UNIVERSITY of VIRGINIA

# Three papers to cover:

**1. DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence**

**2. Language Models for Code Optimization: Survey, Challenges, and Future Directions**

**3. Iterative Refinement of Project-Level Code Context for Precise Code Generation with Compiler Feedback**

# DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence

Presented by:
Aditya Kakkar (zjq5mr), Aryan Sawhney (ryd2fx), Yagnik Panguluri (yye7pm)

Aditya Kakkar (zjq5mr)

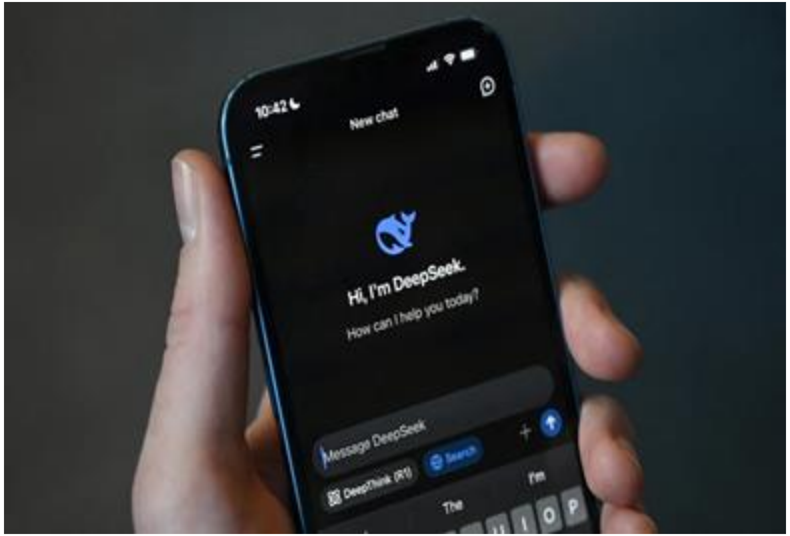
# Presentation Outline

- ❖ Introduction
- ❖ Background & Motivation
- ❖ DeepSeek Evolution
- ❖ Data Collection & Training Strategy
- ❖ Model Architecture & Improvements
- ❖ Evaluation & Benchmarks
- ❖ Competitive Programming & Code Completion
- ❖ Applications & Real-World Impact
- ❖ Limitations & Future Improvements
- ❖ Conclusion & Q&A

# Presentation Outline

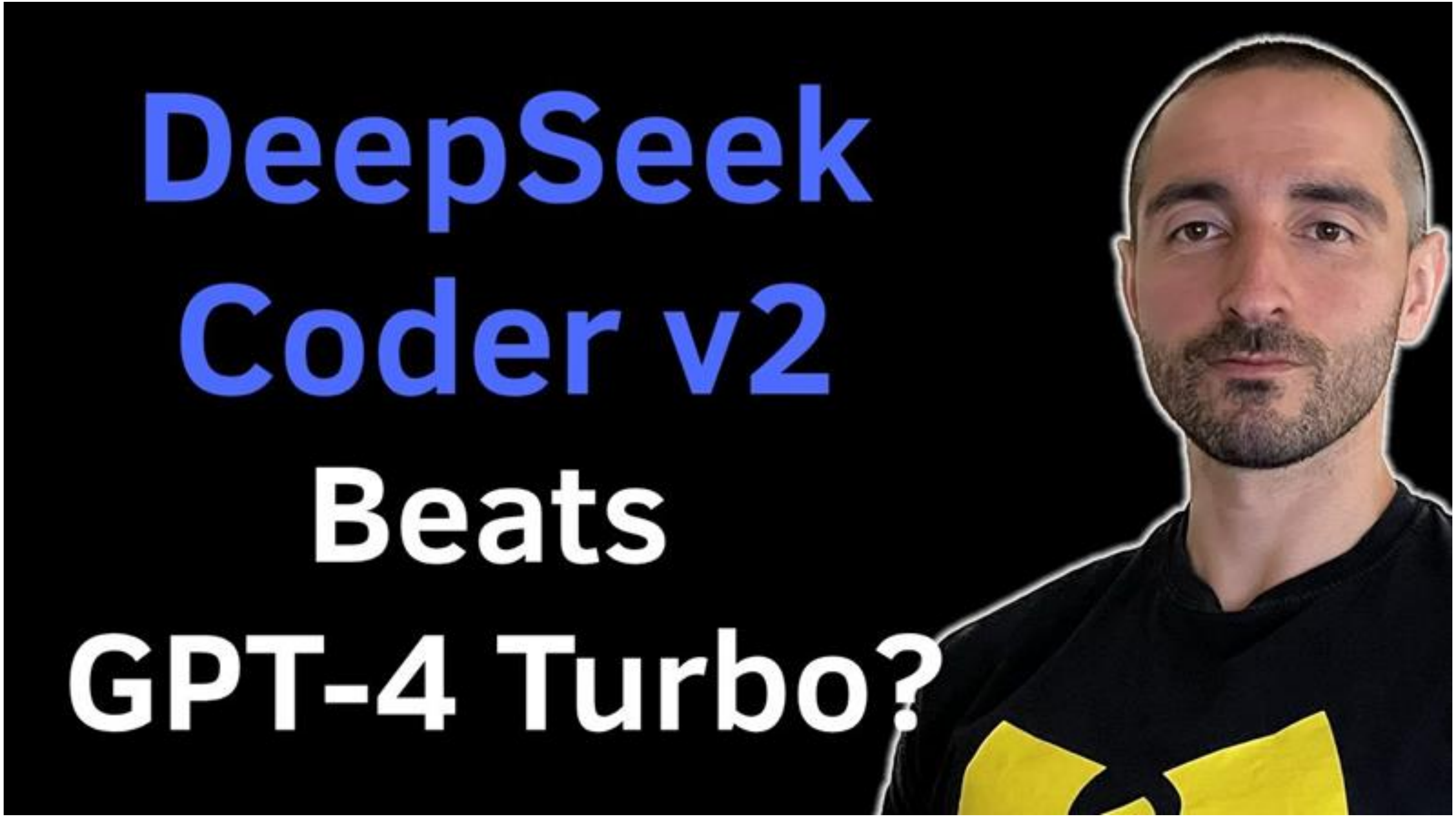
- ❖ Introduction
- ❖ Background & Motivation
- ❖ DeepSeek Evolution
- ❖ Data Collection & Training Strategy
- ❖ Model Architecture & Improvements
- ❖ Evaluation & Benchmarks
- ❖ Competitive Programming & Code Completion
- ❖ Applications & Real-World Impact
- ❖ Limitations & Future Improvements
- ❖ Conclusion & Q&A

# Introduction




DeepSeek-Coder-V2:Breaking the Barrier of Closed-Source Models in Code Intelligence

# Background



# Motivation


OPEN
SOURCE
software
copyright
applications
ideology
research
project
license
choice
free
community
internet


MONEY MOVERS
DEEPSEEK SETS OFF NEW AI RACE
CNBC

# What is DeepSeek?


DeepSeek-AI is a leading **Chinese AI research lab**, comparable to OpenAI, specializing in cutting-edge artificial intelligence advancements.

**<u>Note</u>**

This presentation covers information up to **June 2024** and does not include details on the latest **DeepSeek-R1** model.

# Evolution of DeepSeek Models

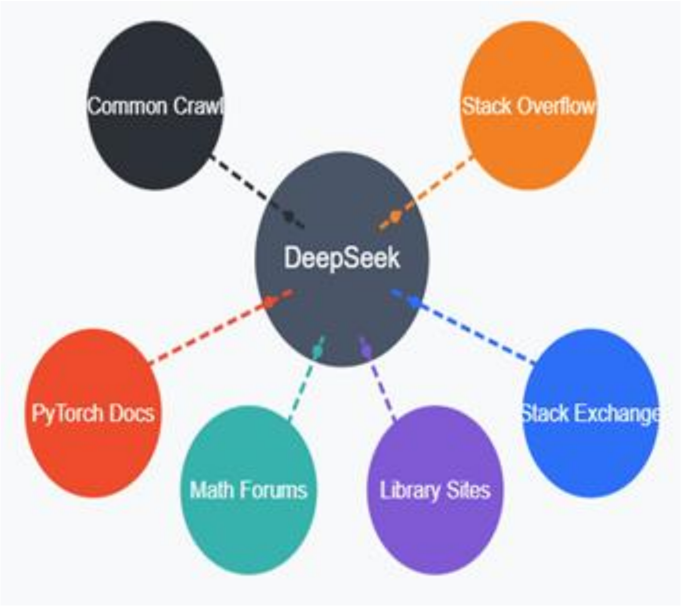

- **DeepSeek-V1 (2022):** Focused on NLP with limited coding and math capabilities.
- **DeepSeek-V2 (2023):** Introduced coding and math training with 4.2T tokens.
- **DeepSeek-Coder (2023):** Specialized in programming with 86 languages, 16K token limit.
- **DeepSeek-Coder-V2 (2024):** Expanded to 338 languages, 128K tokens, surpasses GPT-4 Turbo.

**Focus of This Presentation: We will dive into DeepSeek-Coder-V2 and its advancements.**

# DeepSeek-Coder-V2: Advancing Beyond DeepSeek-V2

| | DeepSeek-V2 | DeepSeek-Coder-V2 |
|---|---|---|
| **Code** | 60% | Expanded: 86 → 338 languages |
| **Math** | 30% | Additional datasets from coding/math forums |
| **Natural Language** | 10% | Reduced proportion |
| **Total Tokens** | 4.2T | 4.2T+6T = 10.2T |

# Data Collection




| Technique | Purpose | Benefit for DeepSeek-Coder-V2 |
| --- | --- | --- |
| **fastText** | Expand the training dataset efficiently | Helps understand rare programming terms and mathematical symbols |
| **BPE Tokenizer** | Splits text into frequent subword units | Efficiently tokenizes code across multiple languages, reducing memory usage |

# Data Filtration

- Removed low-quality and duplicate files.
- Excluded files with excessive line length (>1000 characters) or low alphabetic content (<25%).
- **Filtered out XML files (except XSLT) and ensured readable HTML content.**

Retained only files with character counts between **50 and 5000** to avoid data-heavy content.

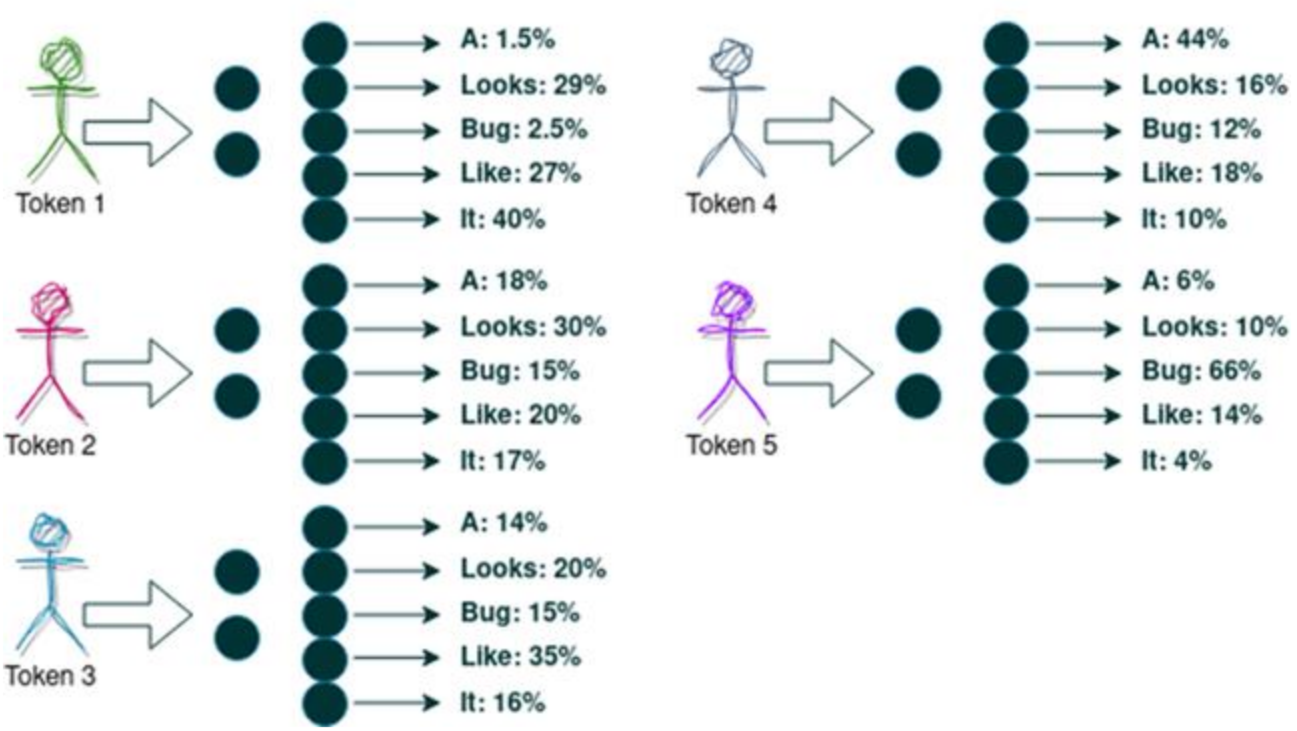
Kept files where visible text is ≥20% of the total code and at least 100 characters.

# Training Strategy: Scaling from 4.2T to 6T Tokens

| Model | DeepSeek-Coder-V2-Lite | DeepSeek-Coder-V2 |
|---|---|---|
| # Total Parameters (#TP) | 16B | 236B |
| # Active Parameters (#AP) | 2.4B | 21B |
| Pre-training Tokens | 4.2T+6T | 4.2T+6T |
| LR Scheduler | Cosine | Cosine |
| FIM | Enable | Disable |

- **Context Length Increase: 16K → 128K Tokens** → Enables handling larger codebases and multi-file projects.
- **Advanced Training Methods** → Fill-In-Middle (FIM), Group Relative Policy Optimization (GRPO), and Reinforcement Learning (RL) for better accuracy.

# Training Techniques: Probability Distributions



# Training Techniques: Next-Token Prediction

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return
```

Model Prediction:

```
python

def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

# Training Techniques: Fill-In-Middle (DeepSeek-Coder-V2-16B)

The model assigns probabilities to possible completions and selects the most likely one.

```python
def factorial(n):
    if n == 0:
        return 1
    <|fim_hole|>
    else:
        return n * factorial(n - 1)
```

**Model Prediction (Middle Completion):**

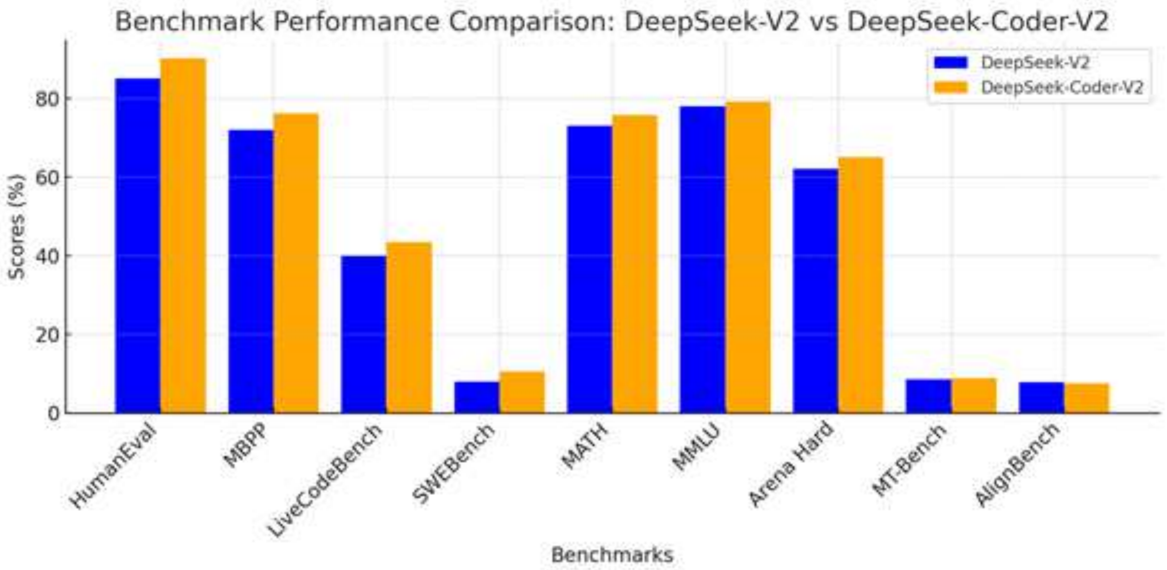
python

```python
def factorial(n):
    if n == 0:
        return 1
    elif n < 0:
        raise ValueError("Input must be a non-negative integer.")
    else:
        return n * factorial(n - 1)
```

# Ablation studies

| Model | Tokens | Python | C++ | Java | PHP | TS | C# | Bash | JS | Avg | MBPP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DeepSeek-Coder-1B | 1T | 30.5% | 28.0% | 31.7% | 23.0% | 30.8% | 31.7% | 9.5% | 28.6% | 26.7% | 44.6% |
| DeepSeek-Coder-V2-1B | 1T | 36.0% | 34.8% | 31.7% | 27.3% | **37.7%** | 34.2% | 6.3% | **38.5%** | 31.2% | 49.0% |
| DeepSeek-Coder-V2-1B | 2T | **37.2%** | **39.1%** | **32.3%** | **31.7%** | 34.6% | **36.7%** | **12.0%** | 32.9% | **32.0%** | **54.0%** |

Table 1 | Performance of 1B base model between DeepSeek-Coder and DeepSeek-Coder-V2.



# Ablation studies

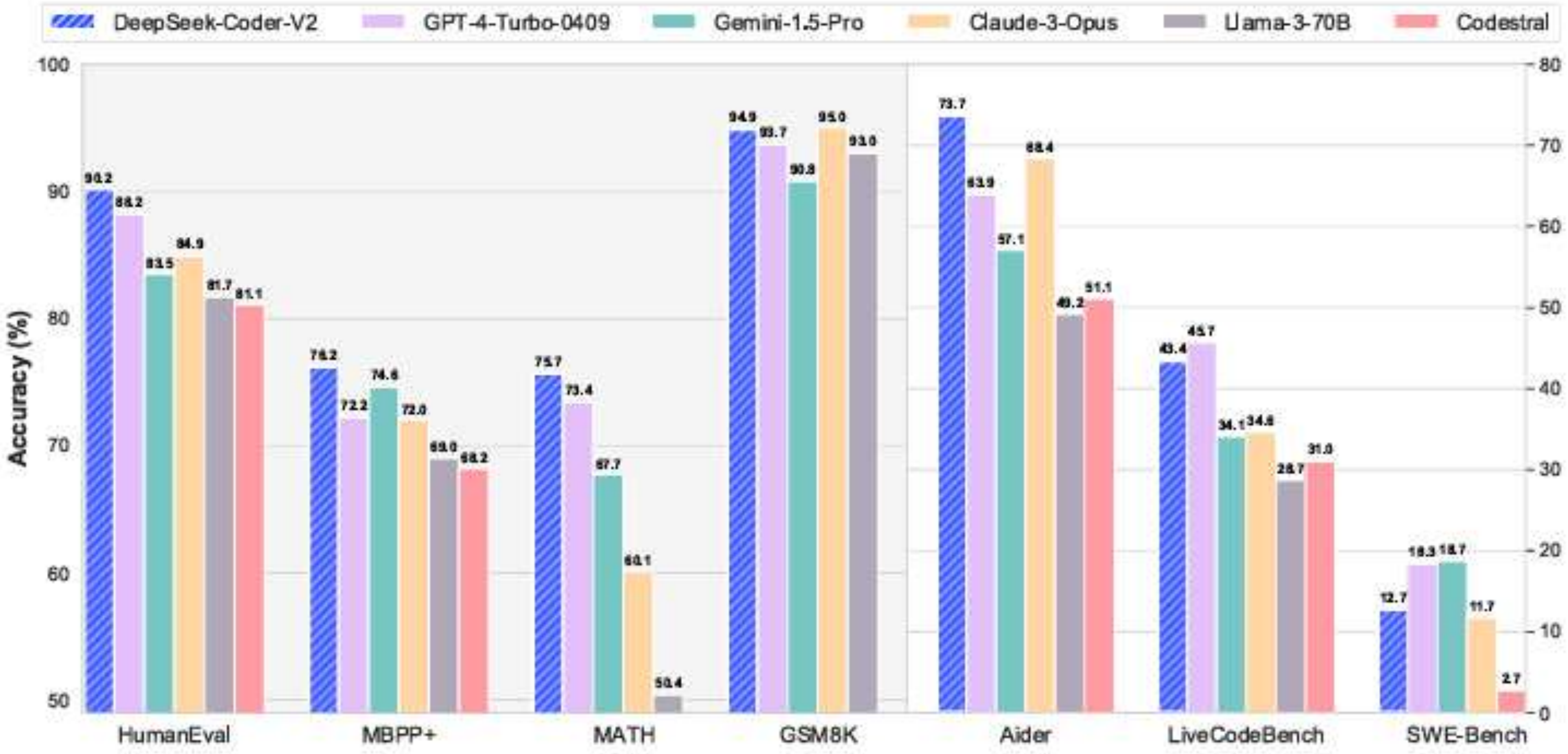



Figure 1 | The Performance of DeepSeek-Coder-V2 on math and code benchmarks.

UNIVERSITY of VIRGINIA

Yagnik Panguluri (yye7pm)

# Training Hyper-Parameters

- **Optimizer:** AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay = 0.1)
- **Learning rate schedule:** Cosine decay
  a. 2000 warm up steps
  b. Decays to 10% of initial LR
- **Batch size tuning per DeepSeek-V2 methodology**

| Model | DeepSeek-Coder-V2-Lite | DeepSeek-Coder-V2 |
|---|---|---|
| # Total Parameters (#TP) | 16B | 236B |
| # Active Parameters (#AP) | 2.4B | 21B |
| Pre-training Tokens | 4.2T+6T | 4.2T+6T |
| LR Scheduler | Cosine | Cosine |
| FIM | Enable | Disable |

Table 2 | Training Setting of DeepSeek-Coder-V2.

# Long Context Extension

DeepSeek-Coder-V2

YARN (Yet Another Retrieval Network)

DeepSeek-Coder-V2-128K

16K Tokens

Limitation for handling longer codebases, documents, and complex tasks

128K Tokens

Helps in long-form reasoning, retrieval tasks, and handling entire software repositories in a single pass

# NIAH Performance

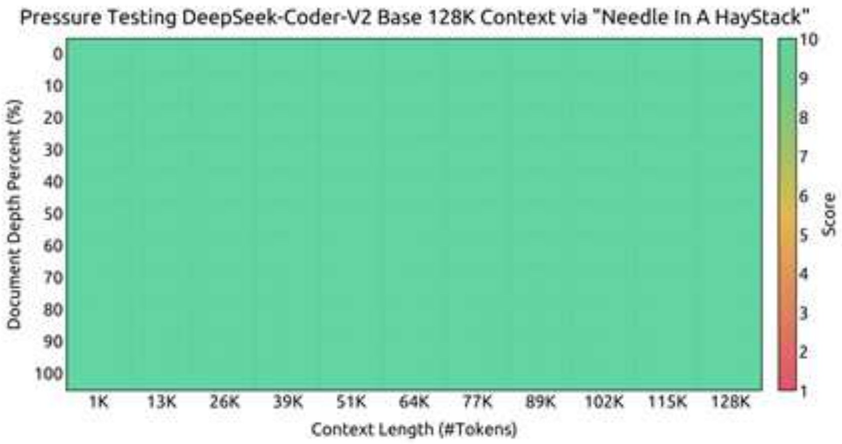



Figure 2 | Evaluation results on the "Needle In A Haystack" (NIAH) tests. DeepSeek-Coder-V2 performs well across all context window lengths up to 128K.

- DeepSeek-Coder-V2 maintains strong retrieval performance across all testing context lengths
- Model demonstrates consistent accuracy up to 128K tokens, outperforming many prior open source models
- Upsampling of long-context data during training enhances model robustness for long-context tasks.

# Alignment

Alignment ensures the model generates accurate, human-preferred responses

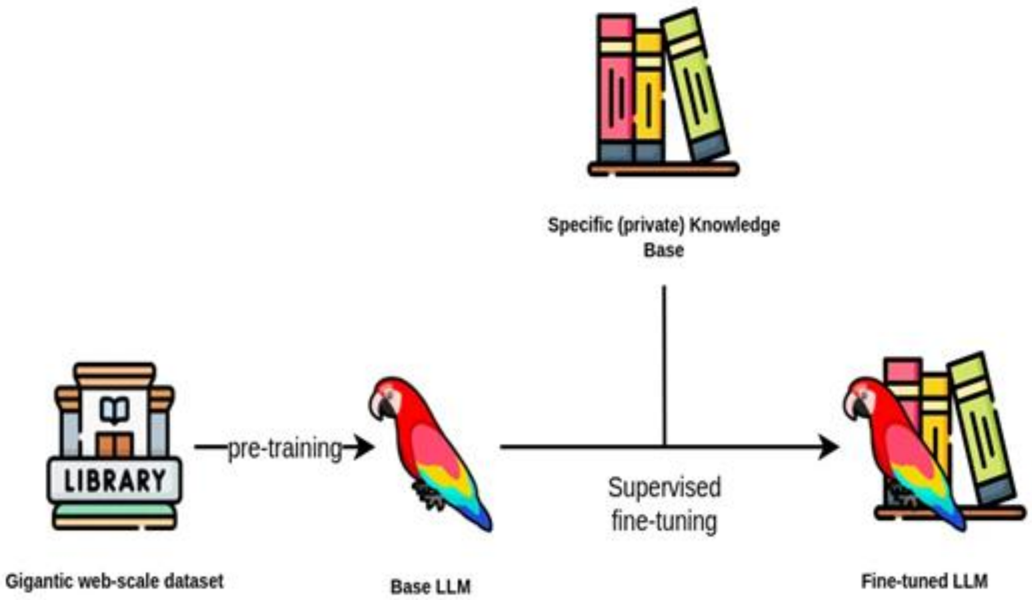
Optimizes the model behavior for code generation, math reasoning, and instruction-following

**Supervised Fine Tuning**

**Reinforcement Learning**

# Supervised Fine Tuning

- A method to refine the model's capabilities by training it on curated instruction-following datasets
- Ensures the model understands instructions and generates accurate code/math responses
- Prepares the model for RL alignment





Source

# SFT - Setup

## Configurations

- **Learning rate:** 5e-6
- **Learning rate schedule:** Cosine decay with 100 warm-up steps
- **Batch size**: 1M tokens per batch
- **Total training tokens:** 1B tokens

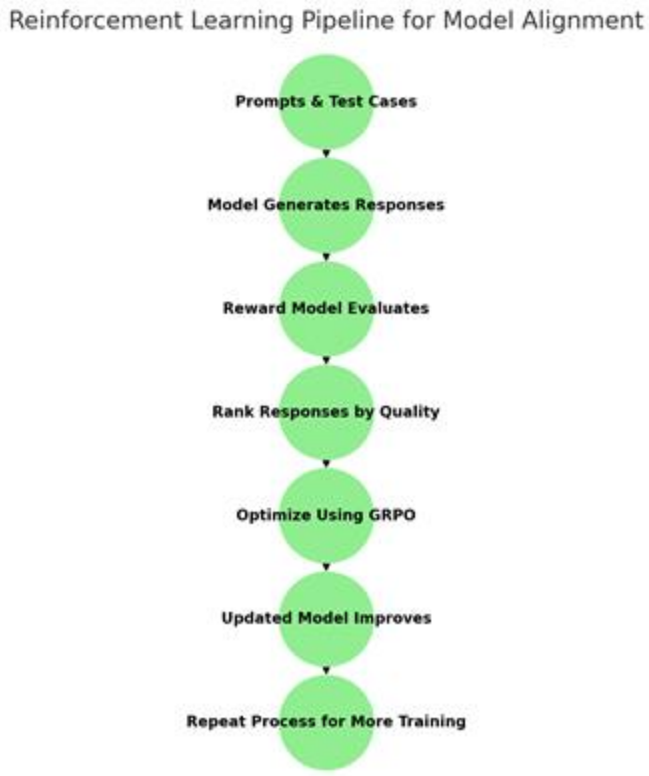
## Optimizations

- Cosine decay learning rate
- High quality instruction dataset
- Large batch size

## Outcomes

- Better instruction-following performance
- Stronger generalization across coding and mathematical tasks
- Reduces errors in multi-step reasoning

# Reinforcement Learning




- SFT helps, but it's limited by static datasets
- RL further optimizes the model's response by learning from dynamic feedback
- Improves performance on code/math tasks by training with real-world prompts
- Reduces errors and hallucinations

# Prompts

Prompts serve as inputs to the model, helping refine its code generation and problem solving abilities

Each code prompt is paired with test cases to validate correctness

Collected and filtered 40K+ prompts

| Prompt Type | Description |
|---|---|
| **Code Prompts** | Algorithmic tasks, debugging challenges |
| **Math Prompts** | Complex problem-solving, theorem proving |
| **General Instructions** | Instruction-based reasoning tasks |

# Reward Modeling

A model which assigns a quality score to generated responses

Helps train the model to prefer better responses based on correctness, efficiency, and alignment with human preferences

Replaces raw compiler signal, which only provides binary (pass/fail) feedback

Collects human preferences or ground truth labels

Trains a model to predict response quality

Uses the predicted reward to fine-tune the main model via RL

# Reward Modeling Results

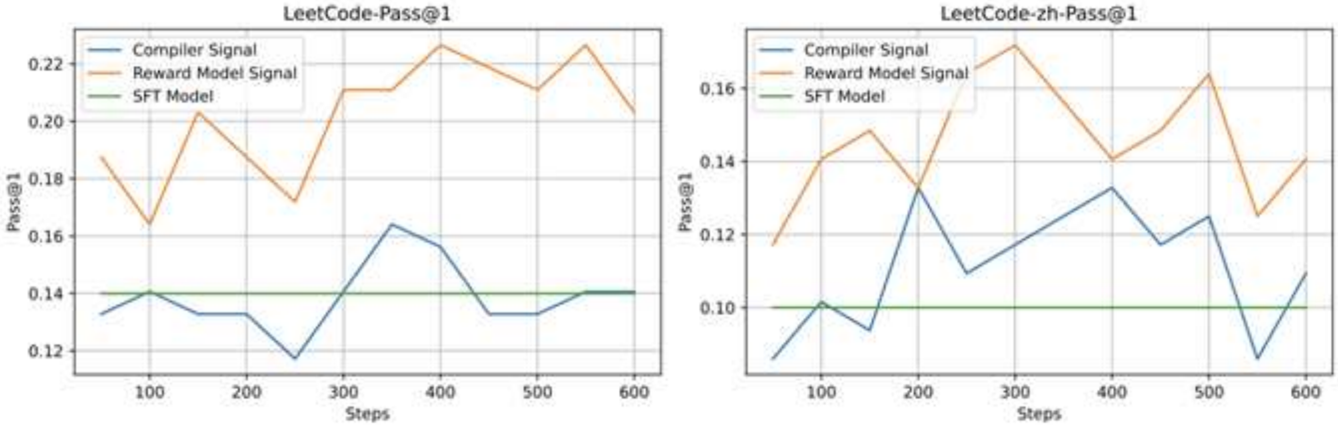



Figure 3 | Performances of Different Methods

# Reinforcement Learning - GRPO

**GRPO -** Group Relative Policy Optimization

An efficient RL algorithm used to improve the DeepSeek-Coder-V2

Similar to PPO but more efficient and cost effective

Model generates multiple responses for a given prompt

Reward model ranks the responses based on quality

GRPO optimizes the model to favor higher-ranked responses

Process repeats

Aryan Sawhney (ryd2fx)

# Results - Comparison Models

Evaluate DeepSeek-Coder-V2 on three types of tasks:

- Coding
- Mathematics
- General natural language

Compared DeepSeek-Coder-V2 with the previous state-of-the-art large language models:

**Open Source**

- StarCoder
- StarCoder2
- CodeLlama
- DeepSeek-Coder (previous version)
- Codestral
- Llama3

**Closed Source**

- GPT-4
- GPT-4 Turbo
- GPT-4o
- Claude 3 Opus
- Gemini 1.5 Pro

# Results - HumanEval & MBPP

| | #TP | #AP | Python | Java | C++ | C# | TS | JS | PHP | Bash |
|---|---|---|---|---|---|---|---|---|---|---|
| **Closed-Source Models** | | | | | | | | | | |
| Gemini-1.5-Pro | - | - | 83.5% | 81.0% | 78.3% | 75.3% | 77.4% | 80.8% | 74.5% | 39.9% |
| Claude-3-Opus | - | - | 84.2% | 78.5% | 81.4% | 74.7% | 76.1% | 75.8% | 78.3% | 48.7% |
| GPT-4-1106 | - | - | 87.8% | **82.3%** | 78.9% | 80.4% | 81.8% | 80.1% | 77.6% | **55.7%** |
| GPT-4-Turbo-0409 | - | - | 88.2% | 81.7% | 78.3% | 79.1% | 79.3% | 80.8% | 78.9% | **55.1%** |
| GPT-4o-0513 | - | - | **91.0%** | 80.4% | **87.0%** | **82.9%** | **86.2%** | **87.6%** | **79.5%** | 53.8% |
| **Open-Source Models** | | | | | | | | | | |
| Codestral | 22B | 22B | 78.1% | 71.5% | 71.4% | 77.2% | 72.3% | 73.9% | 69.6% | 47.5% |
| DS-Coder-instruct | 33B | 33B | 79.3% | 73.4% | 68.9% | 74.1% | 67.9% | 73.9% | 72.7% | 43.0% |
| Llama3-Instruct | 70B | 70B | 81.1% | 67.7% | 64.0% | 69.6% | 69.8% | 70.2% | 65.8% | 36.1% |
| DS-Coder-V2-Lite-Instruct | 16B | 2.4B | 81.1% | 76.6% | 75.8% | 76.6% | 80.5% | 77.6% | 74.5% | 43.0% |
| DS-Coder-V2-Instruct | 236B | 21B | **90.2%** | **82.3%** | **84.8%** | **82.3%** | **83.0%** | **84.5%** | **79.5%** | **52.5%** |

| | #TP | #AP | Swift | R | Julia | D | Rust | Racket | MBPP+ | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| **Closed-Source Models** | | | | | | | | | | |
| Gemini-1.5-Pro | - | - | 66.5% | 53.4% | 71.7% | 55.8% | 73.1% | 48.4% | **74.6%** | 68.9% |
| Claude-3-Opus | - | - | 63.9% | 55.9% | 76.1% | 60.3% | 71.2% | **64.6%** | 72.0% | 70.8% |
| GPT-4-1106 | - | - | 62.7% | 57.8% | 69.2% | 60.9% | **78.8%** | 64.0% | 69.3% | 72.5% |
| GPT-4-Turbo-0409 | - | - | 63.9% | 56.5% | 69.8% | **61.5%** | **78.8%** | 63.4% | 72.2% | 72.3% |
| GPT-4o-0513 | - | - | **75.9%** | **65.2%** | **78.0%** | 60.9% | **80.1%** | **64.6%** | 73.5% | **76.4%** |
| **Open-Source Models** | | | | | | | | | | |
| Codestral | 22B | 22B | 63.3% | 49.7% | 67.9% | 32.1% | 67.3% | 37.3% | 68.2% | 63.2% |
| DS-Coder-instruct | 33B | 33B | 61.4% | 44.7% | 53.5% | 31.4% | 68.6% | 46.0% | 70.1% | 61.9% |
| Llama3-Instruct | 70B | 70B | 55.1% | 46.0% | 62.9% | 48.1% | 58.3% | 46.0% | 68.8% | 60.6% |
| DS-Coder-V2-Lite-Instruct | 16B | 2.4B | 64.6% | 47.8% | 67.3% | 45.5% | 62.2% | 41.6% | 68.8% | 65.6% |
| DS-Coder-V2-Instruct | 236B | 21B | **72.2%** | **64.0%** | **72.3%** | **64.1%** | **78.2%** | **63.4%** | **76.2%** | **75.3%** |

**Benchmarks:**

- HumanEval
- MBPP+
- Multilingual Evaluation
  - C++, Java, PHP, TypeScript, C#, Bash, JavaScript, Swift, R, Julia, D, Rust, and Racket.

**DeepSeek-Coder-V2-Instruct Performance:**

- Achieves the second-highest average score of **75.3%**, surpassed only by GPT-4o, which leads with 76.4%
- Top-tier results across multiple languages, achieving the highest scores in Java and PHP and strong performances in Python, C++, C#, TypeScript, and JavaScript.

**DeepSeek-Coder-V2-Lite-Instruct Performance:**

- Outperforms DeepSeek V1 (larger 33B model) with an average score of 65.6% vs. 61.9% despite its smaller size

# Results - Competitive Programming

| Model | #TP | #AP | LiveCodeBench Easy (82) | LiveCodeBench Medium (87) | LiveCodeBench Hard (57) | LiveCodeBench Overall (226) | USACO |
|---|---|---|---|---|---|---|---|
| **Closed-Source Models** | | | | | | | |
| Gemini-1.5-Pro | - | - | 74.9% | 16.8% | 1.8% | 34.1% | 4.9% |
| Claude-3-Opus | - | - | 77.2% | 16.7% | 0.7% | 34.6% | 7.8% |
| GPT-4-1106 | - | - | 78.4% | 20.2% | 3.5% | 37.1% | 11.1% |
| GPT-4-Turbo-0409 | - | - | 84.1% | **35.4%** | **6.1%** | **45.7%** | 12.3% |
| GPT-4o-0513 | - | - | **87.4%** | 27.5% | 4.9% | 43.4% | **18.8%** |
| **Open-Source Models** | | | | | | | |
| Codestral | 22B | 22B | 66.5% | 17.7% | 0.2% | 31.0% | 4.6% |
| DS-Coder-instruct | 33B | 33B | 51.6% | 9.7% | 0.4% | 22.5% | 4.2% |
| Llama3-Instruct | 70B | 70B | 62.4% | 14.4% | 2.1% | 28.7% | 3.3% |
| DS-Coder-V2-Lite-Instruct | 16B | 2.4B | 58.5% | 8.0% | 0.0% | 24.3% | 6.5% |
| DS-Coder-V2-Instruct | 236B | 21B | **84.1%** | **29.9%** | **5.3%** | **43.4%** | **12.1%** |

**Benchmarks:**

- LiveCodeBench
  - Gathers novel challenges from LeetCode, AtCoder, and CodeForces.
  - Uses the subset (1201-0601) since the training data cut-off is before November 2023
- USACO
  - Contains 307 problems from the USA Computing Olympiad

**DeepSeek-Coder-V2-Instruct Performance:**

- Tied for second overall at 43.4%, matching GPT-4o just behind GPT-4-Turbo-0409, which leads with 45.7%.
- Demonstrates strong capability in handling complex coding challenges.
- Establishes itself as a top contender, closely trailing GPT-4-Turbo.

# Results - Repository-Level Completion

**Benchmarks:**

- RepoBench
  - Sources data from GitHub repositories; cut-off is before November 2023
  - Covers two programming languages: Python and Java
  - Five context length levels: 2k, 4k, 8k, 12k, and 16k tokens

**DeepSeek-Coder-V2-Lite-Base Model:**

- Python performance comparable to DeepSeek-Coder-Base 33B (V1)
- Java performance comparable to DeepSeek-Coder-Base 7B (V1)
- Slightly lower performance but faster than CodeStral in code completion tasks due to having only one-tenth of the active parameters

| Model | #TP | #AP | Python | | | | | | Java | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2k | 4k | 8k | 12k | 16k | Avg | 2k | 4k | 8k | 12k | 16k | Avg |
| StarCoder2-Base | 15B | 15B | 35.7% | 36.7% | 34.6% | 27.4% | 25.1% | 32.1% | 46.2% | 45.0% | 39.8% | 30.5% | 30.7% | 38.7% |
| CodeLlama-Base | 7B | 7B | 32.0% | 34.4% | 35.3% | 33.3% | 32.2% | 33.5% | 43.1% | 42.1% | 40.4% | 37.0% | 40.3% | 40.6% |
| CodeLlama-Base | 13B | 13B | 33.0% | 36.5% | 37.0% | 34.6% | 35.0% | 35.2% | 43.5% | 44.8% | 40.7% | 38.6% | 41.1% | 41.8% |
| CodeLlama-Base | 34B | 34B | 35.3% | 37.5% | 39.5% | 34.9% | 35.6% | 36.6% | 45.9% | 45.4% | 42.5% | 41.0% | 41.2% | 43.3% |
| DS-Coder-Base | 6.7B | 6.7B | 36.1% | 37.5% | 38.2% | 34.0% | 35.0% | 36.2% | 46.8% | 46.4% | 42.9% | 38.8% | 40.8% | 43.3% |
| DS-Coder-Base | 33B | 33B | 39.7% | 40.1% | 40.0% | 36.9% | 38.5% | 39.1% | 47.9% | 47.7% | 43.3% | 40.9% | 43.6% | 44.8% |
| Codestral | 22B | 22B | **42.1%** | **44.3%** | **46.6%** | **46.6%** | **51.5%** | **46.1%** | 48.3% | **47.8%** | **46.0%** | **42.2%** | **43.9%** | **45.7%** |
| DS-Coder-V2-Lite-Base | 16B | 2.4B | 38.3% | 38.6% | 40.6% | 38.3% | 38.7% | 38.9% | **48.8%** | 45.7% | 42.4% | 38.1% | 41.1% | 43.3% |

# Results - Fill-in-the-Middle Code Completion

| Model | #TP | #AP | python | java | javascript | Mean |
|---|---|---|---|---|---|---|
| StarCoder[6] | 16B | 16B | 71.5% | 82.3% | 83.0% | 80.2% |
| CodeLlama-Base | 7B | 7B | 58.6% | 70.6% | 70.7% | 68.0% |
| CodeLlama-Base | 13B | 13B | 60.7% | 74.3% | 78.5% | 73.1% |
| DS-Coder-Base | 1B | 1B | 74.1% | 85.1% | 82.9% | 81.8% |
| DS-Coder-Base | 7B | 7B | 79.8% | **89.6%** | 86.3% | 86.1% |
| DS-Coder-Base | 33B | 33B | **80.5%** | 88.4% | 86.6% | **86.4%** |
| Codestral | 22B | 22B | 77.2% | 83.2% | 85.9% | 83.0% |
| DS-Coder-V2-Lite-Base | 16B | 2.4B | 80.0% | 89.1% | **87.2%** | **86.4%** |

**Benchmarks:**

- Single-Line Infilling
  - Benchmarks ability to adeptly complete code by filling in blanks using the surrounding context
  - Covers three programming languages: Python, Java, JavaScript

**DeepSeek-Coder-V2-Lite-Base Performance:**

- Achieves significantly high scores across all languages
- Tied with DeepSeek-Coder-Base 33B (V1) for highest mean score of 86.4% despite only having 2.4B active parameters

# Results - Code Fixing

**DeepSeek-Coder-V2-Instruct Performance:**

- Achieved the best performance within the open source models
- Achieved the highest score in Aider with 73.7%, outperforming all models, including closed-source counterparts

| Model | #TP | #AP | Defects4J | SWE-Bench | Aider |
|---|---|---|---|---|---|
| Closed-Source Models | | | | | |
| Gemini-1.5-Pro | - | - | 18.6% | 19.3% | 57.1% |
| Claude-3-Opus | - | - | 25.5% | 11.7% | 68.4% |
| GPT-4-1106 | - | - | 22.8% | 22.7% | 65.4% |
| GPT-4-Turbo-0409 | - | - | 24.3% | 18.3% | 63.9% |
| GPT-4o-0513 | - | - | **26.1%** | **26.7%** | **72.9%** |
| Open-Source Models | | | | | |
| Codestral | 22B | 22B | 17.8% | 2.7% | 51.1% |
| DS-Coder-Instruct | 33B | 33B | 11.3% | 0.0% | 54.5% |
| Llama3-Instruct | 70B | 70B | 16.2% | - | 49.2% |
| DS-Coder-V2-Lite-Instruct | 16B | 2.4B | 9.2% | 0.0% | 44.4% |
| DS-Coder-V2-Instruct | 236B | 21B | **21.0%** | **12.7%** | **73.7%** |

**Benchmarks:**

- Defects4J:
  - Contains real-world software bugs from open-source projects like Apache Commons, JFreeChart, and Closure Compiler
  - Selected 238 bugs that require modifying only one method
- SWE-bench:
  - Evaluates LLMs on real-world GitHub issues by providing a codebase with a specific issue and requiring a generated patch
- Aider Benchmark:
  - Tests LLMs' ability to modify Python code, assessing coding skill and consistency in following prompt specifications
  - Includes 133 distinct coding tasks

# Results - Code Understanding and Reasoning

| Model | #TP | #AP | CruxEval-I-COT | CruxEval-O-COT |
|---|---|---|---|---|
| | | Closed-Source Models | | |
| Gemini-1.5-Pro | - | - | 67.0% | 77.5% |
| Claude-3-Opus | - | - | 73.4% | 82.0% |
| GPT-4-1106 | - | - | 75.5% | 77.1% |
| GPT-4-Turbo-0409 | - | - | 75.7% | 82.0% |
| GPT-4o-0513 | - | - | **77.4%** | **88.7%** |
| | | Open-Source Models | | |
| Codestral | 22B | 22B | 48.0% | 60.6% |
| DS-Coder-Instruct | 33B | 33B | 47.3% | 50.6% |
| Llama3-Instruct | 70B | 70B | 61.1% | 64.3% |
| DS-Coder-V2-Lite-Instruct | 16B | 2.4B | 53.0% | 52.9% |
| DS-Coder-V2-Instruct | 236B | 21B | **70.0%** | **75.1%** |

**Benchmarks:**

- CruxEval
  - Used to assess code reasoning capabilities of language models
  - Contains 800 Python functions with corresponding input-output examples
  - Evaluates models on both forward and reverse reasoning tasks
    - CRUXEval-I: Predicts output from a given input
    - CRUXEval-O: Predicts input from a known output

**DeepSeek-Coder-V2-Instruct Performance:**

- Best-performing open-source model
- Lags behind larger closed-source models as it is limited by its 21 billion activation parameters

# Results - Mathematical Reasoning

| Model | #TP | #AP | GSM8K | MATH | AIME 2024 | Math Odyssey |
|---|---|---|---|---|---|---|
| Closed-Source Models | | | | | | |
| Gemini 1.5 Pro | - | - | 90.8% | 67.7% | 2/30 | 45.0% |
| Claude-3-Opus | - | - | 95.0% | 60.1% | 2/30 | 40.6% |
| GPT-4-1106 | - | - | 91.4% | 64.3% | 1/30 | 49.1% |
| GPT-4-Turbo-0409 | - | - | 93.7% | 73.4% | **3/30** | 46.8% |
| GPT-4o-0513 | - | - | **95.8%** | **76.6%** | 2/30 | **53.2%** |
| Open-Source Models | | | | | | |
| Llama3-Instruct | 70B | 70B | 93.0% | 50.4% | 1/30 | 27.9% |
| DS-Coder-V2-Lite-Instruct | 16B | 2.4B | 86.4% | 61.8% | 0/30 | 44.4% |
| DS-Coder-V2-Instruct | 236B | 21B | **94.9%** | **75.7%** | **4/30** | **53.7%** |

**Benchmarks:**

- GSM8K
- MATH
- AIME 2024
- Math Odyssey

**DeepSeek-Coder-V2-Instruct Performance:**

- Outperforms open source models
- Results are comparable to state-of-the-art closed source models such as GPT-4o

# Results - General Natural Language

**DeepSeek-Coder-V2-Lite-Instruct Performance**

- Outperforms DeepSeek-V2-Lite-Chat in BBH and Arena-Hard
- Falls behind in knowledge-intensive benchmarks like TriviaQA due to smaller amount of web data used in pre-training

**DeepSeek-Coder-V2-Instruct Performance**

- Significantly stronger performance in Arena-Hard
- Slightly better performance in MT-Bench, AlpacaEval 2.0, and AlignBench

| | Benchmark (Metric) | # Shots | DeepSeek-V2-Lite Chat | DeepSeek-Coder-V2-Lite Instruct | DeepSeek-V2 Chat | DeepSeek-Coder-V2 Instruct |
|---|---|---|---|---|---|---|
| | # Active Params | - | 2.4B | 2.4B | 21B | 21B |
| | # Total Params | - | 16B | 16B | 236B | 236B |
| | # Training Tokens | - | 5.7T | 10.2T | 8.1T | 10.2T |
| English | BBH (EM) | 3-shot | 48.1 | **61.2** | 79.7 | **83.9** |
| | MMLU (Acc.) | 5-shot | 55.7 | **60.1** | 78.1 | **79.2** |
| | ARC-Easy (Acc.) | 25-shot | 86.1 | **88.9** | **98.1** | 97.4 |
| | ARC-Challenge (Acc.) | 25-shot | 73.4 | **77.4** | 92.3 | **92.8** |
| | TriviaQA (EM) | 5-shot | **65.2** | 59.5 | **86.7** | 82.3 |
| | NaturalQuestions (EM) | 5-shot | **35.5** | 30.8 | **53.4** | 47.5 |
| | AGIEval (Acc.) | 0-shot | **42.8** | 28.7 | **61.4** | 60.0 |
| Chinese | CLUEWSC (EM) | 5-shot | **80.0** | 76.5 | **89.9** | 85.9 |
| | C-Eval (Acc.) | 5-shot | 60.1 | **61.6** | 78.0 | **79.4** |
| | CMMLU (Acc.) | 5-shot | 62.5 | **62.7** | **81.6** | 80.9 |
| Open-ended | Arena-Hard | - | 11.40 | **38.10** | 41.60 | **65.00** |
| | AlpacaEval 2.0 | - | 16.85 | **17.74** | **38.90** | 36.92 |
| | MT-Bench | - | 7.37 | **7.81** | **8.97** | 8.77 |
| | Alignbench | - | 6.02 | **6.83** | **7.91** | 7.84 |

**Benchmarks:**

- Evaluated on standard benchmarks covering both English and Chinese datasets:
  - BigBench Hard (BBH)
  - MMLU
  - ARC
  - TriviaQA
  - NaturalQuestions
  - AGIEval.
  - CLUEWSC
  - C-Eval
  - CMMLU
- Evaluation of Open-Ended Generation Ability:
  - Arena-Hard
  - AlpacaEval2.0
  - MT-Bench
  - Alignbench

# Conclusion

- Introduction of DeepSeek-Coder-V2:
    - Continually pre-trained from DeepSeek-V2 using 6 trillion tokens from a high-quality, multi-source corpus
    - Enhances capabilities in coding and mathematical reasoning while maintaining comparable general language performance to DeepSeek-V2
- Key Improvements Over DeepSeek-Coder (V1):
    - Supports more programming languages: Increased from 86 to 338 languages
    - Extended maximum context length: From 16K to 128K tokens
    - Achieves performance comparable to state-of-the-art closed-source models such as GPT-4 Turbo, Claude 3 Opus, and Gemini 1.5 Pro in code and math-specific task
- Limitations and Areas for Improvement:
    - Significant gap in instruction-following capabilities compared to models like GPT-4 Turbo leading to poor performance in complex scenarios such as SWEbench
    - Real-world programming requires both strong coding abilities and exceptional instruction-following skills
- Future Focus:
    - Enhancing instruction-following capabilities
    - Improving performance in real-world complex programming tasks

# Paper Reference

**DeepSeek-AI, Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, et al.** "DeepSeek-Coder-V2: Breaking the barrier of closed-source models in code intelligence," *arXiv preprint*, arXiv:2406.11931, 2024. Available: https://arxiv.org/abs/2406.11931.

# Questions?

# Language Models for Code Optimization: Survey, Challenges, and Future Directions

By: Mihika Rao, Nina Chinnam, Anisha Patrikar

# Presentation Outline

- Introduction
- Background on Code Optimization
- Role of Language Models (LMs) in Code Optimization
- Current Challenges in LM-Based Code Optimization
- Key Findings from the Survey
- Research Questions and Specific Insights
- Techniques to Address These Challenges
- Future Research Directions
- Conclusion

# Mihika Rao (xsw5kn)

# Presentation Outline

- Introduction
- Background on Code Optimization
- Role of Language Models (LMs) in Code Optimization
- Current Challenges in LM-Based Code Optimization
- Research Questions
- Key Insights
- Techniques to Address These Challenges
- Future Research Directions
- Conclusion

# Introduction

- Code Optimization -> Improving code efficiency, speed, and memory usage;
- Ex: reducing execution time, improving energy efficiency
- Importance:
  - Faster and more efficient programs
  - Critical for large-scale applications
- Role of AI in Code Optimization:
  - Automating tedious optimization tasks
  - Enhancing traditional compiler techniques
  - Using Language Models to predict optimized code structures

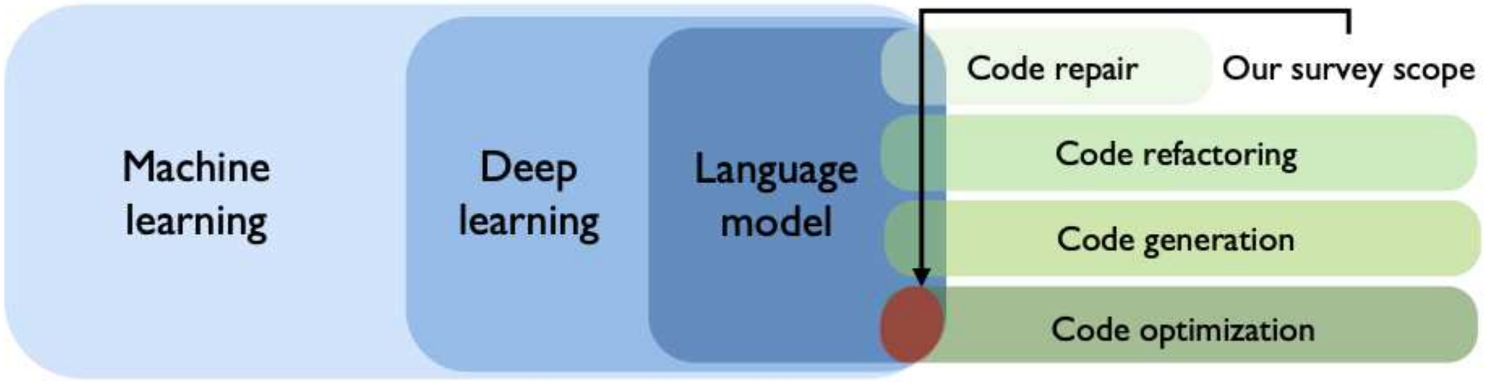



Fig. 1. Visualization of the survey scope.

Scope of surveyed LM-based code optimization methods, highlighting key areas such as code repair, refactoring, generation, and optimization

```python
total = 0
for i in range(1, n+1):
    total += i
# Time complexity of O(n)
```

(a) Unoptimized Python code

```python
total = n * (n + 1) // 2
# Time complexity of O(1)
```

(b) An optimized version of 2a

Fig. 2. Two Python implementations for calculating the sum of the first $n$ natural numbers.

# Background on Code Optimization

## Traditional Code Optimization Techniques:

Manual optimization by developers

Compiler-based optimization (Ie.g., loop unrolling, register allocation)

ML based optimization techniques

## Challenges in Traditional Code Optimization:

Requires domain experience

Not always generalizable across different architectures

Time-consuming and often limited in scalability

# Background on Code Optimization

- Using AI and LMs in Optimization:
  - Can analyze large codebases efficiently
  - Automates tedious optimization tasks
  - Enables cross-platform optimizations

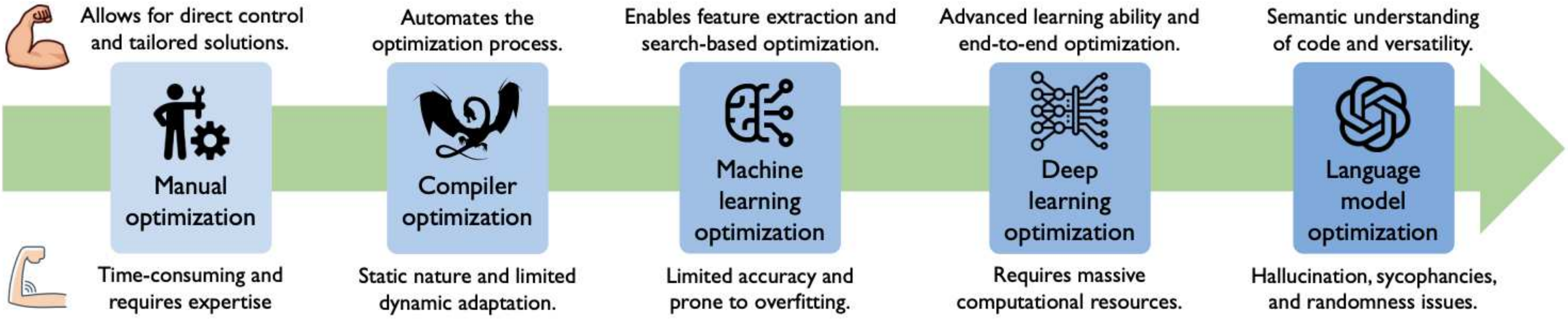
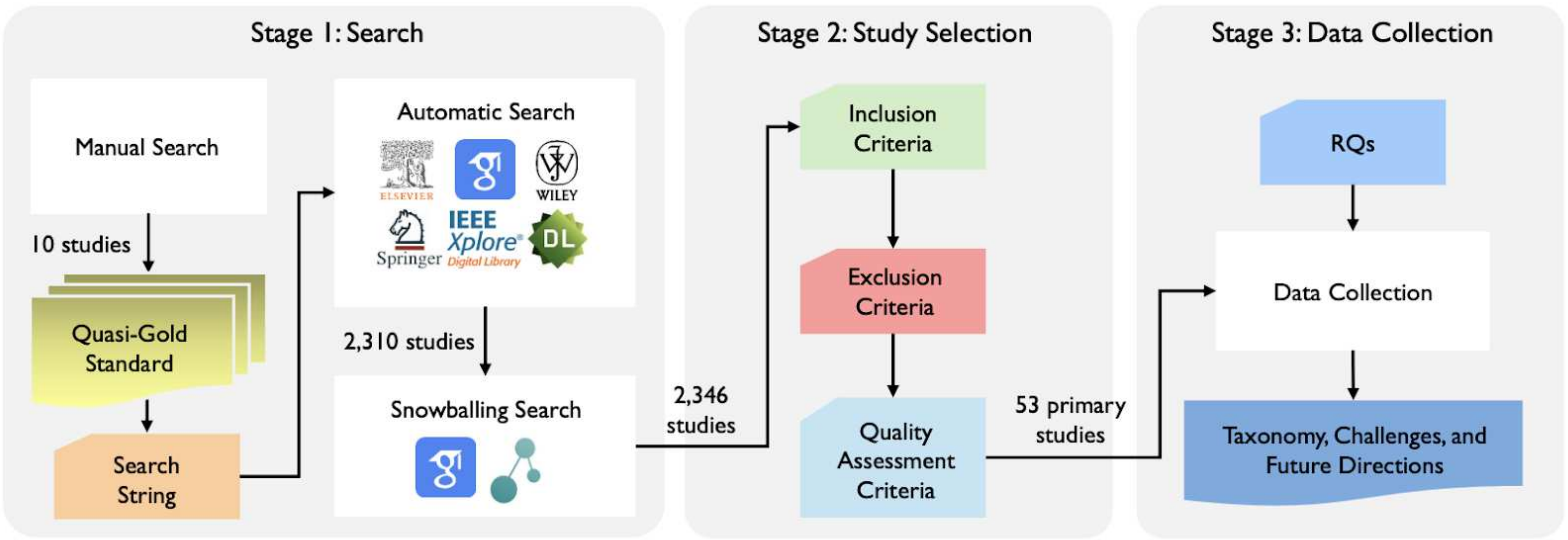



Fig. 3. Development of code optimization methods: strengths and weaknesses

# Survey Methodology



# Role of Language Models in Code Optimization

## How LMs Enhance Optimization:

- Understands complex code patterns and structures
- Automates repetitive and computationally expensive optimization tasks
- Adapts to different programming languages and styles.

## Types of Language Models Used:

- General-purpose LMs (e.g., GPT-4, LLaMA, Claude)
- Code-Specialized LMs (e.g., Code LLaMA, StarCoder, Codex)

# Role of Language Models in Code Optimization

| Category | Total # | LM | Parameter size | Open source | Release year | Description | # | Used studies |
|---|---|---|---|---|---|---|---|---|
| General-purpose LMs | 61 | GPT-4 [96] | ≈1.8T | ✗ | 2024 | The vast parameter size and extensive training data enables its improved reasoning abilities and the ability to process more complex instructions. | 15 | [38, 52, 58, 84, 105, 110, 116, 118, 119, 123, 124, 144, 148, 151, 154] |
| | | GPT-3.5-turbo [95] | ≈175B | ✗ | 2022 | Faster response times and more cost-efficient compared to GPT-3.5. | 9 | [24, 40, 54, 58, 62, 84, 117, 129, 144] |
| | | GPT 3.5 [95] | ≈175B | ✗ | 2022 | An earlier version of GPT-4, known for its solid capability in understanding and generating human-like text and code. | 9 | [38, 84, 98, 101, 105, 110, 118, 119, 142] |
| | | GPT-4o [97] | ≈1.8T | ✗ | 2024 | A multi-modal version of GPT-4 that can handle multimodal code contexts. | 7 | [104, 129, 130, 133, 138, 150, 154] |
| | | GPT-4-turbo [96] | ≈1.8T | ✗ | 2024 | Combines the strengths of GPT-4 with improved efficiency for faster processing. | 4 | [56, 58, 129, 147] |
| | | LLaMA-2 [126] | 7B, 13B, 34B | ✓ | 2023 | Enhanced capabilities and efficiency over LLaMA-1. | 4 | [27, 47, 48, 73] |
| | | Claude-3-haiku [6] | ≈20B | ✗ | 2024 | Fastest among the Claude-3 models, optimized for near-instant responsiveness. | 2 | [53, 58] |
| | | Gemini-Pro [4] | ≈540B | ✗ | 2023 | Google's multimodal model, like GPT-4o, leveraging the MoE architecture. | 2 | [38, 91] |
| | | LLaMA-3.1 [88] | 8B | ✓ | 2024 | Improves over LLaMA-2 with expanded context length and multilingual support. | 2 | [105, 154] |
| | | Claude-3-sonnet [6] | ≈70B | ✗ | 2024 | Larger than Claude-3-haiku, providing stronger performance and precision. | 1 | [58] |
| | | LLaMA-1 [125] | 7B, 13B, 34B | ✓ | 2023 | An open-source LM that can be fine-tuned for code optimization. | 1 | [73] |
| | | PaLM-2 [5] | 340B | ✗ | 2023 | Excels at solving complex tasks by decomposing them into simpler subtasks. | 1 | [144] |
| | | Phi-2 [64] | 2.7B | ✓ | 2023 | Achieves remarkable performance despite its relatively compact size. | 1 | [153] |
| | | BLOOM [13] | 3B, 7B | ✓ | 2022 | A multilingual language model designed for general text processing. | 1 | [73] |
| | | GPT-NeoX [15] | 20B | ✓ | 2022 | Provides accurate and contextually relevant responses for text processing tasks. | 1 | [100] |
| | | GPT-3 [16] | ≈175B | ✗ | 2020 | Ealier version of GPT-3.5, known for its general NLP abilities. | 1 | [63] |
| Code-specialized LMs | 43 | Code LLaMA [114] | 7B, 13B, 34B, 70B | ✓ | 2023 | A LLaMA model fine-tuned for strong code-related performance, benefiting from the efficiency and architecture of LLaMA. | 11 | [28, 38, 41, 58, 73, 108, 119, 133, 142, 145, 149] |
| | | DeepSeekCoder [31] | 1.3B, 6.7B, 33B | ✓ | 2023 | Shows competitive performance in coding tasks due to its incorporation of semantic search and retrieval mechanisms. | 7 | [58, 59, 91, 110, 133, 149, 153] |
| | | StarCoder [74] | 1B, 3B, 7B, 15B | ✓ | 2023 | Trained on a massive dataset of permissively licensed source code, making it more readily usable in commercial applications. | 4 | [41, 58, 112, 133] |
| | | CodeT5 [135] | 60M, 220M, 770M | ✓ | 2021 | T5 model fine-tuned for coding tasks, offering a balance of general language understanding and code specialization. | 4 | [32, 77, 101, 148] |
| | | WizardCoder [83] | 13B | ✓ | 2024 | Improved coding capabilities due to the Evol-Instruct training method. | 3 | [58, 118, 133] |
| | | Qwen2.5-Code [61] | 7B | ✓ | 2024 | Provides advanced coding assistance and improves productivity for developers. | 2 | [59, 145] |
| | | CodeX [94] | 12B | ✓ | 2021 | A powerful coding assistant that is integrated with GitHub Copilot. | 2 | [63, 84] |
| | | StarCoder2 [81] | 7B | ✓ | 2024 | Trained on significantly larger and more diverse coding data than StarCoder. | 1 | [153] |
| | | CodeGemma [87] | 7B | ✓ | 2024 | Optimized for coding tasks using pre-trained Gemma models. | 1 | [133] |
| | | OpenCodeInterpreter [158] | 1.3B, 6.7B, 33B | ✓ | 2024 | Combines a language model with a code execution environment, allowing it to optimize code by directly evaluating its performance. | 1 | [58] |
| | | Codey [46] | 340B | ✓ | 2023 | Provides code suggestions, completions, and refactoring assistance. | 1 | [112] |
| | | XwinCoder [90] | 7B, 13B, 34B | ✓ | 2023 | Focuses on cross-lingual code understanding and generation. | 1 | [58] |
| | | CodeGen-mono [92] | 350M | ✓ | 2023 | Achieves superior coding accuracy by focusing exclusively on one language | 1 | [101] |
| | | PolyCoder [141] | 400M | ✓ | 2022 | Emphasizing multilingual programming capabilities | 1 | [101] |
| | | CodeBERT [35] | 125M | ✓ | 2020 | Leverages BERT architecture for better understanding of code semantics. | 1 | [23] |
| | | PyMT5 [25] | 374M | ✓ | 2020 | Optimized for Python code, providing targeted code improvements. | 1 | [39] |
| | | TransCoder [115] | ≈60M | ✓ | 2020 | Specialized in translating code between programming languages. | 1 | [50] |
| Trans-formers | 2 | Bert-tiny [128] | 4.4M | ✓ | 2019 | A smaller version of BERT, suitable for scenarios requiring fast response times. | 1 | [100] |
| | | Transformer [131] | ≈30M | ✓ | 2017 | The foundational architecture for many LMs. | 1 | [120] |

# Role of Language Models in Code Optimization

## Common Applications of LMs in Code Optimization:

- Code generation and transformation
- Automated bug fixing and performance tuning
- Assisting compiler optimizations through learned heuristics

# Challenges in LM-Based Code Optimization: Performance

LMs need significant computational resources for training and inference

Trade-offs between optimization accuracy and execution time

Difficulty in balancing correctness and efficiency improvements

# Challenges in LM-Based Code Optimization: Code

Handling different programming languages

Adapting to dynamic and evolving codebases

Ensuring code readability

# Challenges in LM-Based Code Optimization: Dataset and Training

Need for diverse and high-quality datasets to train LMs

Overfitting to specific coding styles or patterns

Lack of standardized benchmarks for evaluating LM-based code optimizations

# Nina Chinnam (fhs9af)

# Presentation Outline

- Introduction
- Background on Code Optimization
- Role of Language Models (LMs) in Code Optimization
- Current Challenges in LM-Based Code Optimization
- Research Questions
- Key Insights
- Techniques to Address These Challenges
- Future Research Directions
- Conclusion

# Research Questions (RQ) and Key Insights

# RQ1: What were the characteristics of the LMs used for Code Optimization?

# RQ1: Types of LMs that are used

Table 1. Distribution of LMs used for code optimization (one study might be in multiple categories).

| Category | Total # | LM | Parameter size | Open source | Release year | Description | # | Used studies |
|---|---|---|---|---|---|---|---|---|
| General-purpose LMs | 61 | GPT-4 [96] | ≈1.8T | ✗ | 2024 | The vast parameter size and extensive training data enables its improved reasoning abilities and the ability to process more complex instructions. | 15 | [38, 52, 58, 84, 105, 110, 116, 118, 119, 123, 124, 144, 148, 151, 154] |
| | | GPT-3.5-turbo [95] | ≈175B | ✗ | 2022 | Faster response times and more cost-efficient compared to GPT-3.5. | 9 | [24, 40, 54, 58, 62, 84, 117, 129, 144] |
| | | GPT 3.5 [95] | ≈175B | ✗ | 2022 | An earlier version of GPT-4, known for its solid capability in understanding and generating human-like text and code. | 9 | [38, 84, 98, 101, 105, 110, 118, 119, 142] |
| | | GPT-4o [97] | ≈1.8T | ✗ | 2024 | A multi-modal version of GPT-4 that can handle multimodal code contexts. | 7 | [104, 129, 130, 133, 138, 150, 154] |
| | | GPT-4-turbo [96] | ≈1.8T | ✗ | 2024 | Combines the strengths of GPT-4 with improved efficiency for faster processing. | 4 | [56, 58, 129, 147] |
| | | LLaMA-2 [126] | 7B, 13B, 34B | ✓ | 2023 | Enhanced capabilities and efficiency over LLaMA-1. | 4 | [27, 47, 48, 73] |
| | | Claude-3-haiku [6] | ≈20B | ✗ | 2024 | Fastest among the Claude-3 models, optimized for near-instant responsiveness. | 2 | [53, 58] |
| | | Gemini-Pro [4] | ≈540B | ✗ | 2023 | Google's multimodal model, like GPT-4o, leveraging the MoE architecture. | 2 | [38, 91] |
| | | LLaMA-3.1 [88] | 8B | ✓ | 2024 | Improves over LLaMA-2 with expanded context length and multilingual support. | 2 | [105, 154] |
| | | Claude-3-sonnet [6] | ≈70B | ✗ | 2024 | Larger than Claude-3-haiku, providing stronger performance and precision. | 1 | [58] |
| | | LLaMA-1 [125] | 7B, 13B, 34B | ✓ | 2023 | An open-source LM that can be fine-tuned for code optimization. | 1 | [73] |
| | | PaLM-2 [5] | 340B | ✗ | 2023 | Excels at solving complex tasks by decomposing them into simpler subtasks. | 1 | [144] |
| | | Phi-2 [64] | 2.7B | ✓ | 2023 | Achieves remarkable performance despite its relatively compact size. | 1 | [153] |
| | | BLOOM [13] | 3B, 7B | ✓ | 2022 | A multilingual language model designed for general text processing. | 1 | [73] |
| | | GPT-NeoX [15] | 20B | ✓ | 2022 | Provides accurate and contextually relevant responses for text processing tasks. | 1 | [100] |
| | | GPT-3 [16] | ≈175B | ✗ | 2020 | Ealier version of GPT-3.5, known for its general NLP abilities. | 1 | [63] |
| Code-specialized LMs | 43 | Code LLaMA [114] | 7B, 13B, 34B, 70B | ✓ | 2023 | A LLaMA model fine-tuned for strong code-related performance, benefiting from the efficiency and architecture of LLaMA. | 11 | [28, 38, 41, 58, 73, 108, 119, 133, 142, 145, 149] |
| | | DeepSeekCoder [31] | 1.3B, 6.7B, 33B | ✓ | 2023 | Shows competitive performance in coding tasks due to its incorporation of semantic search and retrieval mechanisms. | 7 | [58, 59, 91, 110, 133, 149, 153] |
| | | StarCoder [74] | 1B, 3B, 7B, 15B | ✓ | 2023 | Trained on a massive dataset of permissively licensed source code, making it more readily usable in commercial applications. | 4 | [41, 58, 112, 133] |
| | | CodeT5 [135] | 60M, 220M, 770M | ✓ | 2021 | T5 model fine-tuned for coding tasks, offering a balance of general language understanding and code specialization. | 4 | [32, 77, 101, 148] |
| | | WizardCoder [83] | 13B | ✓ | 2024 | Improved coding capabilities due to the Evol-Instruct training method. | 3 | [58, 118, 133] |
| | | Qwen2.5-Code [61] | 7B | ✓ | 2024 | Provides advanced coding assistance and improves productivity for developers. | 2 | [59, 145] |
| | | CodeX [94] | 12B | ✓ | 2021 | A powerful coding assistant that is integrated with GitHub Copilot. | 2 | [63, 84] |
| | | StarCoder2 [81] | 7B | ✓ | 2024 | Trained on significantly larger and more diverse coding data than StarCoder. | 1 | [153] |
| | | CodeGemma [87] | 7B | ✓ | 2024 | Optimized for coding tasks using pre-trained Gemma models. | 1 | [133] |
| | | OpenCodeInterpreter [158] | 1.3B, 6.7B, 33B | ✓ | 2024 | Combines a language model with a code execution environment, allowing it to optimize code by directly evaluating its performance. | 1 | [58] |
| | | Codey [46] | 340B | ✓ | 2023 | Provides code suggestions, completions, and refactoring assistance. | 1 | [112] |
| | | XwinCoder [90] | 7B, 13B, 34B | ✓ | 2023 | Focuses on cross-lingual code understanding and generation. | 1 | [58] |
| | | CodeGen-mono [92] | 350M | ✓ | 2023 | Achieves superior coding accuracy by focusing exclusively on one language | 1 | [101] |
| | | PolyCoder [141] | 400M | ✓ | 2022 | Emphasizing multilingual programming capabilities | 1 | [101] |
| | | CodeBERT [35] | 125M | ✓ | 2020 | Leverages BERT architecture for better understanding of code semantics. | 1 | [23] |
| | | PyMT5 [25] | 374M | ✓ | 2020 | Optimized for Python code, providing targeted code improvements. | 1 | [39] |
| | | TransCoder [115] | ≈60M | ✓ | 2020 | Specialized in translating code between programming languages. | 1 | [50] |
| Transformers | 2 | Bert-tiny [128] | 4.4M | ✓ | 2019 | A smaller version of BERT, suitable for scenarios requiring fast response times. | 1 | [100] |
| | | Transformer [131] | ≈30M | ✓ | 2017 | The foundational architecture for many LMs. | 1 | [120] |

# RQ1: Sizes of LMs that were used

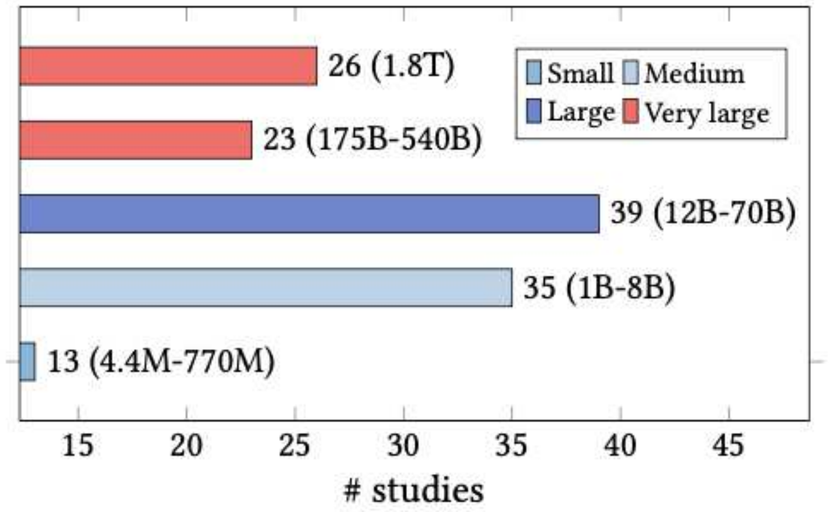



Fig. 6. Distribution of parameter sizes (one study might be in multiple categories).

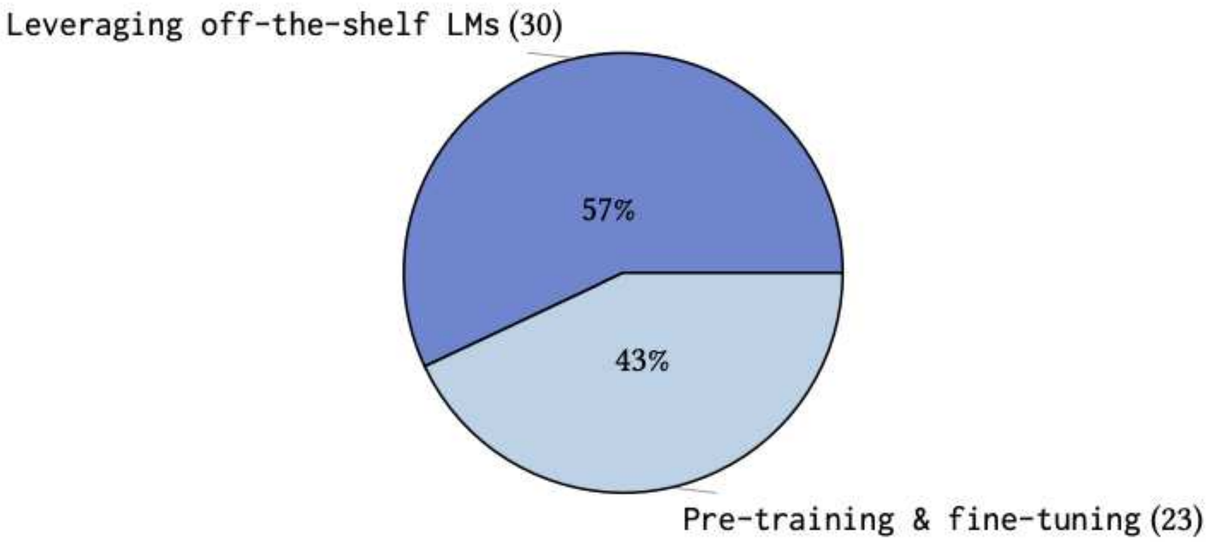



Fig. 7. Distribution of training the LMs.

# RQ2: How were LMs applied to Code Optimization Tasks?

# RQ2: Common Challenges

Table 2. Distribution of addressed challenges (one study might be in multiple categories).

| Category | Total # | Challenge | # studies | Reference |
|---|---|---|---|---|
| Performance | 49 | Limitation of one-step generation | 18 | [32, 53, 54, 58, 62, 77, 84, 101, 105, 108, 110, 117, 120, 124, 142, 145, 150, 154] |
| | | Balancing correctness and performance | 15 | [41, 58, 59, 91, 98, 100, 101, 104, 108, 124, 129, 130, 133, 149, 153] |
| | | Reliance on human experts | 13 | [23, 24, 39, 40, 53, 56, 98, 123, 129, 142, 145, 147, 151] |
| | | Poor code maintainability | 2 | [52, 118] |
| | | Hardware-dependent performance variability | 1 | [119] |
| Code | 24 | Complexity of code | 10 | [28, 50, 84, 112, 117, 124, 138, 144, 147, 150] |
| | | Limitation on localized code modifications | 4 | [27, 38, 100, 119] |
| | | Incomplete code representation | 4 | [28, 47, 63, 77] |
| | | Limited exploration of low-level languages | 3 | [28, 50, 138] |
| | | Limited applicability to real-world code | 2 | [24, 120] |
| | | Limited representation of problems | 1 | [110] |
| Dataset | 18 | Limited efficiency-related datasets | 9 | [32, 39, 41, 59, 91, 101, 119, 133, 149] |
| | | Reliance on manually labeled data | 3 | [84, 116, 153] |
| | | Limited low-level language datasets | 2 | [28, 56] |
| | | Limited real-world datasets | 1 | [120] |
| | | Limited code maintainability datasets | 1 | [118] |
| | | Limited code editing datasets | 1 | [73] |
| | | Limited type inference datasets | 1 | [148] |
| LM | 15 | Limited generalizability across domains | 3 | [23, 40, 56] |
| | | Inefficiency of querying LMs | 2 | [145, 151] |
| | | Limitation of sampling methods | 2 | [48, 110] |
| | | High cost of fine-tuning | 2 | [32, 40] |
| | | Hallucination Issues of LMs | 2 | [105, 123] |
| | | Sycophancies of LMs | 1 | [105] |
| | | Inherent randomness of LMs | 1 | [147] |
| | | Handling multiple types of inputs | 1 | [100] |
| | | Limited exploration of solution space | 1 | [112] |
| Compiler | 3 | Limited optimization ability of compilers | 3 | [23, 27, 147] |

# RQ2: Addressing Challenges with LMs

Table 3. Distribution of code optimization techniques (one study might be in multiple categories).

| Category | Total # | Technique | # studies | Reference | Addressed challenge (# studies) |
|---|---|---|---|---|---|
| Model-based | 51 | Feedback-based iterative optimization | 35 | [24, 32, 38, 41, 47, 52–54, 56, 58, 59, 62, 63, 77, 84, 91, 98, 104, 108, 110, 112, 116, 117, 124, 129, 130, 133, 138, 144, 145, 147, 150, 151, 153, 154] | Limitation of one-step optimization (14), Balancing correctness and performance (12), Complexity of code (8), Reliance on human experts (8), Limited efficiency-related datasets (5), Reliance on manually labeled data (4), Inefficiency of querying LMs (3), Incomplete code representation (3), Hallucination Issues of LMs (2), High cost of fine-tuning (1), Inherent randomness of LMs (1), Limited generalizability across domains (1), Limited exploration of solution space (1) |
| | | Agentic workflow | 6 | [104, 116, 123, 124, 138, 154] | Balancing correctness and performance (2), Complexity of code (2), Reliance on human experts (1), Reliance on manually labeled data (1), Hallucination Issues of LMs (1) |
| | | Compiler emulation | 4 | [27, 28, 47, 50] | Complexity of code (2), Incomplete code representation (2), Limited exploration of low-level languages (2), Limited low-level language datasets (1), Limitation on localized code modifications (1), Limited optimization ability of compilers (1) |
| | | Direct preference optimization | 3 | [41, 91, 153] | Balancing correctness and performance (3), Limited efficiency-related datasets (2), Reliance on manually labeled data (1) |
| | | Compiler passes sampling | 1 | [48] | Limitation of sampling methods (1) |
| | | Ensemble learning | 1 | [149] | Balancing correctness and performance (1), Limited efficiency-related datasets (1) |
| | | Encoder-decoder | 1 | [100] | Limitation on localized code modifications (1), Handling multiple types of inputs (1) |
| Prompt engineering | 34 | Few-shot prompting | 11 | [54, 73, 84, 112, 116, 117, 119, 130, 133, 142, 153] | Limitation of one-step optimization (3), Complexity of code (3), Reliance on manually labeled data (3), Balancing correctness and performance (3), Limited efficiency-related datasets (2), Reliance on human experts (1), Inefficiency of querying LMs (1) |
| | | Contextual prompting | 9 | [56, 58, 63, 77, 105, 118, 138, 144, 148] | Limitation of one-step optimization (2), Complexity of code (2), Incomplete code representation (2), Poor code maintainability (1), Limited generalizability across domains (1) |
| | | Chain-of-thought | 8 | [38, 62, 116, 119, 123, 145, 149, 150] | Limitation of one-step optimization (3), Limitation on localized code modifications (2), Reliance on human experts (2), Balancing correctness and performance (1), Complexity of code (1), Reliance on manually labeled data (1), Inefficiency of querying LMs (1), Hallucination Issues of LMs (1) |
| | | Retrieval-augmented generation | 5 | [38, 40, 119, 142, 147] | Limitation on localized code modifications (2), Reliance on human experts (2), Limitation of one-step optimization (1), Hardware-dependent performance variability (1), High cost of fine-tuning (1), Limited generalizability across domains (1) |
| | | Scaffolding optimization | 1 | [151] | Inefficiency of querying LMs (1), Reliance on human experts (1) |
| Problem formulation | 33 | Dataset | 19 | [23, 27, 28, 39, 41, 47, 48, 59, 73, 91, 101, 118–120, 133, 145, 148, 149, 153] | Limited efficiency-related datasets (8), Balancing correctness and performance (7), Limitation of one-step optimization (2), Limitation on localized code modifications (2), Reliance on human experts (2), Incomplete code representation (2), Limited low-level language datasets (1), Limited real-world datasets (1), Limited code maintainability datasets (1), Limited code editing datasets (1), Limited type inference datasets (1), Complexity of code (1), Reliance on manually labeled data (1) |
| | | Reinforcement learning | 6 | [32, 53, 62, 77, 91, 116] | Limitation of one-step optimization (3), Limited efficiency-related datasets (2), Balancing correctness and performance (1), Reliance on manually labeled data (1), Incomplete code representation (1), High cost of fine-tuning (1) |
| | | Search-based | 4 | [38, 54, 112, 120] | Limitation of one-step optimization (1), Complexity of code (1), Limitation on localized code modifications (1), Limited exploration of solution space (1) |
| | | Code token tree | 1 | [108] | Limitation of one-step optimization (1), Balancing correctness and performance (1) |
| | | Modular generation | 1 | [144] | Complexity of code (1) |
| | | Metric design | 1 | [101] | Limitation of one-step optimization (1), Balancing correctness and performance (1) |
| | | Diff synthesis | 1 | [23] | Reliance on human experts (1), Limited generalizability across domains (1) |

# RQ2: Addressing Challenges with LMs

**Model-based approaches**

- Feedback-based iterative optimization
- Agentic workflows for self improvement
- Compiler emulation (LMs acting like compilers)

**Prompt engineering techniques**

- Few-shot prompting
- Chain-of-thought (CoT) (step by step reasoning)
- Retrieval augmented generation (RAG)

**New problem formations**

- Reinforcement Learning for iterative optimization
- Search-based techniques

# RQ2: Roles of LMs

Table 4. Distribution of roles of LMs (one study might be in multiple categories).

| Category | Total # | Role | # studies | Reference |
|---|---|---|---|---|
| Generation | 73 | Optimizer | 46 | [24, 32, 38–41, 48, 52–54, 56, 58, 59, 62, 63, 73, 77, 84, 91, 98, 101, 104, 105, 108, 110, 112, 116–120, 123, 124, 129, 130, 133, 138, 142, 144, 145, 147, 149–151, 153, 154] |
| | | Generator | 21 | [24, 41, 53, 54, 58, 63, 77, 84, 98, 105, 108, 110, 112, 116, 117, 123, 129, 130, 144, 150, 154] |
| | | Compiler | 4 | [27, 28, 47, 50] |
| | | Decoder | 1 | [100] |
| | | Diff generator | 1 | [23] |
| Evaluation | 10 | Evaluator | 10 | [54, 84, 100, 104, 116, 123, 124, 145, 150, 154] |
| Preprocessing | 6 | Advisor | 2 | [123, 124] |
| | | Encoder | 2 | [100, 142] |
| | | Type inferencer | 2 | [52, 148] |

# RQ2: Roles of LMs

## Generation

- **Optimizer (46 studies)**
- **Generator (21 studies)**
- **Compiler Emulator (4 studies)**
- **Code Diff Generator (1 study)**
- **Decoder Role (1 study)**

## Evaluation

- Evaluate correctness, performance, and quality
- Bug identification, validation, compliance checking
- Faster than compilers, but hallucination issues

## Preprocessing

- Advisor Role
- Encoder Role
- Type Inferencer

# RQ3: How was the Code Optimization Problem defined?

# RQ3: Programming Languages

Table 5. Distribution of optimized languages (one study might be in multiple categories).

| Category | Total # | Languague | # studies | Reference |
|---|---|---|---|---|
| High-level languages | 53 | Python | 30 | [32, 38, 41, 53, 54, 58, 59, 62, 63, 73, 77, 84, 91, 100, 101, 105, 108, 110, 112, 116–118, 129, 130, 133, 144, 148, 150, 151, 153] |
| | | C++ | 9 | [23, 38, 91, 98, 104, 110, 119, 145, 149] |
| | | C | 6 | [23, 50, 52, 98, 110, 124] |
| | | Rust | 3 | [110, 116, 150] |
| | | C# | 3 | [39, 40, 110] |
| | | Java | 2 | [24, 91] |
| Low-level languages | 6 | LLVM-IR | 4 | [27, 28, 47, 48] |
| | | Assembly code | 2 | [28, 120] |
| Domain-specific languages | 6 | Tensor processing code | 1 | [56] |
| | | Mapper code | 1 | [138] |
| | | Heuristsic code | 1 | [123] |
| | | High-Level Synthesis (HSL) | 1 | [142] |
| | | Register Transfer Level (RTL) | 1 | [147] |
| | | Structured Text (ST) | 1 | [52] |

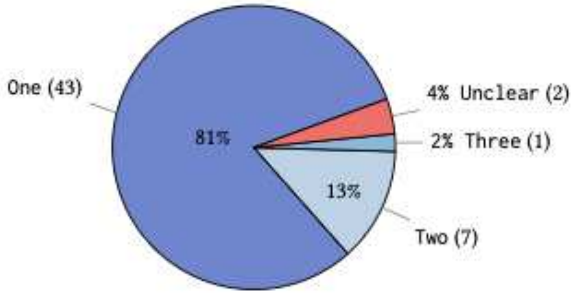



Fig. 8. Distribution of # optimized programming languages.

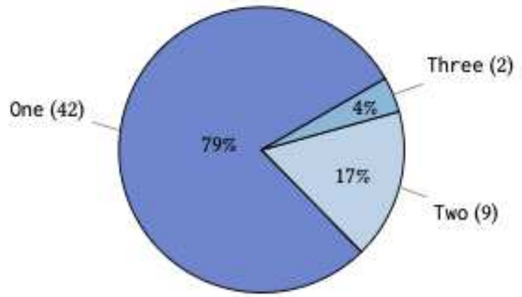



Fig. 9. Distribution of # targeted performance metrics.

# RQ3: Performance Metrics that were Optimized

Table 6. Distribution of targeted performance metrics (one study might be in multiple categories).

| Category | Total # | Metric | # studies | Reference |
|---|---|---|---|---|
| Efficiency | 27 | Runtime | 24 | [23, 32, 38, 41, 54, 58, 59, 84, 91, 98, 100, 101, 105, 108, 110, 119, 120, 124, 133, 145, 149–151, 153] |
| | | Latency | 2 | [104, 142] |
| | | Throughput | 1 | [138] |
| General quality | 16 | Code size | 5 | [27, 28, 41, 47, 48] |
| | | Complexity | 5 | [24, 77, 112, 117, 118] |
| | | Readability | 3 | [52, 77, 118] |
| | | Maintainability | 3 | [52, 77, 118] |
| Task-specific | 14 | Task completion rate | 2 | [144, 154] |
| | | Convergence quality | 2 | [129, 130] |
| | | Synthesis accuracy | 1 | [63] |
| | | Number of instances solved | 1 | [123] |
| | | Success rate | 1 | [53] |
| | | Synthesis performance | 1 | [147] |
| | | Hardware performance | 1 | [56] |
| | | Reference match | 1 | [50] |
| | | Code edit accuracy | 1 | [73] |
| | | Decision-making performance | 1 | [116] |
| | | Driving score | 1 | [62] |
| | | Type inference speed | 1 | [148] |
| Resource usage | 9 | Memory usage | 6 | [23, 39, 58, 59, 110, 133] |
| | | CPU usage | 2 | [39, 40] |
| | | Energy | 1 | [104] |

# RQ4: How were the Proposed Code Optimization Methods Evaluated?

# RQ4: Existing Datasets and Benchmarks

Table 7. Distribution of datasets and benchmarks (one study might be in multiple categories).

Go to page 31

| Category | Total # | Dataset | Source | Size | Languages | Performance | Repo | Reference |
|---|---|---|---|---|---|---|---|---|
| Competitive programming | 35 | HumanEval [21] | Hand-crafted by experts | 164 programming tasks | Python | Correctness | Link | [41, 58, 59, 77, 105, 116, 153] |
| | | MBPP [8] | Programming problems | 974 programming tasks | Python | Correctness | Link | [41, 58, 77, 105, 116, 153] |
| | | PIE [119] | CodeNet | 77K pairs of slow-fast code | C++ | Runtime | Link | [32, 38, 84, 119] |
| | | LeetcodeHardGym [116] | Leetcode | 40 questions | Python, Rust | Runtime | Link | [116, 150, 154] |
| | | EffiBench [60] | Leetcode | 1K efficiency-critical coding problems | Python | Runtime, memory | Link | [58, 59] |
| | | CodeContests [76] | Aizu Online Judge, AtCoder | 13,610 samples | Python, C++, Java | Runtime, memory | Link | [110, 117] |
| | | APPS [55] | Coding websites | 10k coding problems | Python | Correctness | Link | [77, 105] |
| | | ECCO [133] | CodeNet | 50K solution pairs | Python | Runtime, memory | Link | [133] |
| | | FunSearch [112] | Algorithmic problems | $10^6$ samples | Python | Complexity, readability, maintainability | Link | [112] |
| | | Supersonic [23] | CodeNet | 314,435 samples | C, C++ | Runtime, memory | Link | [23] |
| | | GEC [99] | CodeForces | 31,577 pairs of slow-fast code | Python | Runtime | Link | [100] |
| | | CodeNet [107] | AIZU Online Judge, AtCoder | 14 million samples | C++, C, C#, Python, Java... | Runtime, memory, code size | Link | [50] |
| | | ACEOB [101] | CodeForces | 95,359 pairs of efficient-inefficient code | Python | Runtime | ✗ | [101] |
| | | Effi-Code [59] | Coding datasets | 9,451 tasks | Python | Runtime, memory | Link | [59] |
| | | SAPIE [145] | CodeNet | 77k pairs of slow-fast code | C++ | Runtime | ✗ | [145] |
| | | PIE-problem [149] | CodeNet | 18,242 pairs of slow-fast code | C++ | Runtime | ✗ | [149] |
| General SE | 13 | DeepDev-PERF [39] | GitHub | 45k open-source repositories | C# | CPU, memory | ✗ | [39, 40] |
| | | AnghaBench [30] | GitHub | 1 million samples | C | Runtime, code size | Link | [50] |
| | | InstructCoder [73] | GitHub | 114K instruction-input-output triplets | Python | Complexity, readability, maintainability | Link | [73] |
| | | Energy-Language [106] | Software repositories | 10 problems | 27 languages | Energy, memory, runtime | Link | [104] |
| | | BetterPython [118] | CommitPackFT, CodeAlpaca | 34,139 samples | Python | Complexity, readability, maintainability | Link | [118] |
| | | Defects4J [66] | Open-srouce projects | 17 projects | Java | Complexity | Link | [24] |
| | | PP4F [68] | Synthesis | 699 examples | HLS | Latency | Link | [142] |
| | | RewriterBench [147] | Industry cases | 55 cases | RTL | Synthesis performance | Link | [147] |
| | | ST-to-C [52] | Industry cases | 3 case studies | Structured Text (ST), C | Readability, maintainability | ✗ | [52] |
| | | PandasEval [63] | StackOverflow, Hackathon | 89 Pandas tasks | Python | Correctness | Link | [63] |
| | | Big Assembly [120] | GitHub | 25,141 assembly functions | x86-64 assembly language | CPU-clock cycles | ✗ | [120] |
| | | CSmith [146] | Synthesis | Unlimited | C | Runtime | Link | [50] |
| Compiler | 7 | PolyBench [1] | Synthesis | 30 numerical polyhedral kernels | Python, C | Runtime, memory | Link | [56, 91, 148] |
| | | LLM4Compiler [27] | GitHub, synthesis | 1 million functions | LLVM-IR | Code size | ✗ | [27, 47] |
| | | TSVC [85] | Synthesis | 149 test cases | C | Runtime, code size | Link | [124] |
| | | Priority Sampling [48] | GitHub | 50K functions | LLVM-IR | Code size | ✗ | [48] |
| Data science | 2 | Big-DS-1000 [108] | StackOverflow | 1000 data science problems | Python | Runtime | ✗ | [108] |
| | | DS-1000 [70] | StackOverflow | 1000 data science problems | Python | Correctness | Link | [153] |

# RQ4: Data and Metrics for Evaluation

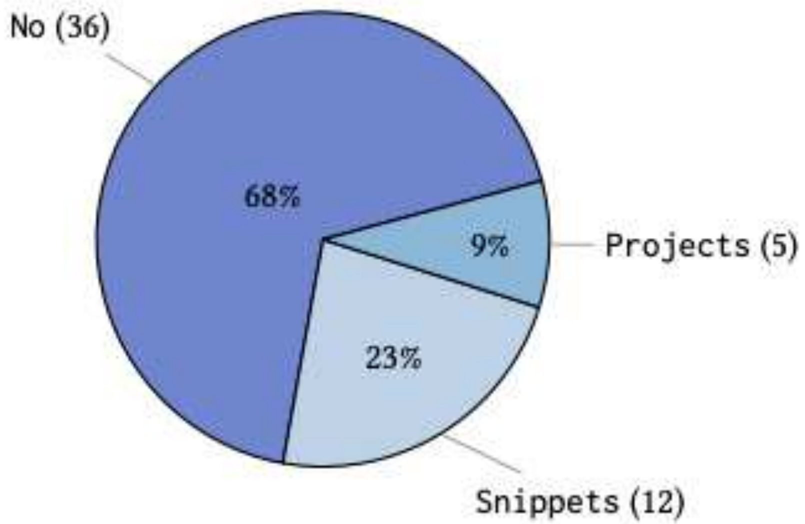



Fig. 10. Distribution of evaluation using real-world code.

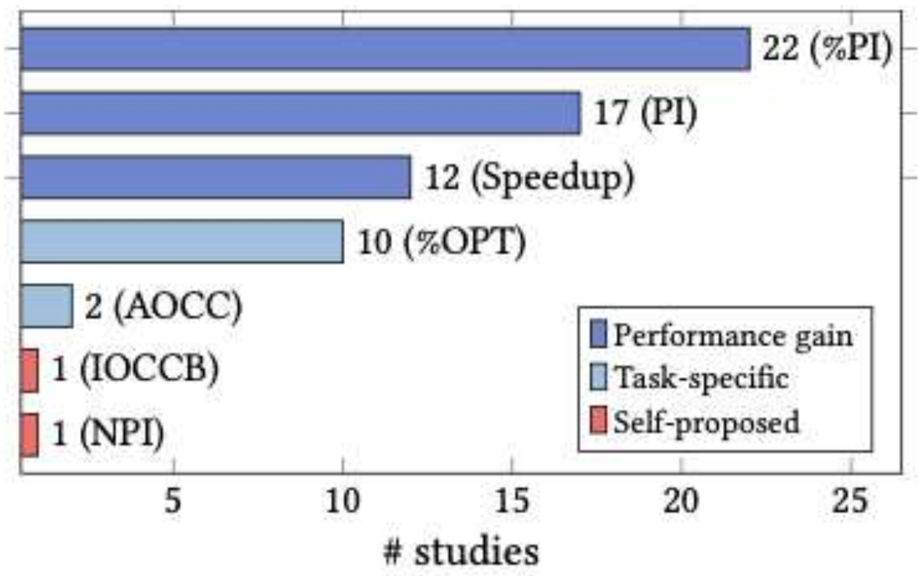



Fig. 11. Distribution of evaluation metrics (one study might be in multiple categories).

# Anisha Patrikar (gjq2yf)

# Presentation Outline

- Introduction
- Background on Code Optimization
- Role of Language Models (LMs) in Code Optimization
- Current Challenges in LM-Based Code Optimization
- Research Questions
- Key Insights
- Techniques to Address These Challenges
- Future Research Directions
- Conclusion

# Challenges and Future Directions

# Challenge 1: Balancing Model Complexity and Practicality


Large models (e.g., GPT-4 with 1.8T parameters) require substantial computational resources

Scaling LMs for real-world, large-scale codebases remains difficult

Trade-off between model size, efficiency, and cost-effectiveness

# Future Directions: Balancing Model Complexity

**MODEL COMPRESSION**

**ENSEMBLING SMALLER MODELS**

# Challenge 2: Limited Interaction with External Systems

LMs currently operate in isolated environments, unlike human programmers

Lack of seamless integration with external tools, IDEs, and expert knowledge

Results in suboptimal code optimization

# Future Directions: Enhancing LM Interaction

## Agentic LMs:

- LMs that can dynamically access external resources and interact with other systems

## Multi-Agent Collaboration:

- Multiple LMs working together, leveraging specialized knowledge

# Challenge 3: Limited Generalizability Across Languages and Metrics

- LMs struggle to optimize across different programming languages and performance metrics
- Syntax, semantics, and execution behavior vary widely
- 81% of research focuses on a single language, limiting real-world applicability

# Future Directions: Improving Generalizability

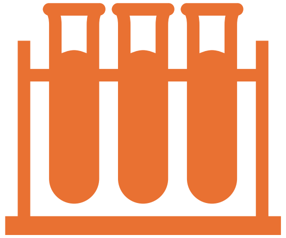
**Cross-Lingual Optimization Models:**

- Adapting multi-lingual LMs for code optimization

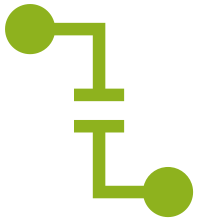
**Multi-Objective Optimization:**

- Balancing multiple performance metrics (runtime, memory, energy consumption)

# Challenge 4: Limited Evaluation on Real-World Code



Only 32% of studies tested on real-world datasets



Optimizations degrade when applied to complex, legacy, or undocumented codebases

Need for more practical testing beyond synthetic datasets

# Future Directions: Real-World Evaluation

## Standardized Real-World Benchmarks:

- Developing open-source datasets that reflect real-world coding complexity

## Context-Aware Optimization:

- Leveraging documentation, comments, and version history for better optimization

# Challenge 5: Trust and Reliability in AI-Driven Code Optimization

LMS CAN GENERATE INCONSISTENT, RANDOM, OR HALLUCINATED CODE OPTIMIZATIONS

DEVELOPERS STILL NEED TO VALIDATE AI-DRIVEN OPTIMIZATIONS

ENSURING FAIRNESS, ROBUSTNESS, AND SECURITY IN AI-ASSISTED CODING

# Future Directions: Trust and Reliability

**Reinforcement Learning from Human Feedback (RLHF):**

Using human preferences as reward signals to improve LM decisions.

**Human-AI Collaboration:**

Combining developer expertise with AI-generated suggestions for reliable optimizations.

# Conclusion

- LMs in code optimization present opportunities but face challenges
- Key gaps include model complexity, external system interaction, generalizability, real-world evaluation, and trust
- Future research should focus on improving scalability, adaptability, and human-AI collaboration

*Research Paper Presentation*

# Iterative Refinement of Project-Level Code Context for Precise Code Generation with Compiler Feedback

Authors: Zhangqian Bi, Yao Wan, Zheng Wang, Hongyu Zhang, Batu Guan, Fangxin Lu, Zili Zhang, Yulei Sui, Hai Jin, Xuanhua Shi

Presentation by Aditya Kakkar(zjq5mr) and Aryan Sawhney(ryd2fx)

# Aditya Kakkar (zjq5mr)

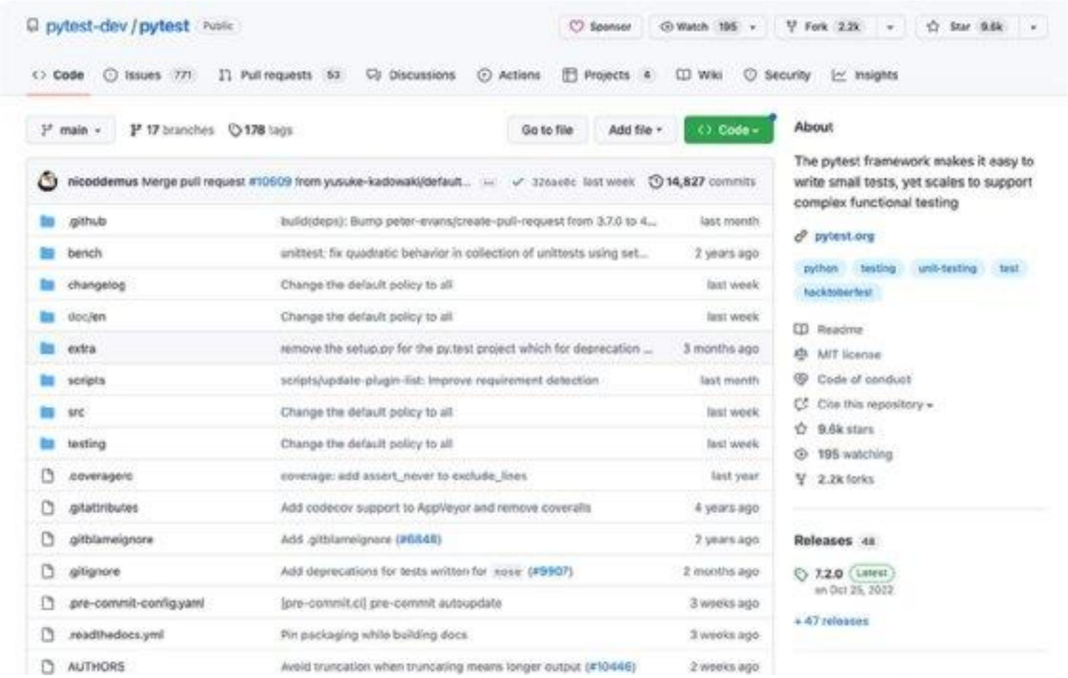
# Presentation Outline

- **Intro**
- **Problem**
- **Current standing**
- **CoCoGen**
- Experimental Setup
- Results
- Analysis
- Limitations

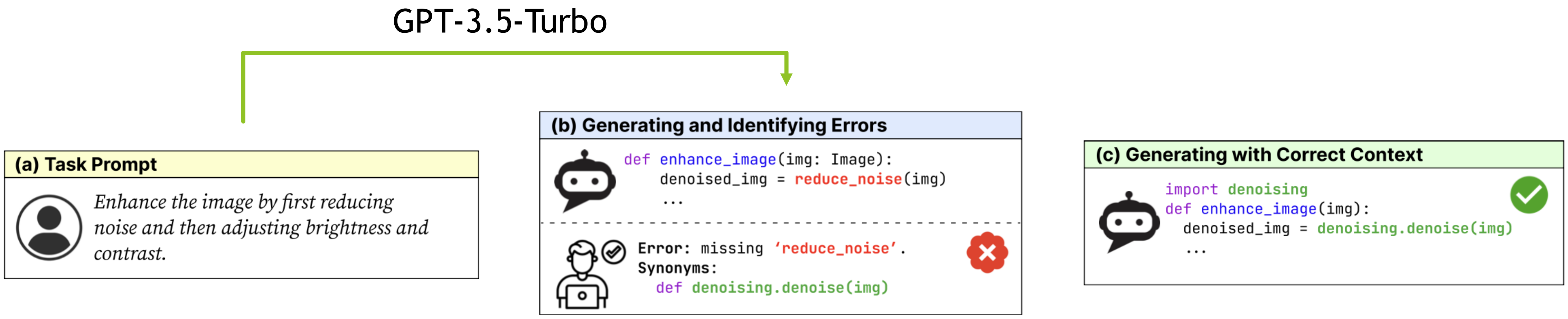

# Introduction

## Background

# Emergence of LLMs in code generation


pytest-dev / pytest Public
Sponsor
Watch 195
Fork 2.2k
Star 9.6k
Code
Issues 771
Pull requests 53
Discussions
Actions
Projects 4
Wiki
Security
Insights
main
17 branches
178 tags
Go to file
Add file
Code
nicoddemus Merge pull request #10609 from yusuke-kadowaki/default...
last week
14,827 commits
.github
build(deps): Bump peter-evans/create-pull-request from 3.7.0 to 4...
last month
bench
unittest: fix quadratic behavior in collection of unittests using set...
2 years ago
changelog
Change the default policy to all
last week
doc/en
Change the default policy to all
last week
extra
remove the setup.py for the py.test project which for deprecation ...
3 months ago
scripts
scripts/update-plugin-list: Improve requirement detection
last month
src
Change the default policy to all
last week
testing
Change the default policy to all
last week
.coveragerc
coverage: add assert_never to exclude_lines
last year
.gitattributes
Add codecov support to AppVeyor and remove coveralls
4 years ago
.gitblameignore
Add .gitblameignore (#6848)
2 years ago
.gitignore
Add deprecations for tests written for nose (#9907)
2 months ago
.pre-commit-config.yaml
[pre-commit.ci] pre-commit autoupdate
3 weeks ago
.readthedocs.yml
Pin packaging while building docs
3 weeks ago
AUTHORS
Avoid truncation when truncating means longer output (#10446)
2 weeks ago
About
The pytest framework makes it easy to write small tests, yet scales to support complex functional testing
pytest.org
python
testing
unit-testing
test
hacktoberfest
Readme
MIT license
Code of conduct
Cite this repository
9.6k stars
195 watching
2.2k forks
Releases 48
7.2.0 Latest
on Oct 25, 2022
+ 47 releases

# Motivation

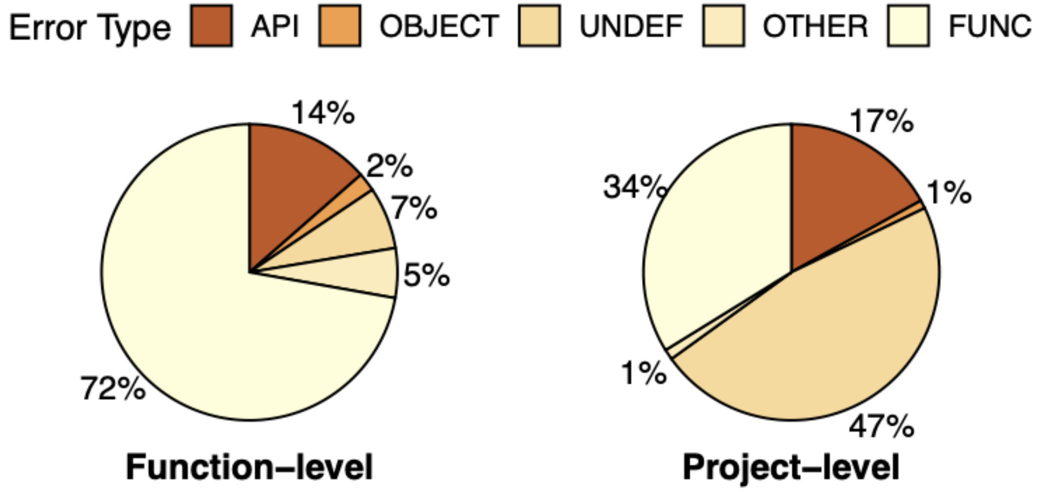



# The Problem

- LLMs struggle with code generation that depends on extensive project-specific context

# The Problem – Error Analysis

- Errors are more prevalent in project-level code generation compared to function-level code generation
- Project-Level Code: Higher incidence of UNDEF (Undefined Symbol) and API errors



# Existing Code Generation Approaches

## Standard LLM-Based Code Generation:

- Focus on generating code from isolated prompts
- Limitations due to input length constraints

## Retrieval-Based Methods:

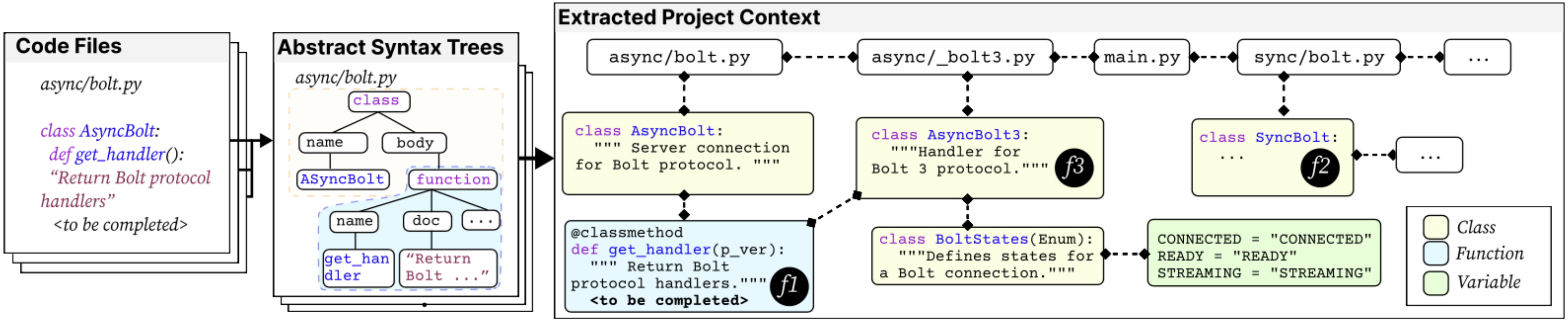
- Augmenting LLM prompts with retrieved code snippets
- Challenges with relevance and context matching

## Limitations:

- Inability to handle large project contexts
- Lack of iterative refinement using feedback mechanisms

# CoCoGen

- A method for extracting project-level code context through both syntactic and semantic approaches
- A component responsible for iterative generation and evaluation of solutions.

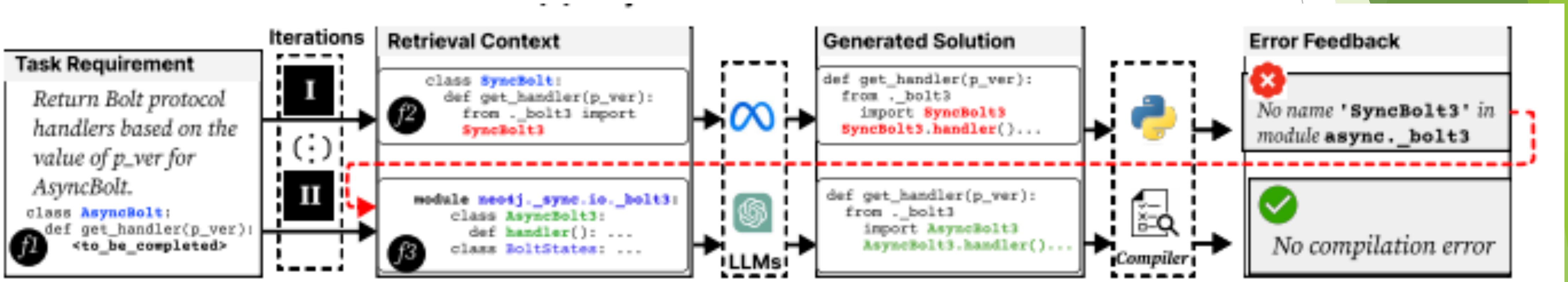



(a) Project-level Context Extraction

COCOGEN uses **Syntax-Directed Program Analysis**
Abstract Syntax Tree

# Project-Level Code Context Extraction

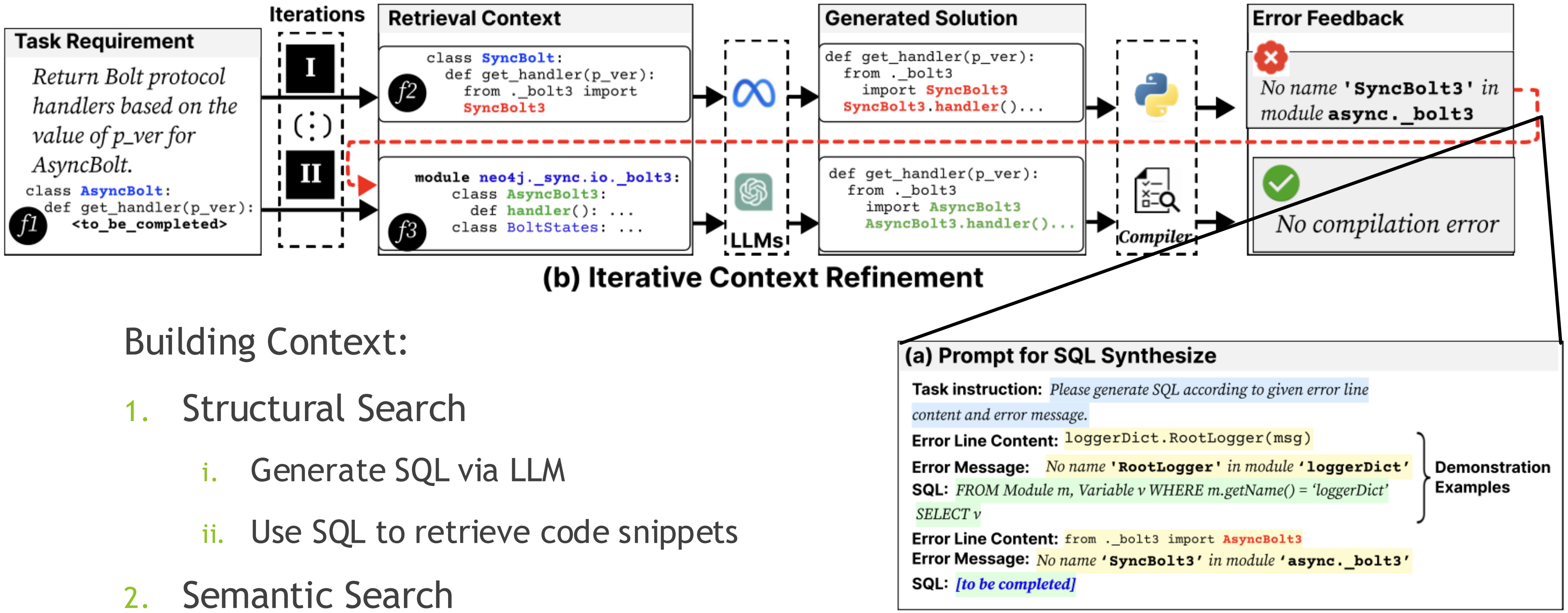
# Retrieval-Augmented Code Generation



(b) Iterative Context Refinement

$$sim(h_q, h_c) = \frac{h_q^\top h_c}{||h_q|| \cdot ||h_c||}$$

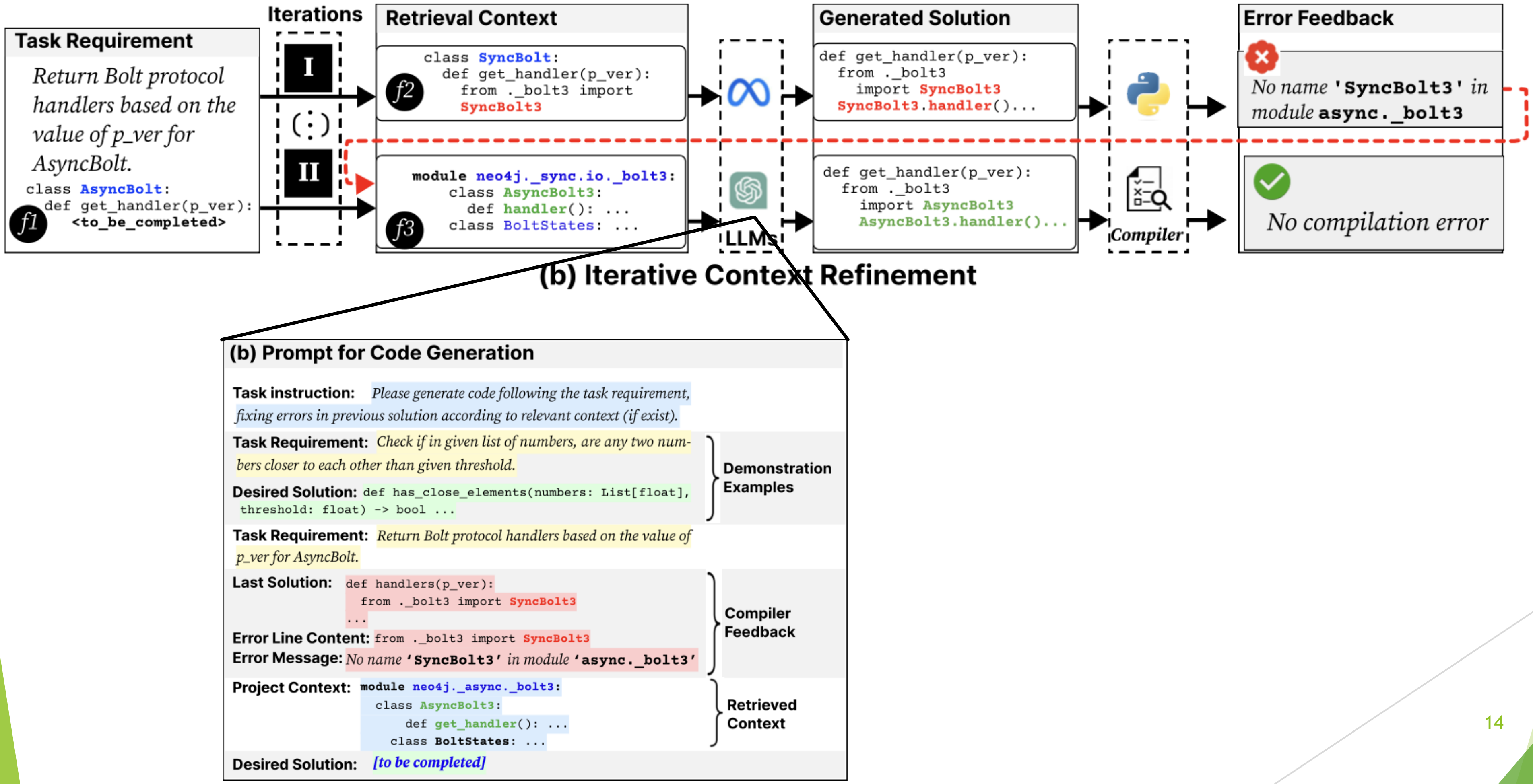
# COCOGEN Method - Retrieval-Augmented Code Generation




Building Context:

1. Structural Search
   i. Generate SQL via LLM
   ii. Use SQL to retrieve code snippets
2. Semantic Search
   i. Retrieve similar code using dense passage retrieval

# COCOGEN Method – Iterative Context Refinement



# Aryan Sawhney (ryd2fx)

# Experimental Setup – Models and Datasets

## LLMs Used:

- GPT-3.5-Turbo
- Code Llama (13B variant)

## Benchmark Dataset:

- CoderEval-Python
  - ➢Contains tasks requiring project-specific context
  - ➢Categorized into different levels of context dependency

# Experimental Setup – Baseline Methods

**Direct Generation:** LLMs generate code based solely on the task description.

**ReACC:** Retrieval-augmented generation using semantic similarity

**RepoCoder:** Iterative retrieval and generation without compiler feedback

# Experimental Setup – Evaluation Metrics

## Pass@k Metric:

- Measures the percentage of tasks where at least one out of k generated solutions passes all test cases

## Error Analysis:

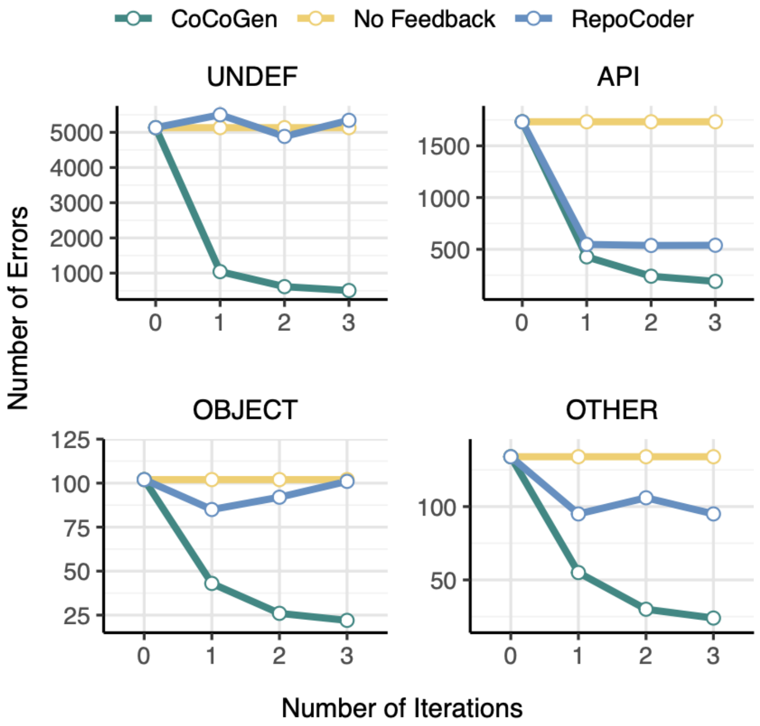
- Categorizing errors into:
  - Undefined Symbols (UNDEF)
  - Incorrect API Usage (API)
  - Improper Object Use (OBJECT)
  - Runtime or Functional Errors (FUNC)
  - Other Syntax/Semantic Errors (OTHER)

# Results – Overall Performance

- Performance Highlights:
  - Over 80% relative increase in pass rates for project-level tasks
  - Consistent performance gains with both GPT-3.5-Turbo and Code Llama

| Data Split | Class Runnable | | | File Runnable | | | Project Runnable | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | Pass@1 | Pass@5 | Pass@10 | Pass@1 | Pass@5 | Pass@10 | Pass@1 | Pass@5 | Pass@10 |
| *LLM: GPT-3.5-Turbo* | | | | | | | | | |
| Direct | 8.73 | 12.57 | 14.55 | 19.85 | 27.62 | 30.88 | 9.57 | 12.08 | 13.04 |
| ReACC | 20.36 | 33.27 | 38.18 | 17.65 | 28.92 | 33.82 | 11.30 | 19.53 | 21.74 |
| RepoCoder | **35.45** | 40.46 | 41.82 | 29.41 | 34.61 | 36.76 | 16.96 | 19.57 | 21.74 |
| CoCoGen | 28.00 | **44.92** | **49.09** | **30.29** | **43.58** | **47.06** | **21.30** | **36.73** | **39.13** |
| *LLM: Code Llama (13B)* | | | | | | | | | |
| Direct | 18.91 | 30.65 | 34.55 | 18.53 | 27.82 | 29.41 | 5.22 | 8.70 | 13.04 |
| ReACC | 20.36 | 33.27 | 38.18 | **17.65** | 27.61 | 33.82 | 11.30 | 19.53 | 21.74 |
| RepoCoder | 17.82 | 35.22 | 40.00 | 15.00 | 28.31 | 32.35 | **16.09** | 21.36 | 21.74 |
| CoCoGen | **26.36** | **39.42** | **41.82** | 17.06 | **29.39** | **33.82** | 13.04 | **28.04** | **34.78** |

# Results – Analysis




- Performance Gains with Iterations:
  - Pass rates increase with each iteration of refinement
  - Majority of errors are resolved within the first few iterations
  - COCOGEN's iterative approach yields better results than methods without iterative refinement
- Significant reduction in 'Undefined Symbol' (UNDEF) and 'Incorrect API Usage' (API) errors
  - Compiler feedback effectively guides the retrieval of necessary context

# Results – Ablation Studies

| Method | Pass@1 | Pass@5 | Pass@10 |
|---|---|---|---|
| CoCoGen | 28.01 | **43.01** | **46.58** |
| - w/o CF and SQL (RepoCoder) | **28.72** | 34.44 | 36.30 |
| - w/ CF, w/o SQL, w/o Semantic | 25.69 | 37.50 | 41.78 |
| - w/ CF and SQL, w/o Semantic | 26.37 | 38.31 | 41.78 |
| - w/ CF and Semantic, w/o SQL | 27.39 | 40.02 | 44.45 |

- Component Contribution:
  - Compiler Feedback
  - Structural Queries (SQL retrieval)
  - Semantic Retrieval
- Removing any component leads to a drop in performance
- The synergy of all components makes COCOGEN effective

# Limitations

**Runtime Errors:**

- COCOGEN focuses on compilation errors
- Runtime errors that occur during execution are not addressed

**Degenerate Solutions:**

- Instances where LLMs generate simplistic or incorrect code that compiles but doesn't function correctly

# Future Work

### Integration of Execution Feedback:

- Incorporate test execution results to handle runtime errors
- Use debugging techniques to refine code further

### Enhanced Contextual Understanding:

- Utilize project documentation and comments
- Improve semantic retrieval methods

### Scalability:

- Optimize for larger projects with complex dependencies
- Explore more efficient retrieval and analysis techniques

# Questions?