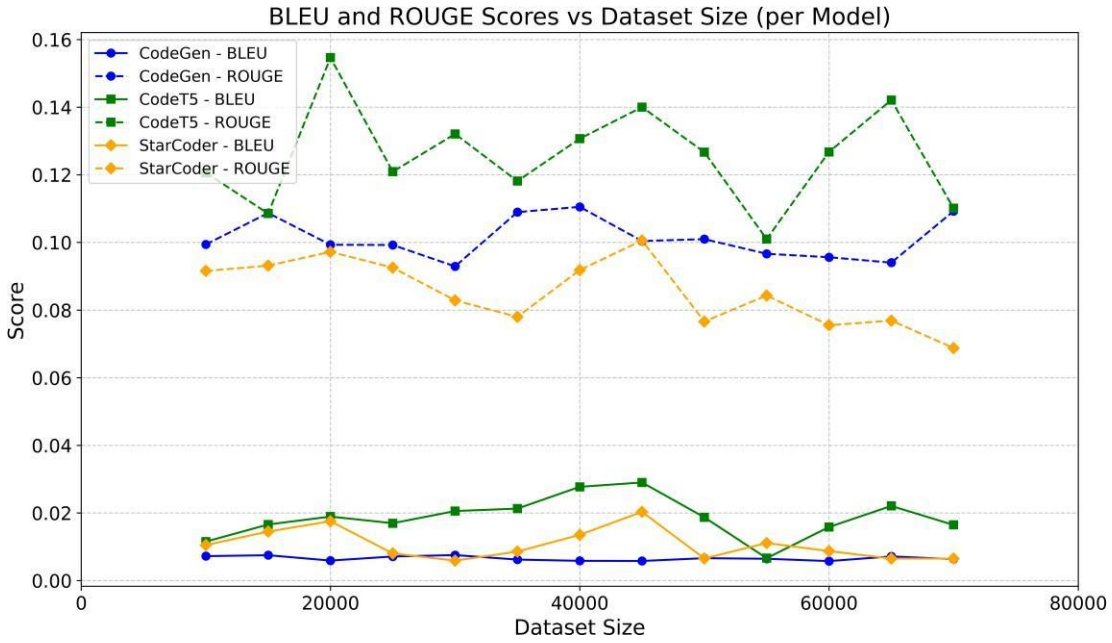


Figure 2: Comparing how under-study models perform in terms of BLEU and ROUGE scores as the dataset size increases, clearly showing which models handle more data better and which struggle.

Similarly, the average BLEU and ROUGE score for individual model is also provided in the table 1.

Table 1: Average BLEU and ROUGE scores achieved by each model after fine-tuning, showing their overall performance across all datasets.

Metric	CodeGen	CodeT5	TinyStarCoder
Average BLEU Score	0.0066	0.0186	0.0106
Average ROUGE Score	0.1012	0.1256	0.0854

Additionally, the line graph visualizing the time difference for the fine-tuning process of models is also depicted in Fig. 3.

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

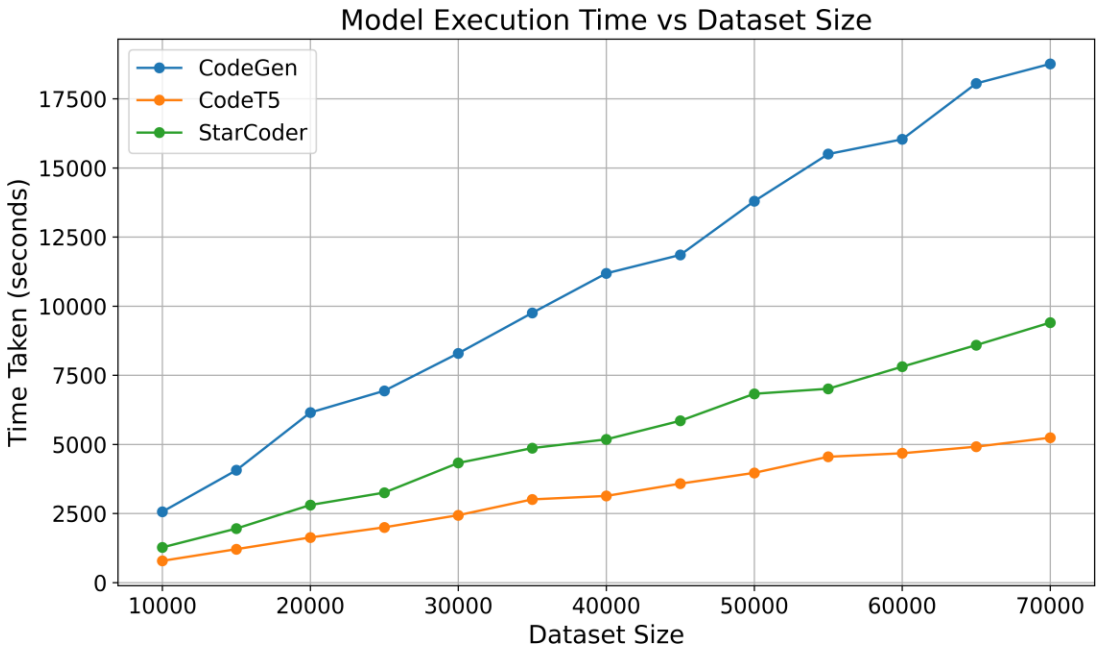


Figure 3: Comparing how long each model takes to complete fine-tuning on different dataset sizes, given the same hyperparameters and conditions, to notice which models are faster or slower to learn as data scales.

DISCUSSION

This section presents a comparative analysis of the three models, CodeGen, CodeT5, and TinyStarCoder, across different dataset sizes.

Impact of Dataset Size on BLEU Scores

The BLEU scores (solid lines) in Fig. 2 indicate a progression through changing dataset size for each model. From looking at the scores plotted when connected in this graph we can start to see a pattern between the three models and their differences. CodeT5 is the clear winner, with its BLEU score increasing from .0115 with a 10k dataset to .0290 scoring with a 45k dataset, This indicates that CodeT5 is able to learn the code generation process better and improve upon its stopping conditions, due to the number of code generation examples and sizes utilized during training as reported. There is a slight dip in performance after reaching the best score for CodeT5, as previously mentioned; therefore it appears as though CodeT5 does have the ability to utilize larger datasets and improve the quality of the code it is capable of generating.

TinyStarCoder's BLEU score started strong with a .0105 BLEU score at 10k samples and held that score until reaching a maximum of .0203 at 45,000 samples; however the tiny starcoder's score dropped to a range of .0065 at 70k samples. This indicates that TinyStarCoder is benefiting from additional training data but is having trouble generalizing from it or is experiencing over- fitting from too many examples of large datasets.At the same time, CodeGen has a relatively constant BLEU score across the entire dataset. Although it starts at 0.0072 (10k samples), it dropped to a low of

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

0.0059 (20k samples) and reached a peak of 0.0075 (30k samples) before settling in between these two points with little growth overall. After examining the entire dataset, CodeGen only improved slightly (0.0063) when it was trained on up to 70,000 samples. The indication here is that CodeGen may be limited by either: its learning ability or model structure in terms of being able to learn from more data to achieve greater BLEU scores. Although there was clear improvement by CodeT5 as additional data was added; TinyStarCoder is proving to have potential, but there are inconsistencies, and for CodeGen, as more data is used to train the model, it doesn't seem to be able to utilize it for improved BLEU scores.

Impact of Dataset Size on ROUGE Scores

The dotted line representations of ROUGE scores in Fig. 2 illustrate that there is a more elastic relationship between size of the dataset and model performance than with BLEU. This can be demonstrated by how distinct the models react when provided with an increasing number of training examples, illustrating how unique their architectures and generalization capabilities are. Through the increase in size of the training datasets, CodeT5 exhibited the most stable trajectory with respect to the increasing ROUGE scores, beginning at a score of ROUGE 0.0249 (i.e., the size of the dataset was 10,000 entries) and continuing steadily to ROUGE 0.0408 (i.e., dataset size = 30,000 entries) and ROUGE 0.0460 (i.e., dataset size = 65,000 entries). There was a small decline to ROUGE 0.0436 (dataset size = 70K); however, the trend is an upward trajectory. Therefore, CodeT5 is better able to capture more semantic and structural information than CodeGen due to its use of a large number of available datasets as input training examples.

In comparison, CodeGen exhibited only a limited growth in the ROUGE scores across the same range of dataset sizes with only some irregular increases. CodeGen started with ROUGE 0.01408 at the beginning of the dataset size of 10,000, steadily increasing but with fluctuations along the way. The most significant drops in ON every two points occurred at 20,000 entries (ROUGE 0.0122) and again at 40,000 entries (ROUGE 0.0129), eventually rising to 0.0154 at 70,000 entries. This shows that CodeGen is not capable of benefiting as much from the larger datasets compared to CodeT5.

TinyStarCoder initially shows good performance, jumping from 0.0154 (10k) to 0.0319 (30k). However, the model becomes unstable as the dataset grows larger. After reaching 0.0319, it begins to decline, dropping to 0.0239 (40k), 0.0198 (60k), and finally 0.0183 at 70,000. This decline suggests that TinyStarCoder might struggle with generalization or suffer from overfitting when exposed to high-volume training data.

Overall Performance and Efficiency comparison of models

Zsolt et al. [37] reports a monotonic improvement in BLEU and ROUGE scores with increased fine-tuning data (i.e. Performance, Data Volume), our results show a diminishing-returns curve, where performance gains begin to plateau once dataset size exceeds 60% of the full CodeSearchNet-Python corpus. According to the results in Fig. 3, CodeGen took longer to fine- tune at each dataset size, but it did not improve its performance in any way; the model still showed non-pleasing flat

Annual Methodological Archive Research Review
http://amresearchreview.com/index.php/Journal/about
Volume 3, Issue 12 (2025)
http://amresearchreview.com/index.php/Journal/about
Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

performance. TinyStarCoder took moderate time, whereas CodeT5 showed the best-improving performance and took less time compared to other models. The reason for this may be its transformer-based architecture. Our study does not prescribe any single optimal dataset size. Rather, it offers a scalable decision framework that helps practitioners balance dataset volume against model performance, enabling them to determine an appropriate size based on their computational constraints and accuracy requirements.

CONCLUSION AND FUTURE WORK

Our research indicates that although CodeT5-small has the lowest number of parameters among the models studied, it consistently performs better than all of the other models regardless of the increase in dataset size, demonstrating the capability of the model to learn from a large amount of data. Due to the smaller size of the training dataset, TinyStarCoder-Py does well with small datasets and is able to train effectively with a small amount of training data; however, it becomes less effective at learning and generalising at larger sizes as the number of training examples increases. CodeGen-350M-mono is the model that is least responsive to increased dataset size; therefore, it has the least amount of capacity to learn from an increase in data. The findings from this research highlight the importance of dataset size in determining the performance of the models used for code generation. Due to the limited resources available to train the models used in this research, all of the models used in this study were smaller models with fewer parameters and were trained on smaller datasets. Moving forward, we will strive to create larger models that include more parameters and to test them on larger datasets while also including additional advanced evaluation metrics such as CodeBLEU, ExactMatch, Pass@k etc.

References

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein et al., “On the opportunities and risks of foundation models,” arXiv:2108.07258, 2021. [Online]. Available: https://arxiv.org/abs/2108.07258

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, “Evaluating large language models trained on code,” arXiv:2107.03374, Jul 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

H. Naveed et al., “A comprehensive overview of large language models,” arXiv:2307.06435, August 2023. [Online]. Available: https://arxiv.org/abs/2307.06435

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry,

Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child,

Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark,

Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” arXiv:2005.14165, vol. 4, May 2020. [Online].

Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

Available: https://arxiv.org/abs/2005.14165

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," https://cdn.openai.com/research-covers/language-unsupervised/language understanding paper.pdf, 2018, openAI Technical Report.

R. Anil, Y. Bai, Y. Bai, B. Balle, D. Banarse, E. Barnes, I. Barr, S. Borgeaud, A. Brock, T. Cai et al., "Gemini: A family of highly capable multimodal models," arXiv:2312.11805, Dec. 2023. [Online]. Available: https://arxiv.org/abs/2312.11805H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozi`ere, N. Goyal, ´ Edouard Grave, G. Lample, and F. Fleuret, "Llama: Open and efficient foundation language models," arXiv:2302.13971, Feb 2023.

D. Guo et al., "Deepseek-coder: When the large language model meets programming– the rise of code intelligence," arXiv:2401.14196, Jan 2024. [Online]. Available: https://arxiv.org/abs/2401.14196

S. Nettur, S. Karpurapu, U. Nettur, L. Gajja, S. Myneni, and A. Dusi, "The role of github copilot on software development: A perspective on productivity, security, best practices and future directions," arXiv:2502.13199, 2025. [Online]. Available: https://arxiv.org/pdf/2502.13199

J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," arXiv:2406.00515, Jun 2024. [Online]. Available: https://arxiv.org/abs/2406.00515

B. Zhang, Z. Liu, C. Cherry, and O. Firat, "When scaling meets llm finetuning: The effect of data, model and finetuning method," arXiv:2402.17193, 2024. [Online]. Available: https://arxiv.org/abs/2402.17193

J. Austin et al., "Program synthesis with large language models," arXiv:2108.07732, Aug 2021. [Online].

Available: https://arxiv.org/abs/2108.07732

OpenAI, "GPT-4," https://openai.com/index/gpt-4-research/, Mar. 2023, accessed: 2025-07-01.

B. Roziere, J. Gehring, S. Grapperon, G. Lample, A. Astier, A. Fan, and A. Joulin, "Code llama: Open foundation models for code," arXiv:2308.12950, Aug. 2023. [Online]. Available: https://doi.org/10.48550/arxiv.2308.12950

B. Wei, G. Li, X. Xia, Z. Fu, and Z. Jin, "Code generation as a dual task of code summarization," in Advances in Neural Information Processing Systems, vol. 32, 2019, accessed: Nov. 10, 2025. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/e52ad5c9f751f599492b4f087ed7ecfc-Abstract.html

M. Li et al., "From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning," arXiv:2308.12032, Aug 2023. [Online]. Available: https://doi.org/10.48550/arxiv.2308.12032

N. Jain, T. Zhang, W.-L. Chiang, J. E. Gonzalez, K. Sen, and I. Stoica, "Llm-assisted code cleaning for training accurate code generators," arXiv:2311.14904, 2023, accessed: May 16, 2025. [Online]. Available: https://arxiv.org/abs/2311.14904

N. Christakis and D. Drikakis, "Evaluating large language models in code generation: Infinite methodology for defining the inference index," Applied Sciences, vol.

Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

15, no. 7, p. 3784, Mar 2025. [Online]. Available: https://doi.org/10.3390/app15073784

Y. Han and C. Lyu, "Multi-stage guided code generation for large language models," Engineering Applications of Artificial Intelligence, vol. 139, p. 109491, Jan 2025. [Online]. Available: https://doi.org/10.1016/j.engappai.2024.109491

V. Iyer, B. Malik, P. Stepachev, P. Chen, B. Haddow, and A. Birch, "Quality or quantity? on data scale and diversity in adapting large language models for low-resource translation," in Proceedings of the WMT 2024 Conference on Machine Translation, 2024. [Online].

Available: https://www2.statmt.org/wmt24/pdf/2024.wmt-1.128.pdf

S. Zhou, U. Alon, S. Agarwal, and G. Neubig, "Codebertscore: Evaluating code generation with pretrained models of code," arXiv:2302.05527, 2023. [Online]. Available: https://arxiv.org/pdf/2302.05527

C. Song, Z. Zhou, J. Yan, Y. Fei, Z. Lan, and Y. Zhang, "Dynamics of instruction fine-tuning for chinese large language models," arXiv:2310.19651, 2023. [Online]. Available: https://arxiv.org/abs/2310.19651

E. Nijkamp et al., "Codegen: An open large language model for code with multi-turn program synthesis," arXiv:2203.13474, Feb 2023. [Online]. Available: https://arxiv.org/abs/2203.13474M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, "Out of the bleu: How should we assess quality of the code generation models?" Journal of Systems and Software, vol. 203, p. 111741, Sep 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111741

H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," arXiv:1909.09436, 2019. [Online]. Available: https://arxiv.org/abs/1909.09436

Sentence Transformers, "CodeSearchNet Dataset," https://huggingface.co/datasets/sentence- transformers/codesearchnet, 2025, accessed: 2025 02-19.

F. T. Morales, "code-search-net-python," https://huggingface.co/datasets/Nan-Do/code-search-net-python, 2025, accessed: 2025-02-19.

GitHub, "The CodeSearchNet challenge has been concluded," Aug. 2023, accessed: 2025-05-23. [Online].

Available: https://github.com/github/CodeSearchNet

OpenAI, "openai/human-eval," https://github.com/openai/human-eval, Apr. 2024, gitHub, accessed 2024-04-02.

Salesforce, "Salesforce/codet5-small· hugging face," https://huggingface.co/Salesforce/codet5-small , Jan. 2024, accessed: 2025 06-01.

B. Chen and Z. Abedjan, "Interactive cross-language code retrieval with auto-encoders," in Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Nov 2021, pp. 167–178. [Online]. Available: https://doi.org/10.1109/ase51524.2021.9678929

Salesforce, "Salesforce/codegen-350M-mono · hugging face," https://huggingface.co/Salesforce/codegen- 350M-mono, 2022, accessed: 2025 06-01.

BigCode, "bigcode/tiny starcoder py · https://huggingface.co/bigcode/tinystarcoderpy

Online ISSN 3007-3197
Print ISSN 3007-3189
https://doi.org/10.5281/zenodo.18091368

, 2025-06-01.

R. Li, L. von Werra, H. de Vries et al., “Starcoder: May the source be with you!” arXiv:2305.06161, vol. 4, p. 12, 2023. [Online]. Available: https://arxiv.org/pdf/2305.06161

J. Ainslie, J. Lee-Thorp, M. De Jong, Y. Zemlyanskiy, F. Lebr´on, and S. Sanghai, “Gqa: Training generalized multi-query transformer models from multi-head checkpoints,” arXiv:2305.13245, 2023. [Online]. Available: https://arxiv.org/pdf/2305.13245

T. Shen, H. Peng, R. Shen, Y. Fu, Z. Harchaoui, and Y. Choi, “Film: Fill-in language models for any-order generation,” arXiv:2310.09930, 2023, accessed: Jun. 02, 2025. [Online]. Available: https://arxiv.org/abs/2310.09930

L. Z. Juh´asz, “Large language models, fine tuning, code generation,” in Proceedings of CANDO-EPE 2024, Oct 2024, pp. 191–200. [Online]. Available: https://doi.org/10.1109/cando-epe65072.2024.1077276