Efficient and Green Code LLMs:

Happier Software Engineers, Happier Planet

SMU SINGAPORE MANAGEMENT UNIVERSITY

David Lo

School of Computing and Information Systems

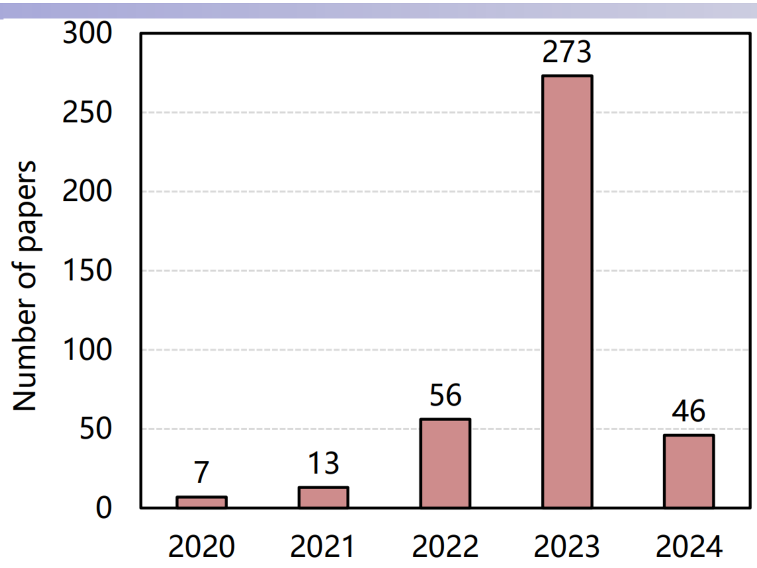
IFIP WG 2.4 Meeting, Singapore

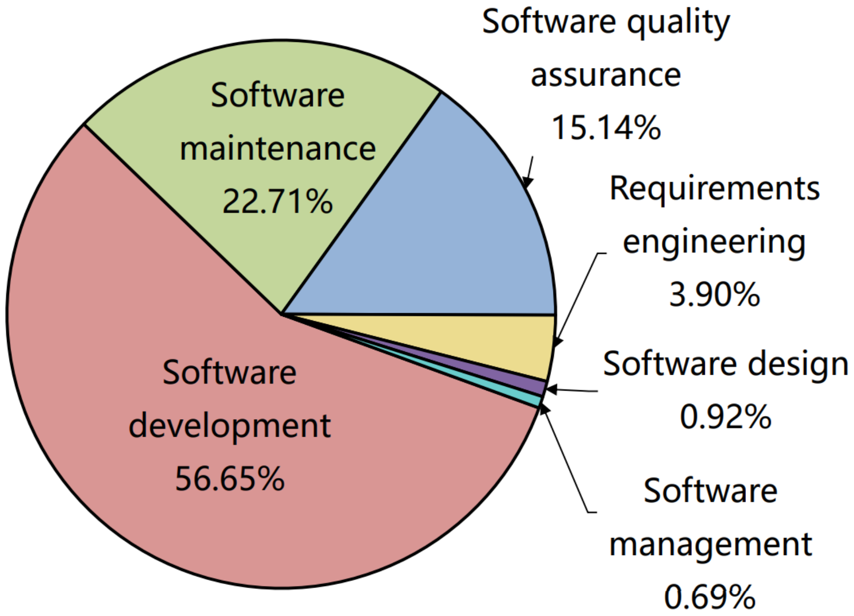
Code LLMs Has Helped Many SE Scenarios

TOSEM 2024

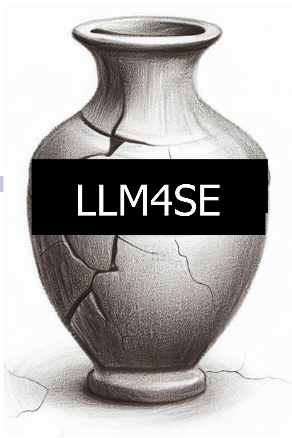
Large Language Models for Software Engineering: A Systematic Literature Review

XINYI HOU*, Huazhong University of Science and Technology, China
YANJIE ZHAO*, Monash University, Australia
YUE LIU, Monash University, Australia
ZHOU YANG, Singapore Management University, Singapore
KAILONG WANG, Huazhong University of Science and Technology, China
LI LI, Beihang University, China
XIAPU LUO, The Hong Kong Polytechnic University, China
DAVID LO, Singapore Management University, Singapore
JOHN GRUNDY, Monash University, Australia
HAOYU WANG†, Huazhong University of Science and Technology, China



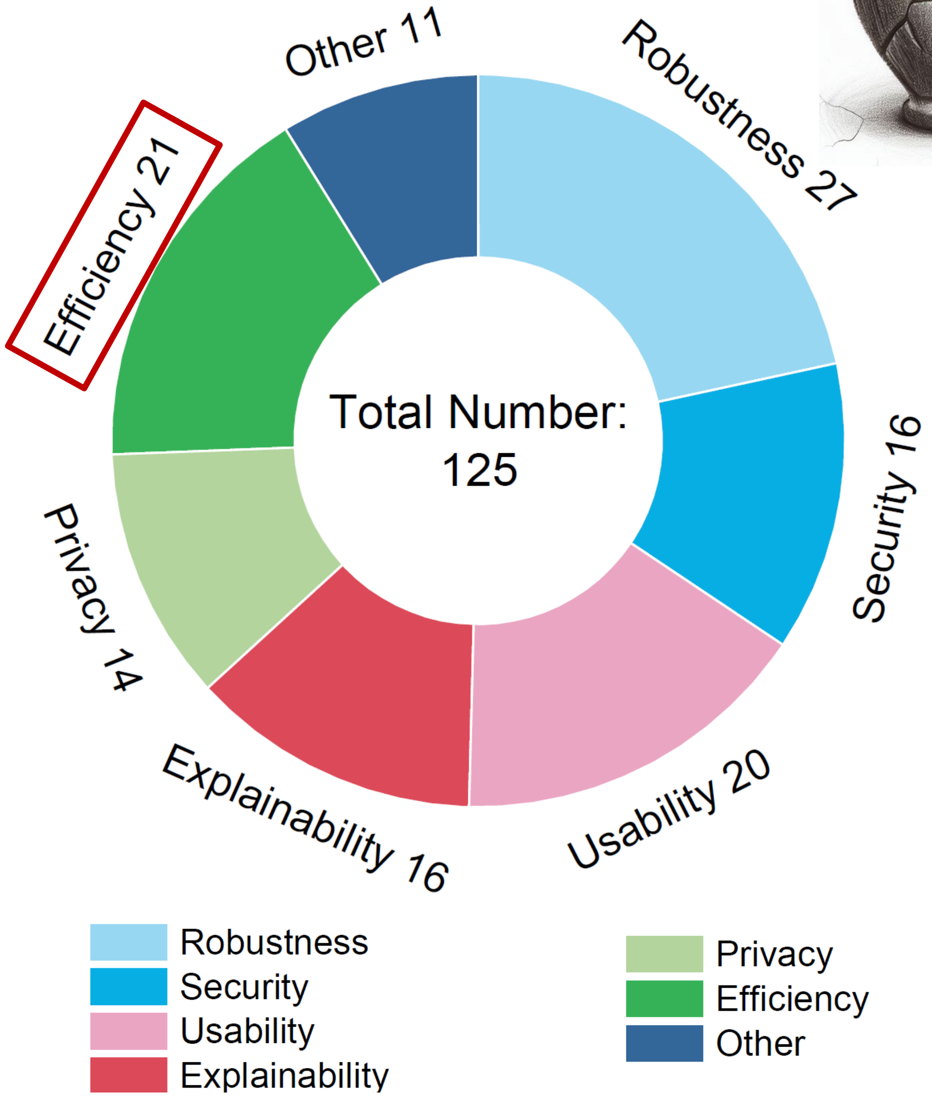


Many Open Problems



***Robustness*, *Security*, *Privacy*, *Explainability*, *Efficiency*, and *Usability* of Large Language Models for Code**

ZHOU YANG, Singapore Management University, Singapore
ZHENSU SUN, Singapore Management University, Singapore
TERRY ZHUO YUE, Singapore Management University, Singapore
PREMKUMAR DEVANBU, Department of Computer Science, UC Davis, USA
DAVID LO, Singapore Management University, Singapore

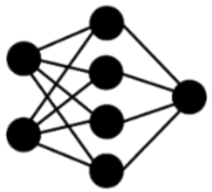


Code LLMs are Large, Slow, …

Developers often prefer local AI4SE tools due to privacy and latency concerns

- *E.g.,* Apple banned internal use of external AI tools
- *E.g.,* 20% of GitHub Copilot's issues are related to network connectivity

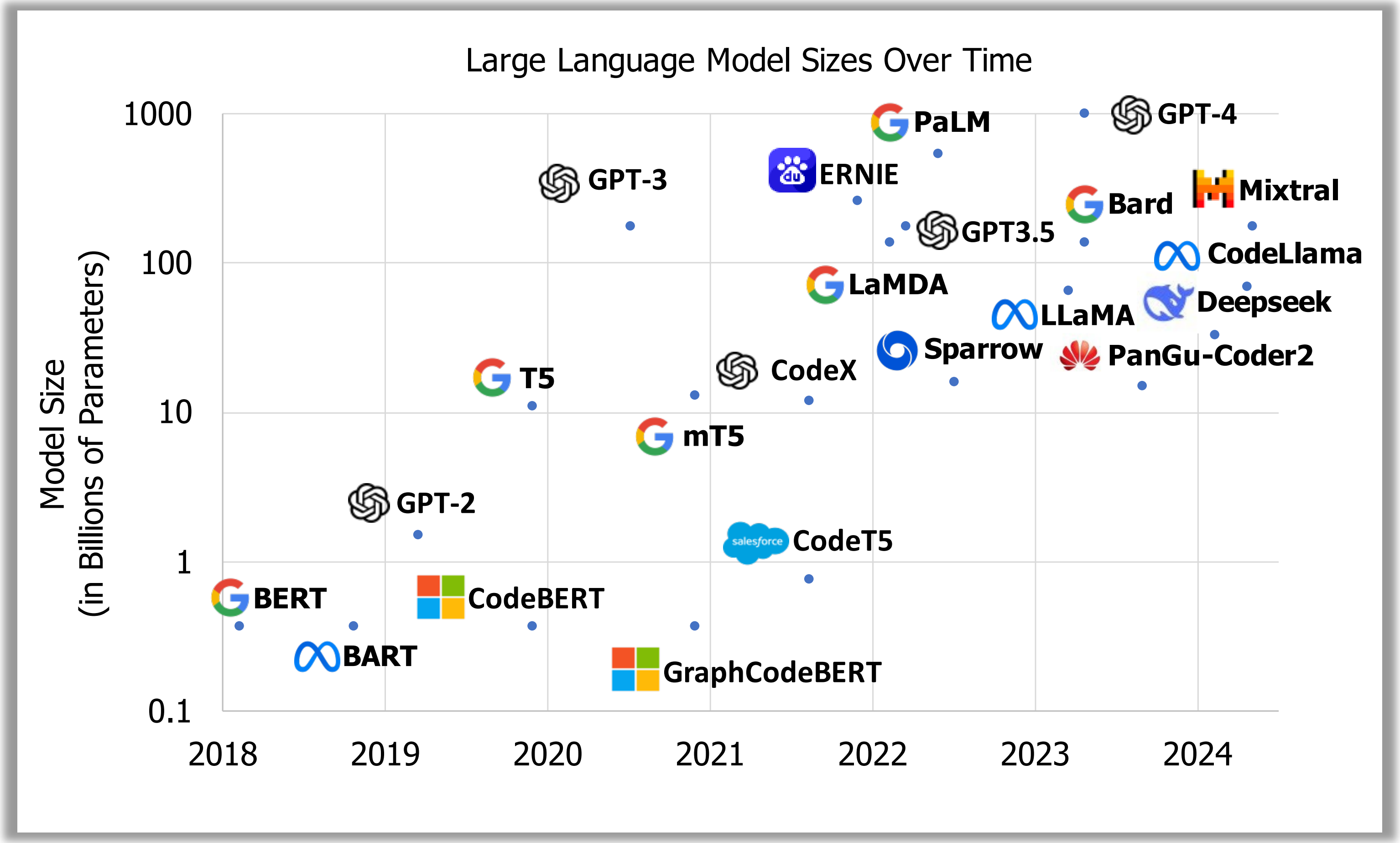
Deploying LLMs to IDE has issues:

Expectations	*Reality*
• "***50MB*** model is upper bound, and ***3MB*** is preferred in modern IDE" • "***0.1 seconds*** is preferred in modern IDE or editor design" - *VSCode Team*	 CodeBERT Size: **> 400MB** Latency: **> 1.5s/query**

Code LLMs are Large, Slow, **and not Green**

LLM has high energy consumption and carbon footprint

- Typical laptop's battery can support CodeBERT for ***13.2 mins***
- Using CodeBERT a thousand times produces ***0.14 kg of CO2*** (driving a car for ***1 km***)
- Much worse for larger LLMs



Battery and Power[3]

M3

70-watt-hour lithium-polymer battery[3]

Efficient and Green Code LLMs: Three Strategies

Stop	**Simplify**	**Shrink**

Efficient and Green Code LLMs: Three **S**trategies

Stop

Simplify

Shrink

Stop Unhelpful Code Completion with *FrugalCoder*

TOSEM 2024

Don't Complete It! Preventing Unhelpful Code Completion for Productive and Sustainable Neural Code Completion Systems

ZHENSU SUN, Singapore Management University, Singapore
XIAONING DU*, Monash University, Australia
FU SONG†‡, Key Laboratory of System Software (Chinese Academy of Sciences), State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
SHANGWEN WANG, National University of Defense Technology, China
MINGZE NI, University of Technology Sydney, Australia
LI LI, Beihang University, Beijing, China
DAVID LO, Singapore Management University, Singapore

First work to investigate the problem of **unhelpful code completions**

LLM-based Code Completion Brings New Challenges

LLM-based Code Completion is Popular

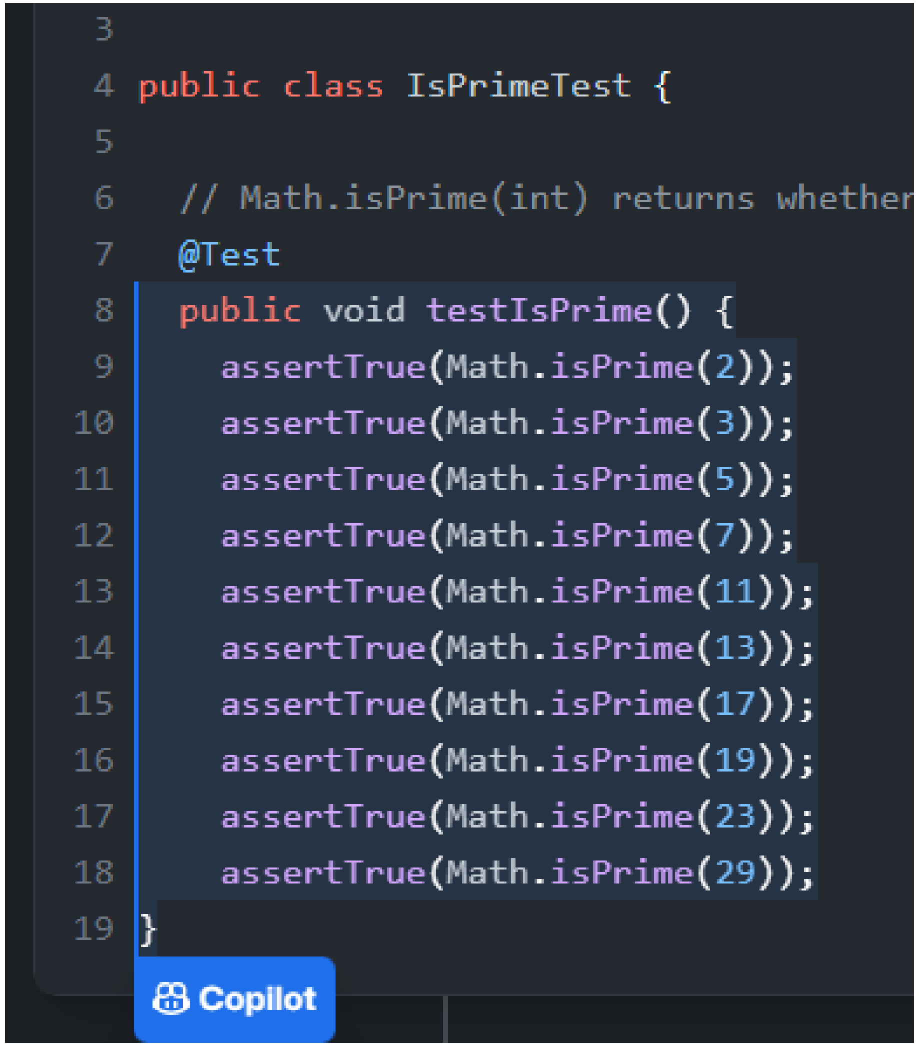
- Each user of Github Copilot receives one suggestion roughly every 3 minutes [GitHub22]

Low Acceptance Rate

- Code completion requests are complex in real world
- Only 30% of completions are accepted by the users of Github Copilot [GitHub22]

High Computation & Latency

- LLM-based code completion requires large scale computing and causes latency

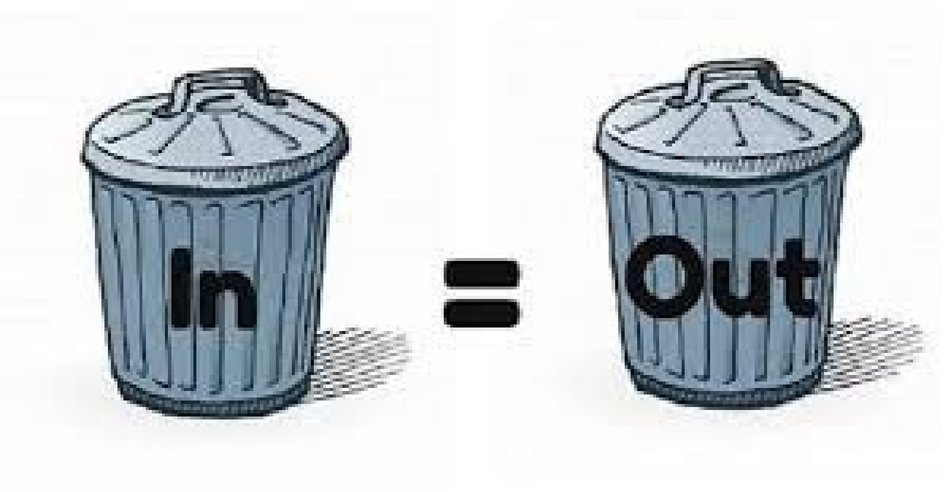


[GitHub22] Albert Ziegler (GitHub), Eirini Kalliamvakou (GitHub), Shawn Simister, et al. Productivity Assessment of Neural Code Completion. MAPS'22 at PLDI'22.

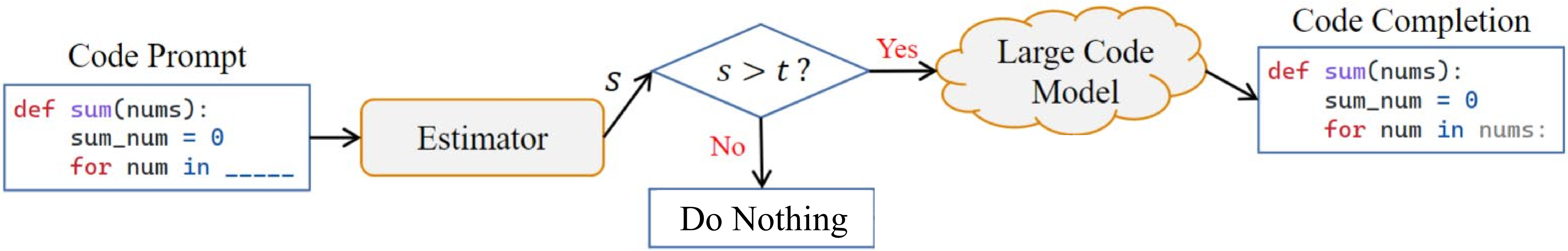
What Cause Unhelpful Code Completions?

(1) Requests that are **beyond the capability of the LLM**

(2) Requests that **do not contain sufficient information**, e.g., meaningless identifiers, vague intention, etc.



FrugalCoder: Identify & Reject Unpromising Code Prompts



- Using lightweight estimator to estimate the quality of the code completion
- Decide whether to proceed based on a pre-defined threshold

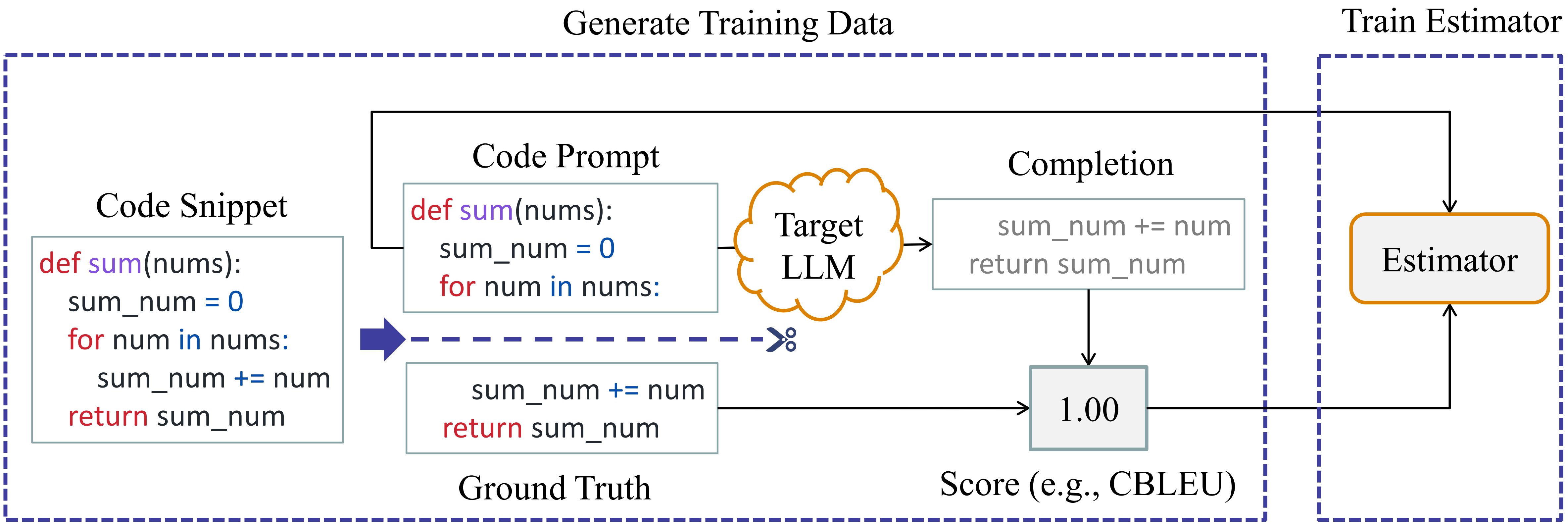
Challenge 1: Efficacy

The estimator should effectively estimate code completion quality

Challenge 2: Efficiency

The cost of running the estimator should be lower than the LLM cost

FrugalCoder: Building the Estimator



Results: Feasibility of FrugalCoder (StarCoder + CodeGen2)

- **Deep Learning as Estimator**

  Lightweight Transformer

Efficacy
Reject **20%** of requests with a **95.1%** Precision
Improve Acceptance Rate from **27.4%** to **33.0%**

Efficiency
5.1 ms for each query

- **Traditional ML as Estimator**

  Adaboost

Efficacy
Reject **20%** of requests with a **92.1%** Precision
Improve Acceptance Rate from **27.4%** to **32.3%**

Efficiency
0.1 ms for each query

Efficient and Green Code LLMs: Three **S**trategies

Stop

Simplify

Shrink

Simplify Programming Language Grammars

ISSTA 2024

AI Coders Are Among Us: Rethinking Programming Language Grammar Towards Efficient Code Generation

Zhensu Sun
Singapore Management University
Singapore
zssun@smu.edu.sg

Xiaoning Du*
Monash University
Australia
xiaoning.du@monash.edu

Zhou Yang
Singapore Management University
Singapore
zyang@smu.edu.sg

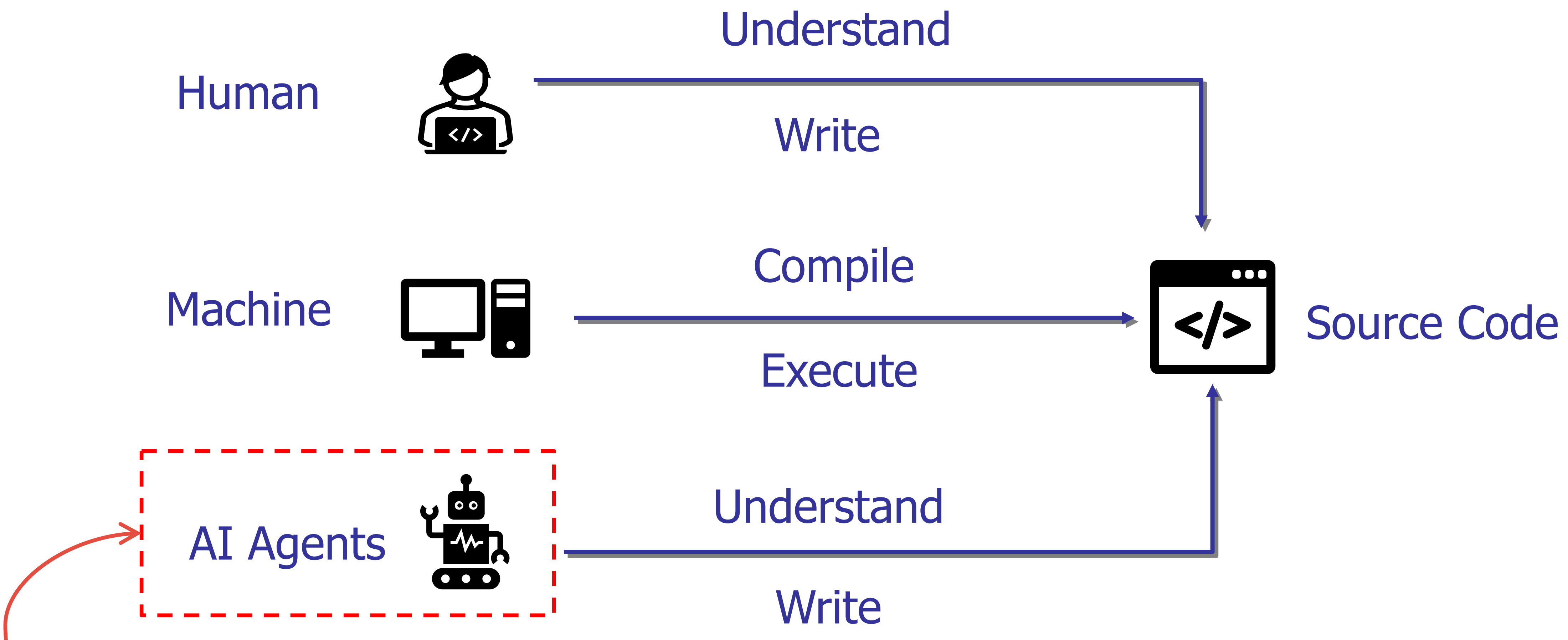
Li Li
Beihang University
China
lilicoding@ieee.org

David Lo
Singapore Management University
Singapore
davidlo@smu.edu.sg

Won ACM SIGSOFT Distinguished Paper Award

First work to propose a **programming language grammar for AI agents**

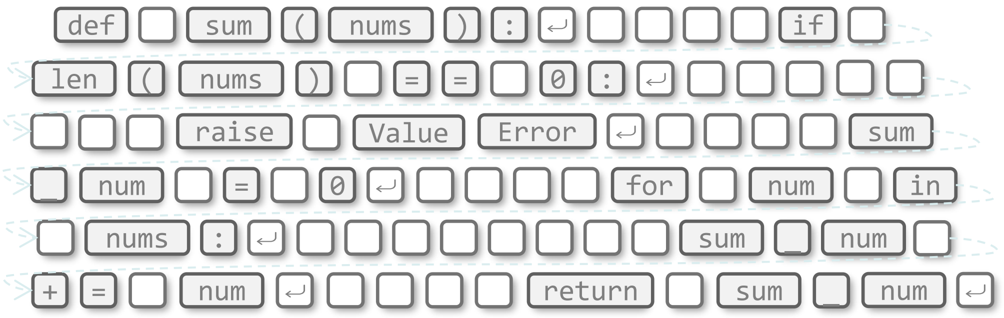
AI Agents: The Third Audience



Active "developers" that utilize programming to accomplish various tasks

Human-Centric Programming Language: Readability Counts

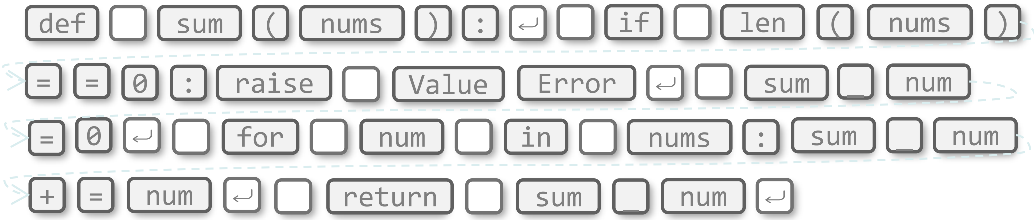
```python
def sum(nums):
    if len(nums) == 0:
        raise ValueError
    sum_num = 0
    for num in nums:
        sum_num += num
    return sum_num
```



Good practice

"Looks" wordy

```python
def sum(nums):
 if len(nums)==0:raise ValueError
 sum_num=0
 for num in nums:sum_num+=num
 return sum_num
```



Bad practice

"Looks" succinct

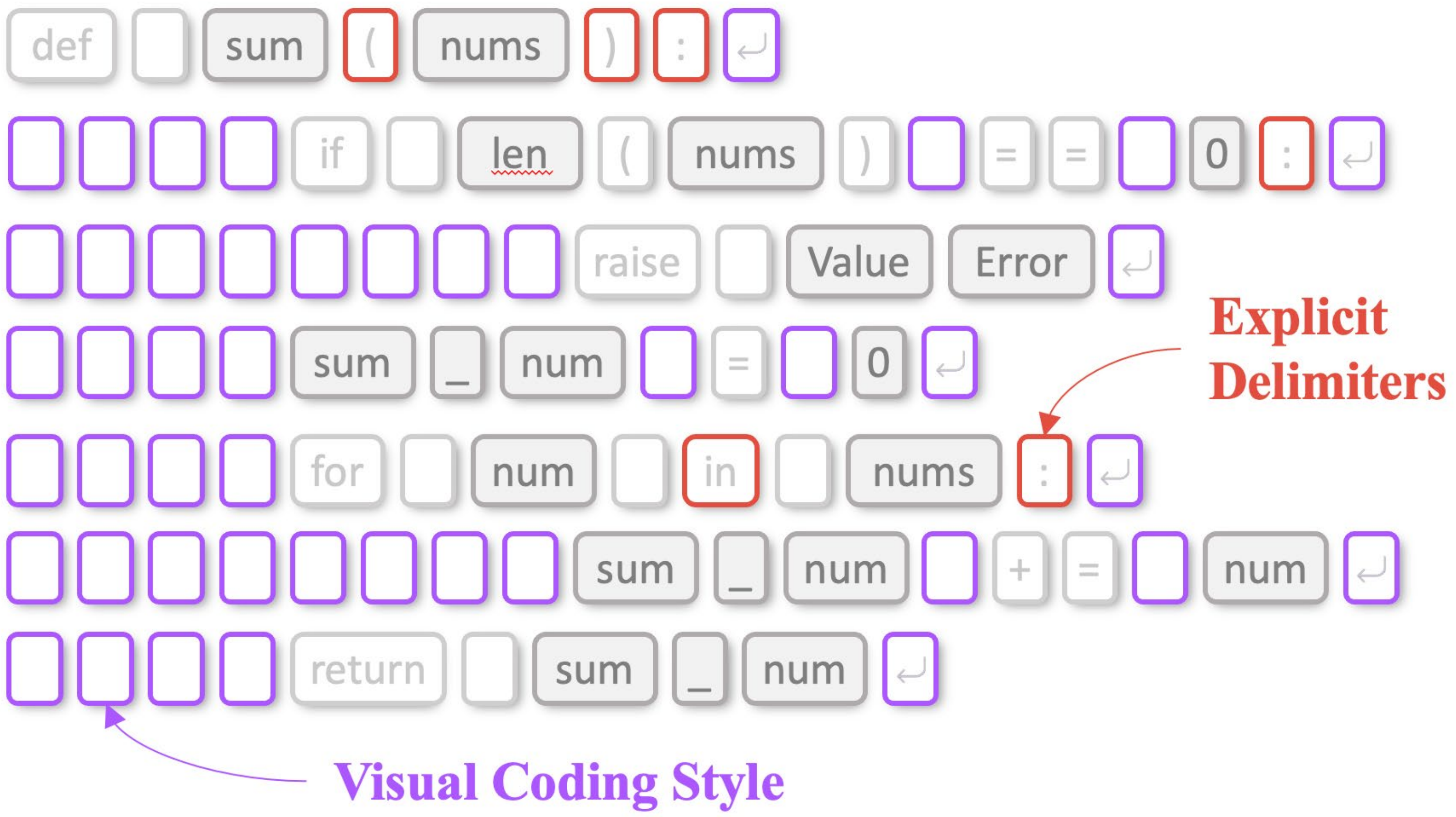
Human-Centric Programming Language Grammar Design

Visual Coding

Include symbols like line breaks and indentions

Explicit Delimiters

Use explicit delimiters to define code structures despite some delimiters not being essential for parsing



Are there better programming language grammars for AI agents?

SimPy: A proof-of-concept AI-agent oriented grammar

Python — Tokenized by CodeBERT — **128 tokens**

```
def two_sum(nums: list[int], target: int) -> list[int]:\n
    chk_map: dict[int, int] = {}\n
    for index, val in enumerate(nums):\n
        compl = target - val\n
        if compl in chk_map:\n
            return [chk_map[compl], index]\n
        chk_map[val] = index\n
    return []
```

Same Execution Results

SimPy — **80 tokens**

```
<def_stmt>two_sum nums:list[int] target:int<arrow>list
[int]<block_start>chk_map:dict[int int]={}<for_stmt>
index,val enumerate(nums)<block_start>compl=target-val
<if_stmt>compl<in>chk_map<block_start><return>[chk_map
[compl] index]<block_end>chk_map[val]=index<block_end>
<return>[]<block_end>
```

Replace notations with tokens

Replace keywords and symbols (e.g., “if”, “for”, etc.) with specialized tokens

Restrict coding style

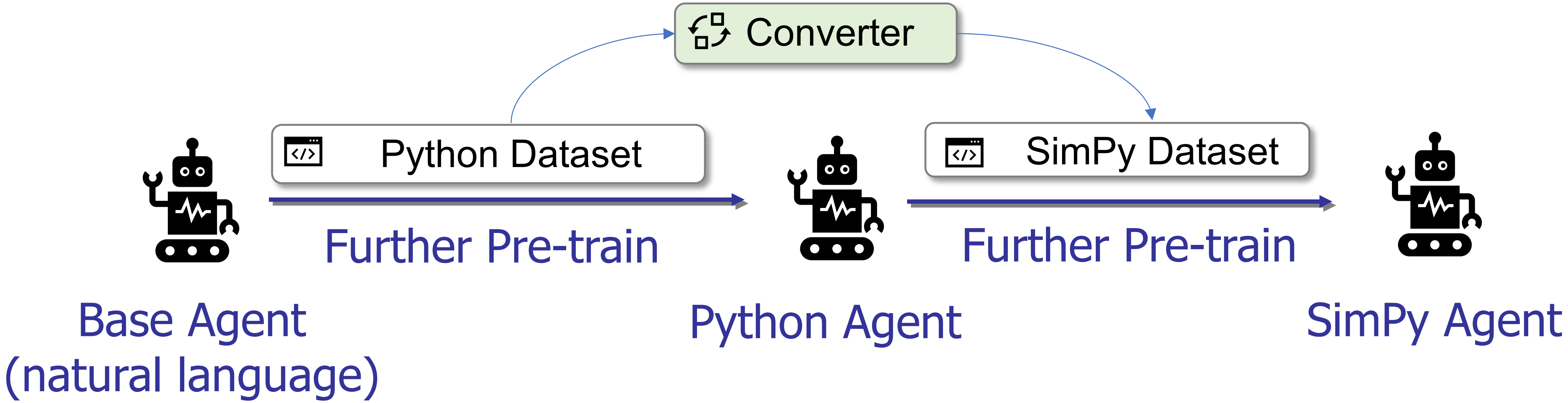
Streamline white spaces, line breaks, indents, etc. preserving only essential separators

Simplify grammar tokens

For every grammar token in every production, we review whether it can be removed, merged with others, or replaced with white spaces

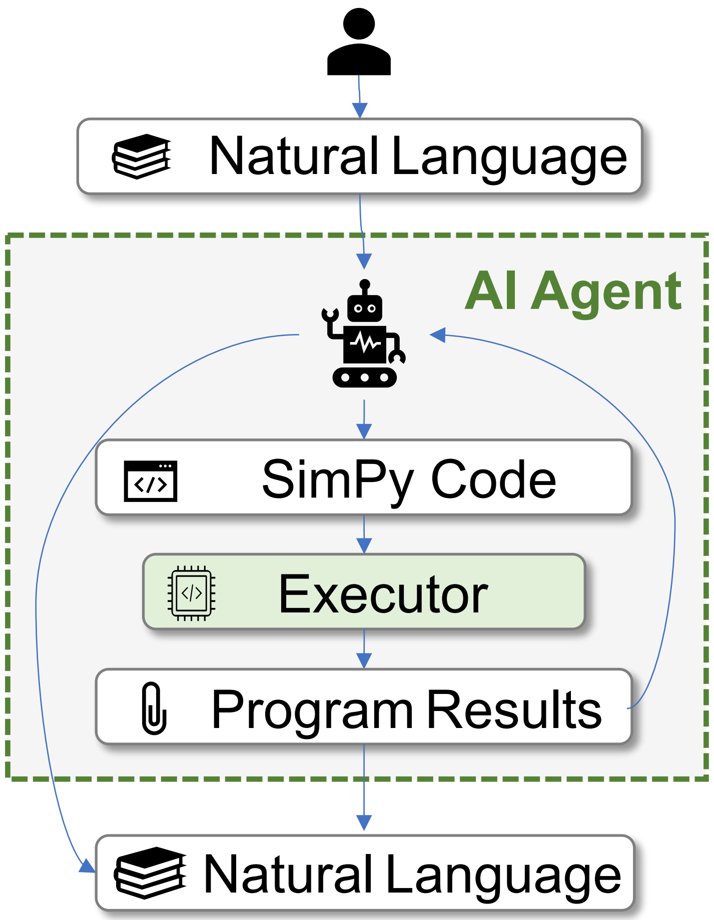
Making AI Agent Understand and Write SimPy

LLM first trained on Python and then on SimPy (with equivalent sample size)

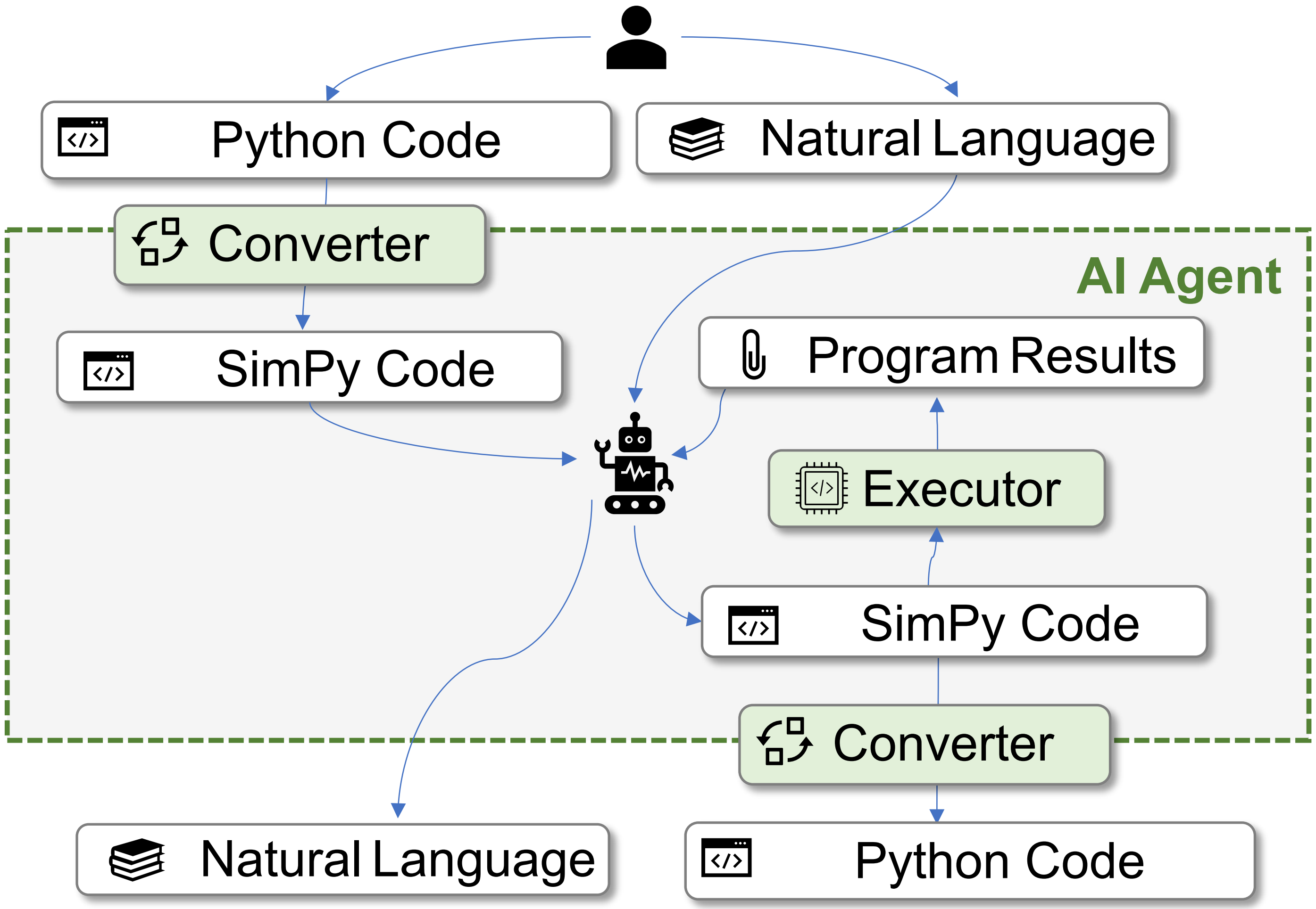


✓ SimPy is more similar to Python than natural language

SimPy's Possible Usage Scenarios



Basic usage scenario



Extended usage scenario: **DualCode**

RQ1: Token Reduction

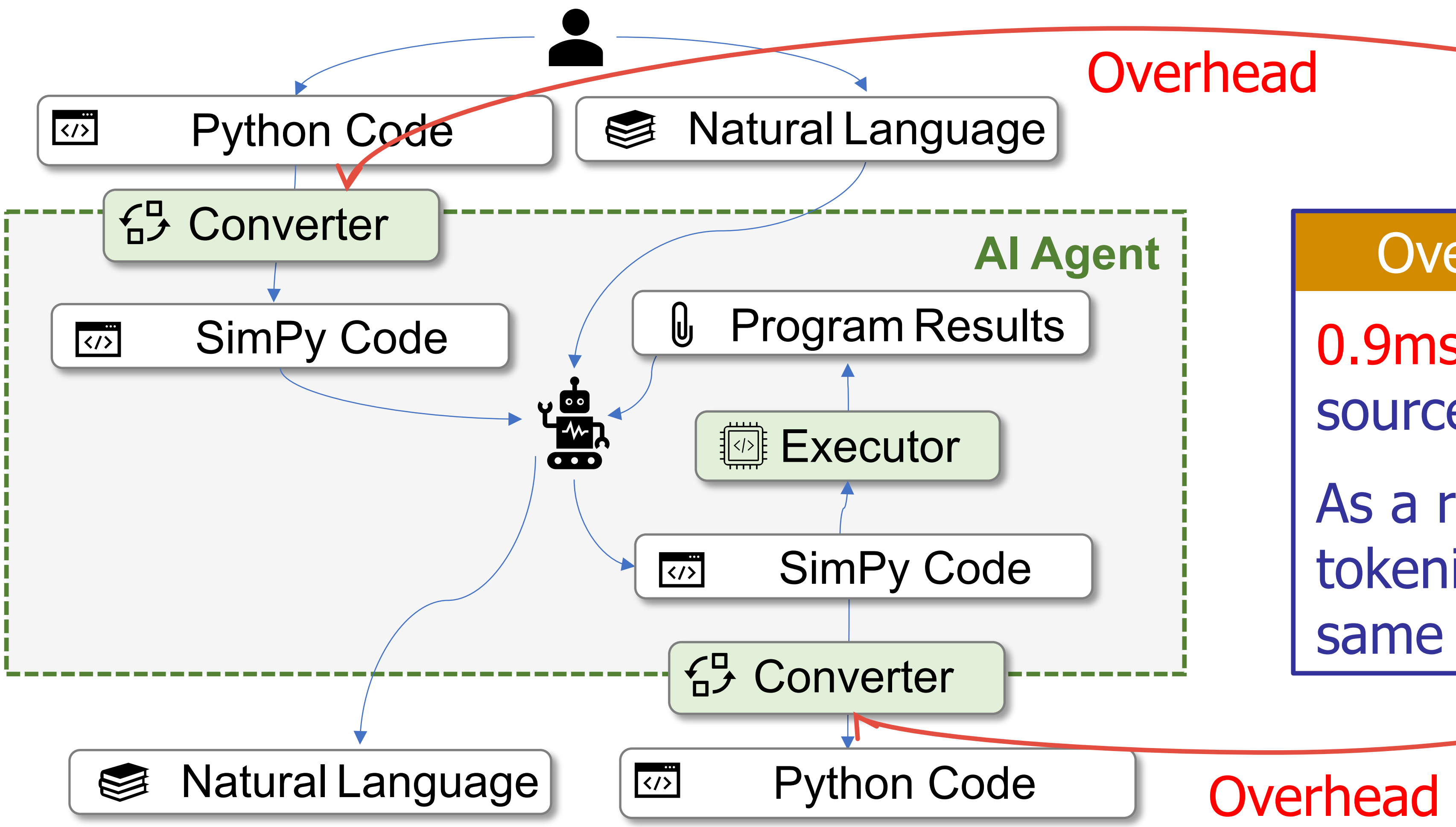
- SimPy reduces the number of tokens required for source code representation.
- CodeBERT and GPT-4 benefiting from a **34.7%** and **10.4%** reduction.

Tokenizer	Vocab Source	Vocab Size	Tokens		
			Python	SimPy	
CodeBert	Code	50k	1.33B	0.87B	34.7%↓
CodeLlama	Web	32k	0.97B	0.84B	13.5%↓
Codex	Web	51k	0.93B	0.82B	12.6%↓
CodeT5	Code	32k	0.91B	0.78B	13.8%↓
StarCoder	Code	49k	0.83B	0.76B	8.6%↓
GPT-3.5	Web	100k	0.71B	0.63B	10.4%↓
GPT-4	Web	100k	0.71B	0.63B	10.4%↓

RQ2: Efficacy of AI Agents Trained on SimPy

Model	Training Strategy	Pass@10
CodeGen-NL (350M)	Python	7.32%
	Python->SimPy	**9.15%**
TinyLlama (1.1B)	Python	13.41%
	Python->SimPy	**14.02%**
Pythia (1.0B)	Python	9.76%
	Python->SimPy	**10.00%**

RQ3: Overhead of DualCode Scenario



Extended usage scenario: DualCode

Overhead of the converter

0.9ms CPU time for processing source code within 500 tokens.

As a reference, Huggingface tokenizer costs 0.7ms for the same code

Efficient and Green Code LLMs: Three **S**trategies

Stop | **Simplify** | **Shrink**

Shrink Code LLMs with *Compressor* & *Avatar*

ASE 2022 Compressor

Compressing Pre-trained Models of Code into 3 MB

Jieke Shi, Zhou Yang, Bowen Xu*, Hong Jin Kang and David Lo
School of Computing and Information Systems
Singapore Management University
{jiekeshi,zyang,bowenxu.2017,hjkang.2018,davidlo}@smu.edu.sg

First work to compress code LLMs: **160× smaller** and **4.23× faster**

Nominated for ACM SIGSOFT Distinguished Paper Award

Today's Sharing

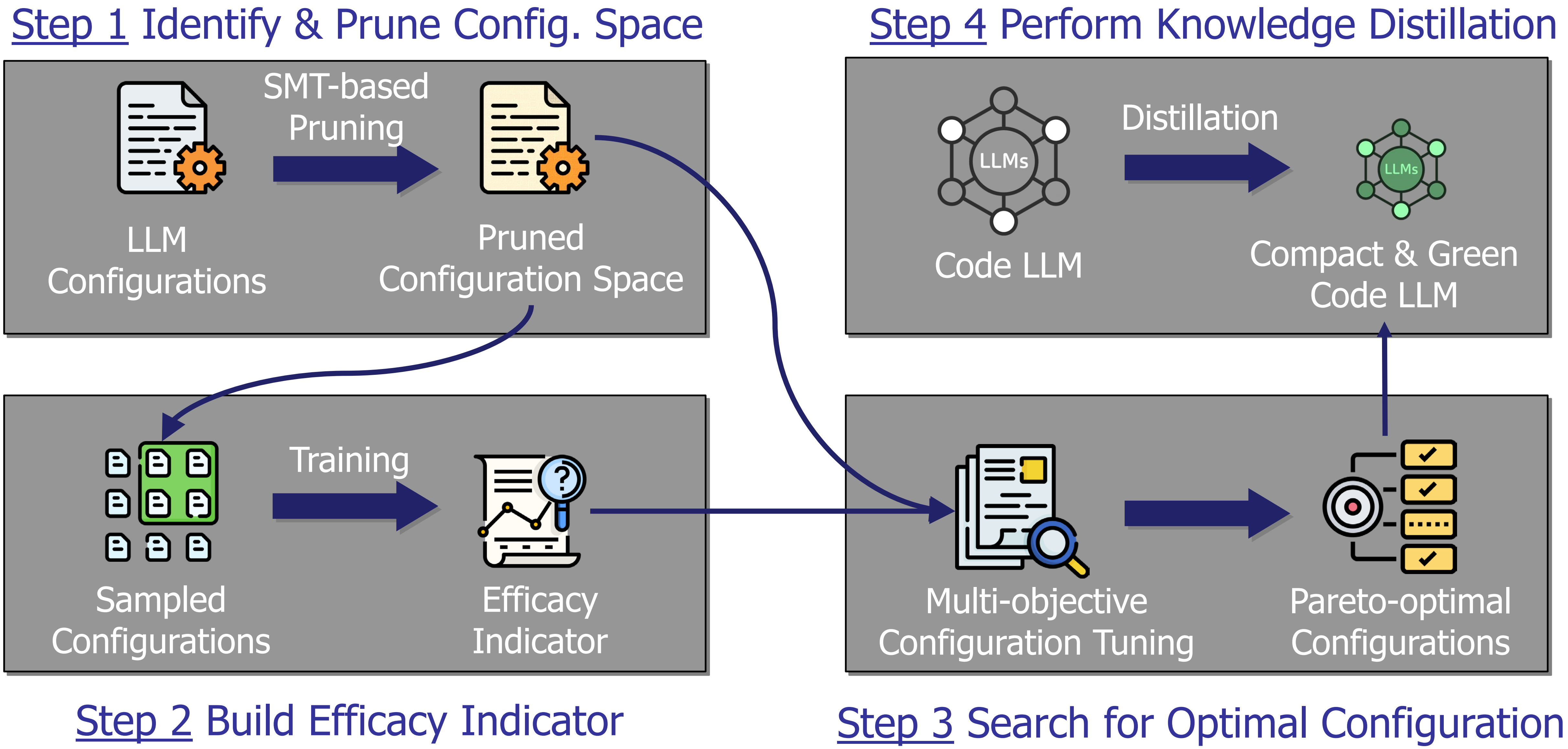
ICSE 2024 Avatar

Greening Large Language Models of Code

Jieke Shi[◇], Zhou Yang[◇], Hong Jin Kang[♠], Bowen Xu[♣], Junda He[◇], and David Lo[◇]
[◇]School of Computing and Information Systems, Singapore Management University, Singapore
[♠]Department of Computer Science, University of California, Los Angeles, USA
[♣]Department of Computer Science, North Carolina State University, Raleigh, USA
{jiekeshi, zyang, jundahe, davidlo}@smu.edu.sg, hjkang@cs.ucla.edu, bxu22@ncsu.edu

Compress code LLMs: **160x smaller, 76× faster, 184× more energy-saving**, and **157× less in carbon footprint**

Avatar’s Overall Workflow



Step 1: Prune Massive Configuration Space

```
"tokenizer": ["Byte-Pair Encoding", "WordPiece",
   ↪ "Unigram", "Word"],
"vocab_size": range(1000, 50265),
"num_hidden_layers": range(1, 12),
"hidden_size": range(16, 768),
"hidden_act": ["GELU", "ReLU", "SiLU", "GELU_new"],
"hidden_dropout_prob": [0.1, 0.2, 0.3, 0.4, 0.5],
"intermediate_size": range(16, 3072),
"num_attention_heads": range(1, 12),
"attention_probs_dropout_prob": [0.1, 0.2, 0.3, 0.4,
   ↪0.5],
"max_sequence_length": range(256, 512),
"position_embedding_type":["absolute", "relative_key",
   ↪ "relative_key_query"],
"learning_rate": [1e-3, 1e-4, 5e-5],
"batch_size": [16, 32, 64]
```

Typical configuration space of LLMs containing 4.5×10^{19} plausible configurations

Too large & many are infeasible!

Step 1: Prune Massive Configuration Space

formulating model size
and its constraint:

$$\mathtt{size}(c) \leq 3\ \mathrm{MB} \quad \text{s.t.} \quad c \in C$$

$$\mathtt{size}(c) = \frac{4(v+s+3)h}{1024 \times 1024} + \frac{4(4h^2+(9+2i)h+i)l}{1024 \times 1024} + \frac{2h^2+4h+2}{1024 \times 1024}$$

- C: the configuration space
- c: a configuration
- v: vocabulary size
- s: model's maximum input length
- l: number of hidden layers
- h: dimension of hidden layers
- i: dimension of intermediate NN layers

Step 1: Prune Massive Configuration Space

Large space of 4.5×10^{19}
plausible configurations

Z3

Using SMT solver
to prune

$$\text{size}(c) \leq 3\text{ MB} \quad \text{s.t.} \quad c \in C$$

$$\text{size}(c) = \frac{4(v+s+3)h}{1024 \times 1024} + \frac{4(4h^2+(9+2i)h+i)l}{1024 \times 1024} + \frac{2h^2+4h+2}{1024 \times 1024}$$

Remaining space after pruning accounts for
only 28.9% of the original one

Step 3: Identify Pareto-Optimal Configurations

Avatar uses a multi-objective optimization algorithm to find Pareto-optimal configurations, i.e., configurations that achieve the **best trade-off among all objectives**

Model Size

Prediction accuracy on a downstream task (using the efficacy indicator created in Step 2)

$$\min_{c} \quad \{\texttt{size}(c), \texttt{FLOPs}(c), -\texttt{efficacy}(c)\}$$
$$\text{s.t.} \quad c \in C$$

Pruned configuration space

Proxy of model's inference latency, energy consumption, and carbon footprint

Step 4: Perform Knowledge Distillation

Outputs of the large code LLM and small model being trained, respectively

$$\mathcal{L} = -\frac{1}{n}\sum_{i}^{n}\sum_{j}^{z} softmax(\frac{p_{ij}}{T}) \log\left(softmax(\frac{q_{ij}}{T})\right) T^2$$

Num of training examples

Num of classes

Softmax function’s temperature parameter

Minimizing this loss means making the outputs of the large and the small code LLMs **as similar as possible**

Results: Effectiveness on Various LLMs

Avatar effectively optimizes CodeBERT & GraphCodeBERT on Vulnerability Prediction & Clone Detection in terms of

model size
481 MB to **3 MB** **160×** smaller

inference latency
up to **76×** faster

energy consumption
up to **184×** less

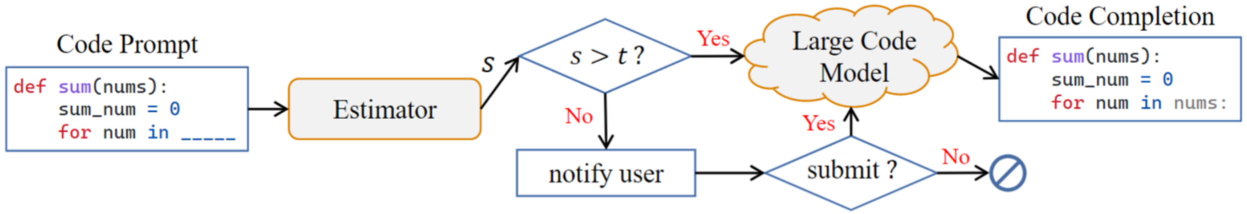
carbon footprint
up to **157×** less

efficacy
Only 1.67% loss

Efficient and Green Code LLMs: Three Strategies

Stop **Simplify** **Shrink**

FrugalCoder: Identify & Reject Unpromising Code Prompts



- Using lightweight estimator to estimate the quality of the code completion
- Decide whether to proceed based on a pre-defined threshold

Challenge 1: Efficacy
The estimator should effectively estimate code completion quality

Challenge 2: Efficiency
The cost of running the estimator should be lower than the LLM cost

SimPy: A proof-of-concept AI-agent oriented grammar

Python — Tokenized by CodeBERT — **128** tokens

```
def two_sum(nums: list[int], target: int) -> list[int]:\n
    chk_map: dict[int, int] = {}\n
    for index, val in enumerate(nums):\n
        compl = target - val\n
        if compl in chk_map:\n
            return [chk_map[compl], index]\n
        chk_map[val] = index\n
    return []
```

⇕ **Same Execution Results**

SimPy — **80** tokens

```
<def_stmt>two_sum nums:list[int] target:int<arrow>list
[int]<block_start>chk_map:dict[int int]={}<for_stmt>
index,val enumerate(nums)<block_start>compl=target-val
<if_stmt>compl<in>chk_map<block_start><return>[chk_map
[compl] index]<block_end>chk_map[val]=index<block_end>
<return>[]<block_end>
```

Replace notations with tokens
Replace keywords and symbols (e.g., “if”, “for”, etc.) with specialized tokens

Restrict coding style
Streamline white spaces, line breaks, indents, etc. preserving only essential separators

Simplify grammar tokens
For every grammar token in every production, we review whether it can be removed, merged with others, or replaced with white spaces

Optimize Code LLMs with *Compressor* & *Avatar*

ASE 2022 Compressor

Compressing Pre-trained Models of Code into 3 MB
Jieke Shi, Zhou Yang, Bowen Xu*, Hong Jin Kang and David Lo
School of Computing and Information Systems
Singapore Management University
{jiekeshi,zyang,bowenxu.2017,hjkang.2018,davidlo}@smu.edu.sg

First work to compress code LLMs: **160× smaller** and **4.23× faster**

Nominated for ACM SIGSOFT Distinguished Paper Award

Today’s Sharing

ICSE 2024 Avatar

Greening Large Language Models of Code
Jieke Shi[◇], Zhou Yang[◇], Hong Jin Kang[♠], Bowen Xu[♣], Junda He[◇], and David Lo[◇]
[◇]School of Computing and Information Systems, Singapore Management University, Singapore
[♠]Department of Computer Science, University of California, Los Angeles, USA
[♣]Department of Computer Science, North Carolina State University, Raleigh, USA
{jiekeshi, zyang, jundahe, davidlo}@smu.edu.sg, hjkang@cs.ucla.edu, bxu22@ncsu.edu

Compress code LLMs: **160x smaller, 76× faster, 184× more energy-saving**, and **157× less in carbon footprint**

Thank you!

Questions? Comments? Advice?

davidlo@smu.edu.sg