Machine Programming

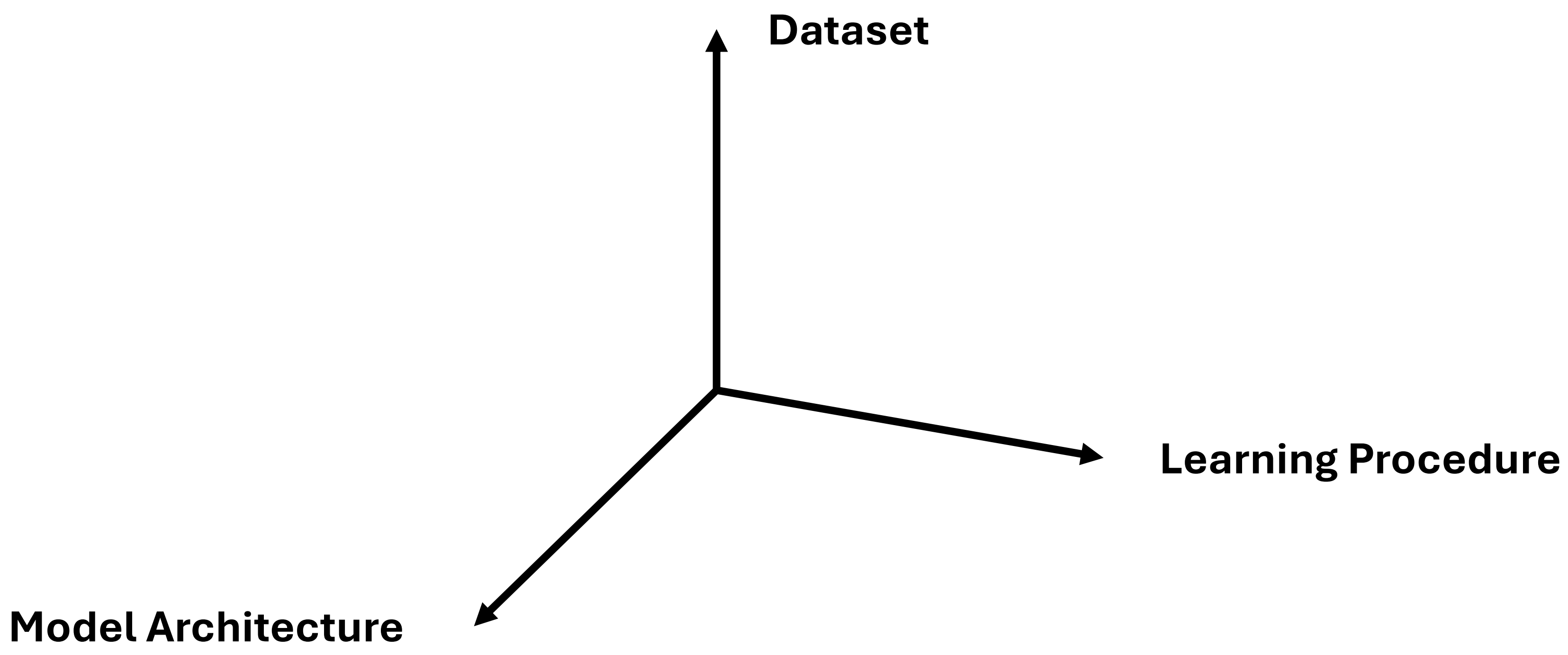
Lecture 13 – Pre-training and Evaluation of Coding Language Models

Ziyang Li

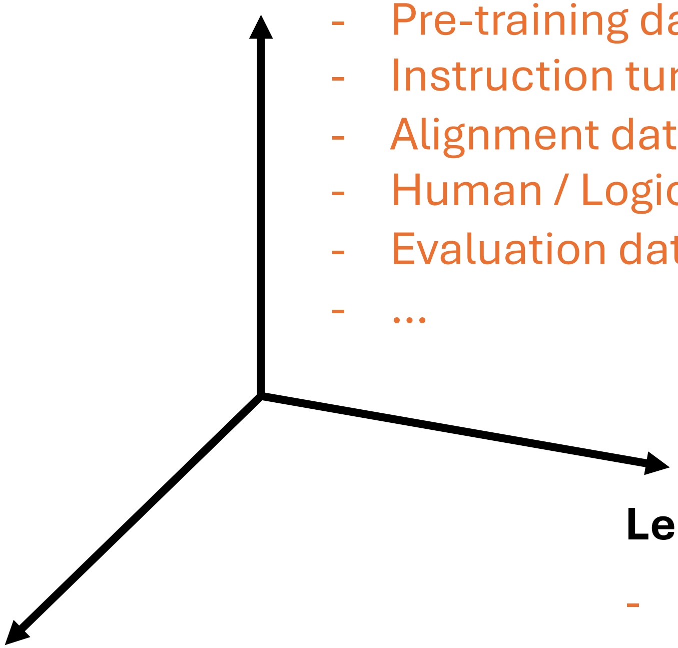
Logistics – Week 7

- Assignment 3: Coding Agents
  - Due: Oct 23
- Oral presentation sign up sheet
  - Sent out during the weekend
  - Oral presentation starting on Week 9
- Forming groups for your final projects!
  - Sign up form will be sent out on Thursday
  - Form a group of 2-3 before Next Thursday (Oct 16)

How to obtain a “good enough” LLM



How to obtain a “good enough” LLM



Dataset

- Pre-training dataset / Fine-tuning dataset
- Instruction tuning dataset
- Alignment dataset
- Human / Logical feedback dataset
- Evaluation dataset
- ...

Learning Procedure

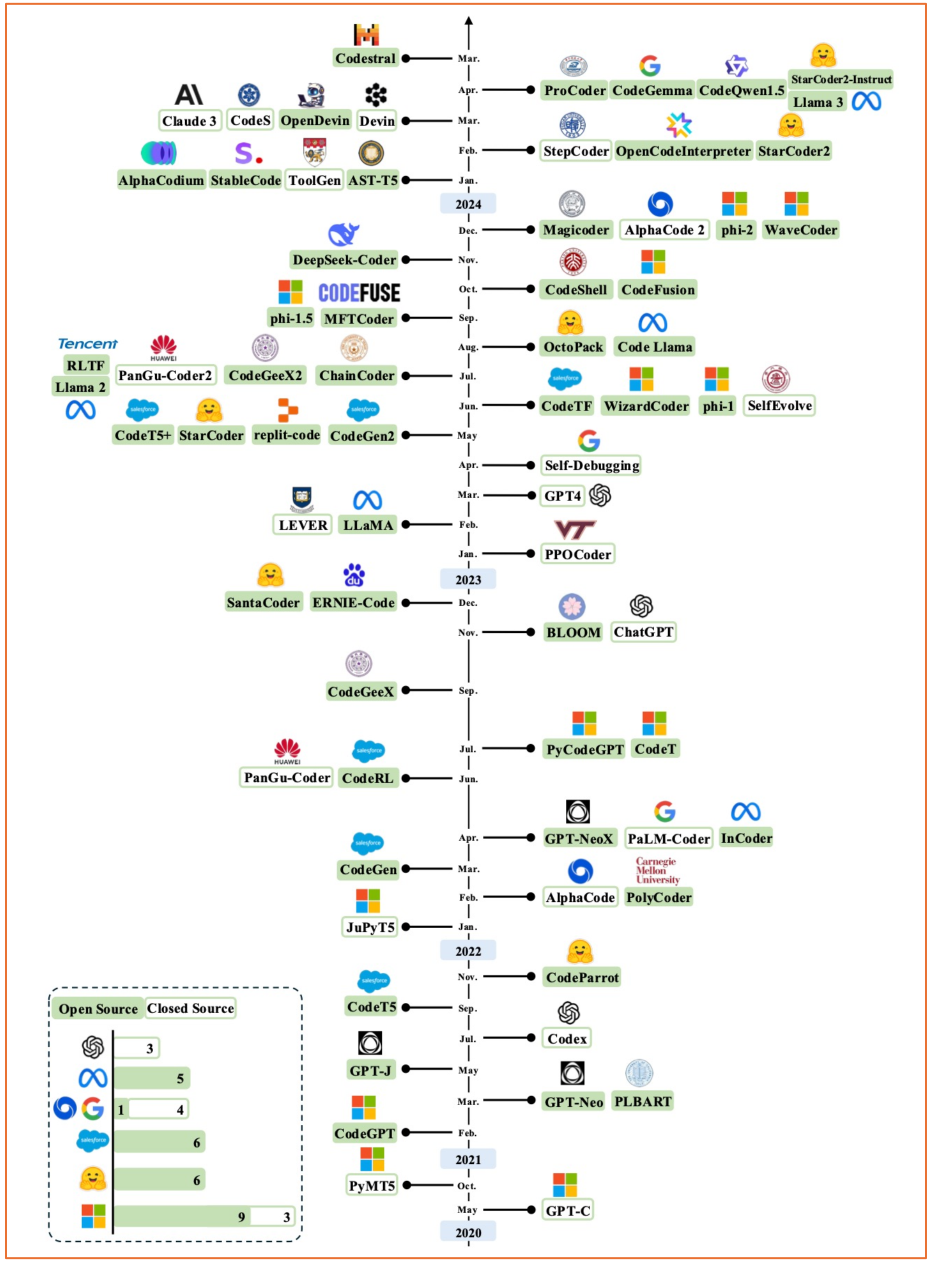
- Optimization objectives
- Learning algorithm (SFT, RL, etc.)
- Continual learning, Curriculum learning
- Staged learning
- ...

Model Architecture

- Encoder-decoder models
- Decoder-only models
- Hyper-parameter tuning
- ...

Language Models

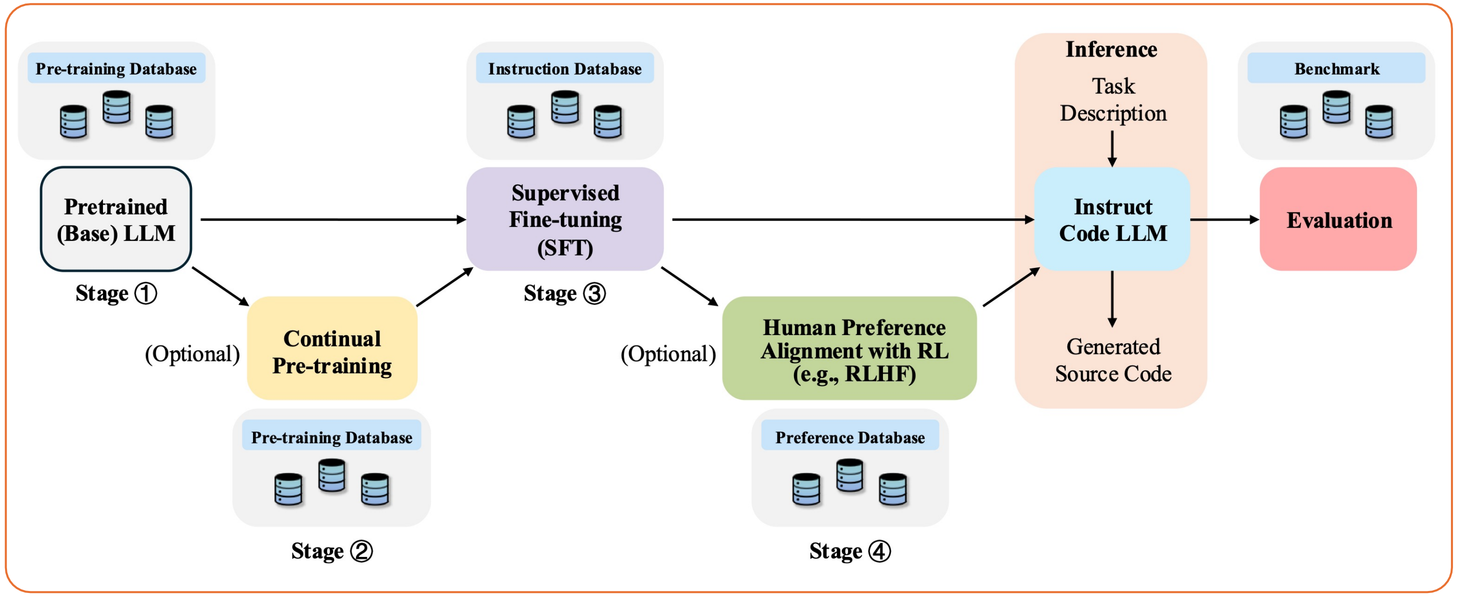
- General purpose ones
  - GPT series (3, 3.5, 4, 4.5, 5, o1, o3)
  - Gemini series (1, 1.5, 2, 2.5) / Gemma
  - Llama series (1, 2, 3, 3.1)
  - Claude series (3, sonnet-4)
  - DeepSeek series (v1, v2, v3)
- Specialized for:
  - Reasoning: deepseek-r1
  - Coding: Code Llama, DeepSeek Coder



A Survey on Large Language Models for Code Generation, Jiang et al., 2024

Today's Agenda

- Pre-training stage
  - ~~Model architecture~~
  - ~~Pre-training dataset~~
  - Learning objectives
  - Optimization
  - Evaluation dataset



Pre-training: Learning Objectives

- Causal Language Modeling
  - Next token prediction
  - Infilling
- Auxiliary pre-training tasks
  - Masked token prediction
  - (Coding) Masked identifier prediction
  - (Coding) Identifier tagging
  - (Coding) Text-code matching
  - (Coding) Text-code contrastive learning

Pre-training: Learning Objectives

- Learning Objective (Machine Learning 101)
  - Loss function $\mathcal{L}(\mathbf{x};\theta)$ where θ is the model parameter

$$\theta = \mathrm{argmin}_{\theta} \sum_{\mathbf{x}\in\mathcal{D}} \mathcal{L}(\mathbf{x};\theta)$$

Pre-training: Learning Objectives

- Learning Objective (Machine Learning 101)
  - Loss function $\mathcal{L}(\mathbf{x};\theta)$ where θ is the model parameter

$$\theta = \mathrm{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x};\theta)$$

- Next-token prediction

$$\mathcal{L}(\mathbf{x};\theta) = \sum_{i=1}^{n} -\log P_{\theta}(x_i \mid \mathbf{x}_{<i})$$

Pre-training: Learning Objectives

- Learning Objective (Machine Learning 101)
  - Loss function $\mathcal{L}(\mathbf{x};\theta)$ where θ is the model parameter

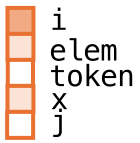
$$\theta = \operatorname{argmin}_\theta \sum_{\mathbf{x}\in\mathcal{D}} \mathcal{L}(\mathbf{x};\theta)$$

- Next-token prediction

$$\mathcal{L}(\mathbf{x};\theta) = \sum_{i=1}^{n} -\log P_\theta(x_i \mid \mathbf{x}_{<i})$$

- Example `[for, i, in, range, (, 10, ), :, print, (]` `i`

$\mathbf{x}_{<11}$ x_i



$P_\theta(x_i \mid \mathbf{x}_{<i})$

Pre-training: Learning Objectives

- Next-token prediction
  - Taking prefix $\mathbf{x}_{<i}$ and predict the next token x_i
  - But what about code editing happening in the middle?

```
impl NodeVisitor<Variable> for LocalTypingContext {
    fn visit(&mut self, node: &Variable) {
        // Collect the variable
        if let Some(local_path: String) = FIRPath::from_ast(path: node.name()).local_path() {
            self &mut LocalTypingContext
                .variables HashMap<String, Vec<NodeLocation>>
                .entry(key: local_path) Entry<'_, String, Vec<NodeLocation>>
                .or_insert(default: vec![]) &mut Vec<NodeLocation>
                .push(node.location().clone());
        }

        let path = FIRPath::from_ast(node.name());

        // Add the variable constraint to the context
        self.constraints.push(TypeConstraint::Variable {
            node: node.location().clone(),
            variable: FIRPath::from_ast(path: node.name()),
        });
    }
}
```

Pre-training: Learning Objectives

- Next-token prediction
  - Taking prefix $\mathbf{x}_{<i}$ and predict the next token x_i
  - But what about code editing happening in the middle?
- Infilling / Fill-in-the-Middle (FIM)
  - Assume prefix $\mathbf{x}_{<i}$ and suffix $\mathbf{x}_{>j}$, predict the middle infill $\mathbf{x}_{i:j}$
  - Idea: reduce the problem of infilling to next-token prediction

deepseek

DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence

Daya Guo*[1], Qihao Zhu*[1,2], Dejian Yang[1], Zhenda Xie[1], Kai Dong[1], Wentao Zhang[1]
Guanting Chen[1], Xiao Bi [1], Y. Wu[1], Y.K. Li[1], Fuli Luo[1], Yingfei Xiong[2], Wenfeng Liang[1]

[1]**DeepSeek-AI**
[2]**Key Lab of HCST (PKU), MOE; SCS, Peking University**
`{zhuqh, guodaya}@deepseek.com`

`https://github.com/deepseek-ai/DeepSeek-Coder` deepseek

DeepSeek-Coder: When the Large Language Model Meets Programming - The Rise of Code Intelligence

Daya Guo*[1], Qihao Zhu*[1,2], Dejian Yang[1], Zhenda Xie[1], Kai Dong[1], Wentao Zhang[1]
Guanting Chen[1], Xiao Bi [1], Y. Wu[1], Y.K. Li[1], Fuli Luo[1], Yingfei Xiong[2], Wenfeng Liang[1]

In our implementation, we have introduced three sentinel tokens specifically for this task. For each code file, we initially divide its content into three segments, denoted as f_{pre}, f_{middle}, and f_{suf}. Using the PSM mode, we construct the training example as follows:

$$<|\texttt{fim\_start}|>f_{pre}<|\texttt{fim\_hole}|>f_{suf}<|\texttt{fim\_end}|>f_{middle}<|\texttt{eos\_token}|>$$

We implement the Fill-in-the-Middle (FIM) method at the document level before the packing process, as proposed in the original work by Bavarian et al. (2022). This is done with an FIM rate of 0.5, following the PSM mode.

Infilling / Fill-in-the-Middle (FIM)

```rust
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

Infilling / Fill-in-the-Middle (FIM)

```
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

Infilling / Fill-in-the-Middle (FIM)

```rust
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

```rust
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
```

```rust
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

```rust
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
```

Infilling / Fill-in-the-Middle (FIM)

```rust
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

→

<FIM_START>

```rust
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
```

<FIM_HOLE>

```rust
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

<FIM_END>

```rust
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
```

<EOS>

Infilling / Fill-in-the-Middle (FIM)

- A single data-point can be augmented into multiple data-points for in-filling
- Suits modern developer workflow nicely:
  - Developer may be working on an existing file
  - Developer wants to change a function or edit a part of the file
- Question:
  - Where do we slice the program?

<FIM_START>

```
impl TypeVarAllocator {
  pub fn new() -> TypeVarAllocator {
    TypeVarAllocator { next_id: 1 }
  }

  pub fn next_id(&mut self) -> FIRUnifVarId {
```

<FIM_HOLE>

```
  }

  pub fn reset(&mut self) {
    self.next_id = 1;
  }
}
```

<FIM_END>

```
    let id = FIRUnifVarId(self.next_id);
    self.next_id += 1;
    id
```

<EOS>

Improving FIM Code Completions via Context & Curriculum Based Learning

Hitesh Sagtani
Sourcegraph Inc
Bengaluru, India
sagtanih@gmail.com

Rishabh Mehrotra*
Pavo AI
London, UK
erishabh@gmail.com

Beyang Liu
Sourcegraph Inc
San Francisco, USA
beyang@sourcegraph.com

Improving FIM Code Completions via Context & Curriculum Based Learning, Sagtani et. al., 2025

Improving FIM Code Completions via Context & Curriculum Based Learning

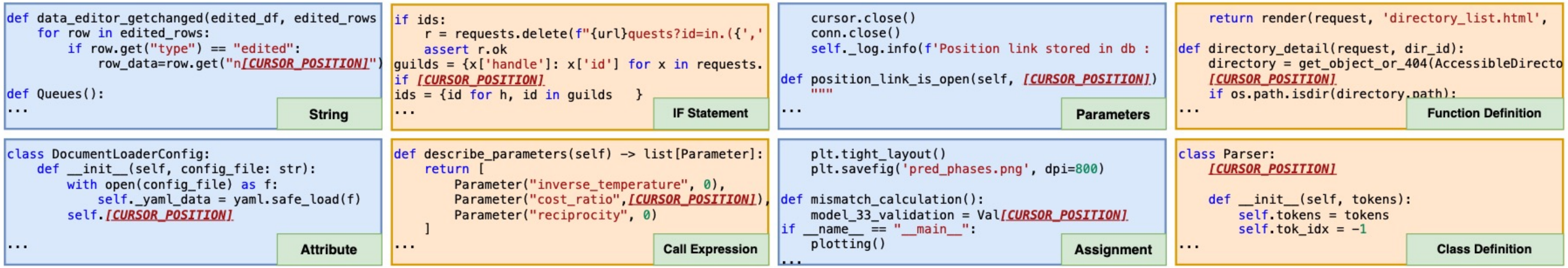


Figure 1: Illustrative examples of various AST node types. The cursor position highlights the position where completions are triggered, with the node type indicated at the bottom right of each example.

Improving FIM Code Completions via Context & Curriculum Based Learning, Sagtani et. al., 2025

Improving FIM Code Completions via Context & Curriculum Based Learning

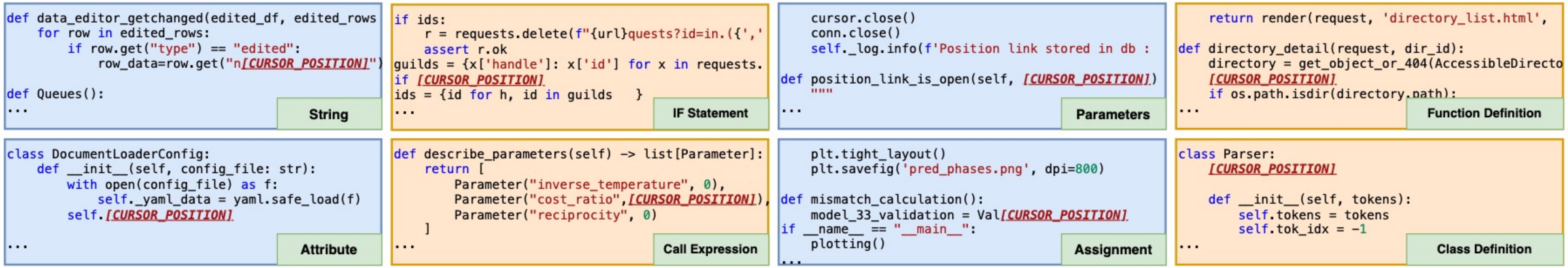


Figure 1: Illustrative examples of various AST node types. The cursor position highlights the position where completions are triggered, with the node type indicated at the bottom right of each example.

Q: How do we detect these AST node types?

Improving FIM Code Completions via Context & Curriculum Based Learning

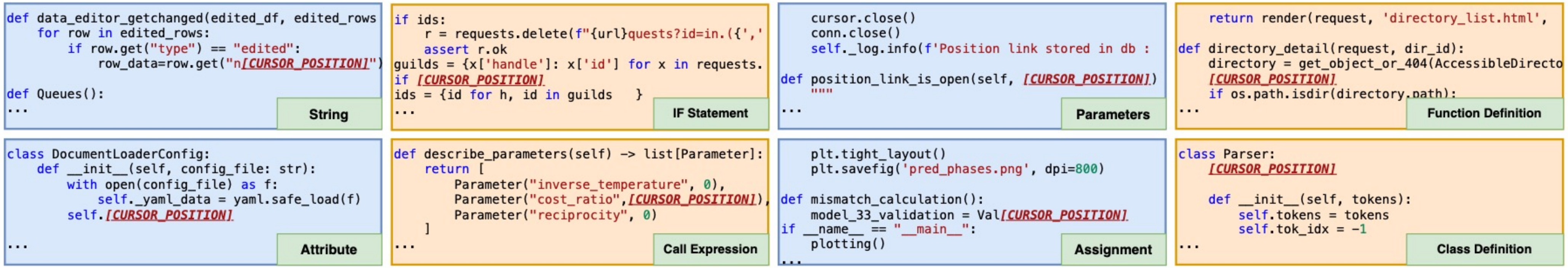


Figure 1: Illustrative examples of various AST node types. The cursor position highlights the position where completions are triggered, with the node type indicated at the bottom right of each example.

Q: How do we detect these AST node types?

A: Using **parser** and **static analyzers**!

Improving FIM Code Completions via Context & Curriculum Based Learning, Sagtani et. al., 2025

Task: “In a Python program, randomly find a body of an if/elif/else statement and make it a prefix-infill-suffix datapoint for FIM training”

```
stmt = FunctionDef(identifier name, arguments args,
                   stmt* body, expr* decorator_list, expr? returns,
                   string? type_comment, type_param* type_params)
      | AsyncFunctionDef(identifier name, arguments args,
                         stmt* body, expr* decorator_list, expr? retu
                         string? type_comment, type_param* type_param

      | ClassDef(identifier name,
         expr* bases,
         keyword* keywords,
         stmt* body,
         expr* decorator_list,
         type_param* type_params)
      | Return(expr? value)

      | Delete(expr* targets)
      | Assign(expr* targets, expr value, string? type_comment)
      | TypeAlias(expr name, type_param* type_params, expr value)
      | AugAssign(expr target, operator op, expr value)
      -- 'simple' indicates that we annotate simple name without pare
      | AnnAssign(expr target, expr annotation, expr? value, int simp

      -- use 'orelse' because else is a keyword in target languages
      | For(expr target, expr iter, stmt* body, stmt* orelse, string?
      | AsyncFor(expr target, expr iter, stmt* body, stmt* orelse, st
      | While(expr test, stmt* body, stmt* orelse)
      | If(expr test, stmt* body, stmt* orelse)
      | With(withitem* items, stmt* body, string? type_comment)
      | AsyncWith(withitem* items, stmt* body, string? type_comment)

      | Match(expr subject, match_case* cases)

      | Raise(expr? exc, expr? cause)
      | Try(stmt* body, excepthandler* handlers, stmt* orelse, stmt*
      | TryStar(stmt* body, excepthandler* handlers, stmt* orelse, st
      | Assert(expr test, expr? msg)

      | Import(alias* names)
      | ImportFrom(identifier? module, alias* names, int? level)

      | Global(identifier* names)
      | Nonlocal(identifier* names)
      | Expr(expr value)
      | Pass | Break | Continue

      -- col_offset is the byte offset in the utf8 string the parser
      attributes (int lineno, int col_offset, int? end_lineno, int? e

      -- BoolOp() can use left & right?
expr = BoolOp(boolop op, expr* values)
     | NamedExpr(expr target, expr value)
     | BinOp(expr left, operator op, expr right)
     | UnaryOp(unaryop op, expr operand)
     | Lambda(arguments args, expr body)
     | IfExp(expr test, expr body, expr orelse)
     | Dict(expr* keys, expr* values)
     | Set(expr* elts)
     | ListComp(expr elt, comprehension* generators)
     | SetComp(expr elt, comprehension* generators)
     | DictComp(expr key, expr value, comprehension* generators)
     | GeneratorExp(expr elt, comprehension* generators)
     -- the grammar constrains where yield expressions can occur
     | Await(expr value)
     | Yield(expr? value)
     | YieldFrom(expr value)
     -- need sequences for compare to distinguish between
     -- x < 4 < 3 and (x < 4) < 3
     | Compare(expr left, cmpop* ops, expr* comparators)
     | Call(expr func, expr* args, keyword* keywords)
     | FormattedValue(expr value, int conversion, expr? format_spec)
     | JoinedStr(expr* values)
     | Constant(constant value, string? kind)

     -- the following expression can appear in assignment context
     | Attribute(expr value, identifier attr, expr_context ctx)
     | Subscript(expr value, expr slice, expr_context ctx)
     | Starred(expr value, expr_context ctx)
     | Name(identifier id, expr_context ctx)
     | List(expr* elts, expr_context ctx)
     | Tuple(expr* elts, expr_context ctx)

     -- can appear only in Subscript
     | Slice(expr? lower, expr? upper, expr? step)

      -- col_offset is the byte offset in the utf8 string the parser
      attributes (int lineno, int col_offset, int? end_lineno, int? e

expr_context = Load | Store | Del

boolop = And | Or

operator = Add | Sub | Mult | MatMult | Div | Mod | Pow | LShift
             | RShift | BitOr | BitXor | BitAnd | FloorDiv

unaryop = Invert | Not | UAdd | USub

cmpop = Eq | NotEq | Lt | LtE | Gt | GtE | Is | IsNot | In | NotIn

comprehension = (expr target, expr iter, expr* ifs, int is_async)
```

Task: "In a Python program, randomly find a body of an if/elif/else statement and make it a prefix-infill-suffix datapoint for FIM training"

https://docs.python.org/3/library/ast.html

```python
import ast

class IfBodyCollector(ast.NodeVisitor):
  def __init__(self):
    self.bodies = [] # (label, first_node, last_node)

  def visit_If(self, node: ast.If):
    # the main "if" body
    if node.body:
      self.bodies.append(("if", node.body[0], node.body[-1]))

    # walk the elif/else chain in orelse
    cur = node
    while True:
      if not cur.orelse: break
      if len(cur.orelse) == 1 and isinstance(cur.orelse[0], ast.If):
        # this is an "elif"
        e = cur.orelse[0]
        if e.body:
          self.bodies.append(("elif", e.body[0], e.body[-1]))
          cur = e
          continue
        else:
           # this is the terminal "else" (a list of statements)
           first = cur.orelse[0]
           last = cur.orelse[-1]
           self.bodies.append(("else", first, last))
           break

    # keep descending to catch nested ifs
    self.generic_visit(node)
```

```python
tree = ast.parse(src)
collector = IfBodyCollector()
collector.visit(tree)
```

Task: “In a Python program, randomly find a body of an if/elif/else statement and make it a prefix-infill-suffix datapoint for FIM training”

```python
import ast

class IfBodyCollector(ast.NodeVisitor):
  def __init__(self):
    self.bodies = [] # (label, first_node, last_node)

  def visit_If(self, node: ast.If):
    # the main "if" body
    if node.body:
      self.bodies.append(("if", node.body[0], node.body[-1]))

    # walk the elif/else chain in orelse
    cur = node
    while True:
      if not cur.orelse: break
      if len(cur.orelse) == 1 and isinstance(cur.orelse[0], ast.If):
        # this is an "elif"
        e = cur.orelse[0]
        if e.body:
          self.bodies.append(("elif", e.body[0], e.body[-1]))
          cur = e
          continue
        else:
           # this is the terminal "else" (a list of statements)
           first = cur.orelse[0]
           last = cur.orelse[-1]
           self.bodies.append(("else", first, last))
           break

    # keep descending to catch nested ifs
    self.generic_visit(node)
```

```python
tree = ast.parse(src)
collector = IfBodyCollector()
collector.visit(tree)
```

Python abstract syntax tree (AST) node visitor

We want the visitor to visit If statements

Collect the body of “if”

Collect the body of “elif”

Collect the body of “else”

Run the if-body collector

Task: “In a Python program, randomly find a body of an if/elif/else statement and make it a prefix-infill-suffix datapoint for FIM training”

Code Llama: Open Foundation Models for Code

Baptiste Rozière[†], Jonas Gehring[†], Fabian Gloeckle[†,*], Sten Sootla[†], Itai Gat, Xiaoqing Ellen Tan, Yossi Adi[◇], Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, Gabriel Synnaeve[†]

Meta AI

Code Llama: Open Foundation Models for Code

Bapt
Ellen
Kozh
Grat
Louis

2.3 Infilling

Code infilling is the task of predicting the missing part of a program given a surrounding context. Applications include code completion at the cursor's position in code IDEs, type inference and generation of in-code documentation (e.g., docstrings).

We train infilling models following the concept of causal masking (Aghajanyan et al., 2022; Fried et al., 2023), where parts of a training sequence are moved to the end, and the reordered sequence is predicted autoregressively. We train the general-purpose 7B, 13B and 70B models with an infilling objective, following the recommendations of Bavarian et al. (2022). More precisely, we split training documents at the character level into a prefix, a middle part and a suffix with the splitting locations sampled independently from a uniform distribution over the document length. We apply this transformation with a probability of 0.9 and to documents that are not cut across multiple model contexts only. We randomly format half of the splits in the *prefix-suffix-middle* (PSM) format and the other half in the compatible *suffix-prefix-middle (SPM)* format described in Bavarian et al. (2022, App. D). We extend Llama 2's tokenizer with four special tokens that mark the beginning of the prefix, the middle part or the suffix, and the end of the infilling span. To limit the distribution shift between autoregressive and infilling training, we suppress the implicit leading space that SentencePiece tokenizers add upon encoding the middle part and the suffix (Kudo & Richardson, 2018). In SPM format, we concatenate the prefix and the middle part before encoding to tokens. Note that our model doesn't encounter split subtokens in the SPM format while it does in the PSM format.

Results on the effect of infilling training on downstream generation tasks and the performance of our infilling models on infilling benchmarks are reported in Section 3.2.

Code Llama: Open Foundation Models for Code

Baptis … at, Xiaoqing
Ellen … pin, Artyom
Kozhe
Gratta
Louis

2.3 Infilling

Code infilling is the task of predicting the missing part of a program given a surrounding context. Applications include code completion at the cursor's position in code IDEs, type inference and generation of in-code documentation

We train infilli
2023), where pa
autoregressively
the recommenda
level into a pre
uniform distribu
to documents th
the *prefix-suffix*
described in Ba
mark the beginn
distribution shi
SentencePiece t
SPM format, we
doesn't encount

Results on the e
models on infill

Model	FIM	Size	HumanEval			MBPP			Test loss
			pass@1	pass@10	pass@100	pass@1	pass@10	pass@100	
Code Llama (w/o LCFT)	✗	7B	33.2%	43.3%	49.9%	44.8%	52.5%	57.1%	0.408
		13B	36.8%	49.2%	57.9%	48.2%	57.4%	61.6%	0.372
Code Llama (w/o LCFT)	✓	7B	33.6%	44.0%	48.8%	44.2%	51.4%	55.5%	0.407
		13B	36.2%	48.3%	54.6%	48.0%	56.8%	60.8%	0.373
Absolute gap	✗ - ✓	7B	−0.4%	−0.7%	1.1%	0.6%	1.1%	1.6%	0.001
		13B	0.7%	0.9%	3.3%	0.2%	0.6%	0.8%	−0.001

Table 5: **Comparison of models with and without FIM training.** pass@1, pass@10 and pass@100 scores on HumanEval and MBPP evaluated at temperature 0.1 for models trained with and without infilling (FIM) objective. Infilling training incurs no cost on autoregressive test set loss, but a small cost on HumanEval and MBPP pass@k metrics that is aggravated at higher sample counts k. The models are compared prior to long context fine-tuning (LCFT).

Code Llama: Open Foundation Models for Code

2.3 Infilling

Baptis ... at, Xiaoqing
Ellen ... pin, Artyom
Kozhe
Gratta
Louis

Code infilling is the task of predicting the missing part of a program given a surrounding context. Applications include code completion at the cursor's position in code IDEs, type inference and generation of in-code documentation

We train infilli
2023), where pa
autoregressively
the recommenda
level into a pre
uniform distribu
to documents th
the *prefix-suffix-*
described in Ba
mark the beginn
distribution shi
SentencePiece t
SPM format, we
doesn't encount

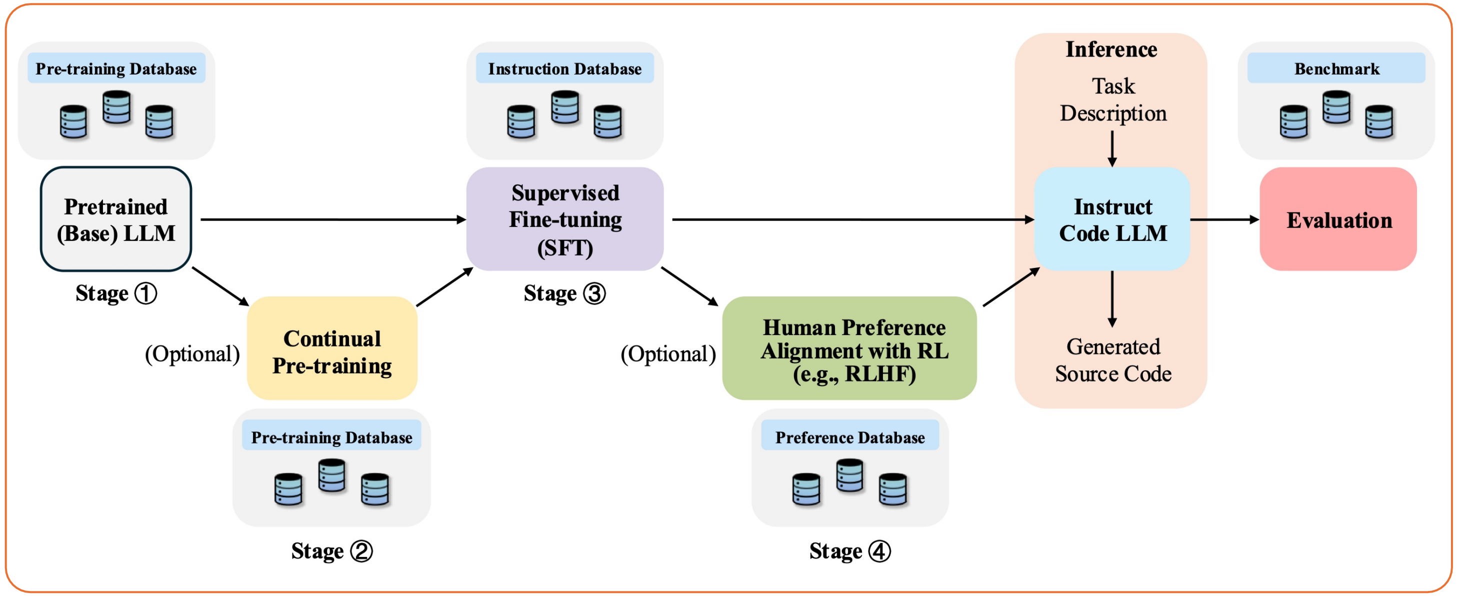
Results on the e
models on infilli

Model	FIM	Size	HumanEval pass@1	HumanEval pass@10	HumanEval pass@100	MBPP pass@1	MBPP pass@10	MBPP pass@100	Test loss
Code Llama (w/o LCFT)	✗	7B	33.2%	43.3%	49.9%	44.8%	52.5%	57.1%	0.408
		13B	36.8%	49.2%	57.9%	48.2%	57.4%	61.6%	0.372
Code Llama (w/o LCFT)	✓	7B	33.6%	44.0%	48.8%	44.2%	51.4%	55.5%	0.407
		13B	36.2%	48.3%	54.6%	48.0%	56.8%	60.8%	0.373
Absolute gap	✗ - ✓	7B	−0.4%	−0.7%	1.1%	0.6%	1.1%	1.6%	0.001
		13B	0.7%	0.9%	3.3%	0.2%	0.6%	0.8%	−0.001

Table 5: **Comparison of models with and without FIM training.** pass@1, pass@10 and pass@100 scores on HumanEval and MBPP evaluated at temperature 0.1 for models trained with and without infilling (FIM) objective. Infilling training incurs no cost on autoregressive test set loss, but a small cost on HumanEval and MBPP pass@k metrics that is aggravated at higher sample counts k. The models are compared prior to long context fine-tuning (LCFT).

Today's Agenda

- Pre-training stage
  - ~~Model architecture~~
  - ~~Pre-training dataset~~
  - ~~Learning objectives~~
  - Optimization
  - Evaluation dataset
- Post-training stage
  - Supervised fine-tuning
  - Reinforcement learning



Pre-training: Optimizations

- Learning Objective (Machine Learning 101)
  - Loss function $\mathcal{L}(\mathbf{x};\theta)$ where θ is the model parameter

$$\theta = \operatorname{argmin}_{\theta} \sum_{\mathbf{x}\in\mathcal{D}} \mathcal{L}(\mathbf{x};\theta)$$

- Next-token prediction

$$\mathcal{L}(\mathbf{x};\theta) = \sum_{i=1}^{n} -\log P_{\theta}(x_i \mid \mathbf{x}_{<i})$$

Pre-training: Optimizations

- Learning Objective (Machine Learning 101)
  - Loss function $\mathcal{L}(\mathbf{x};\theta)$ where θ is the model parameter

$$\theta = \text{argmin}_{\theta} \sum_{\mathbf{x}\in\mathcal{D}} \mathcal{L}(\mathbf{x};\theta)$$

- Next-token prediction

$$\mathcal{L}(\mathbf{x};\theta) = \sum_{i=1}^{n} -\log P_{\theta}(x_i \mid \mathbf{x}_{<i})$$

- Loss function $\mathcal{L}$: Negative-log likelihood (NLL) loss

Pre-training: Optimizations

$$\theta = \text{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Pre-training: Optimizations

$$\theta = \mathrm{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Gradient: $g_t = \nabla_{\theta} \mathcal{L}(\theta)$

Pre-training: Optimizations

$$\theta = \text{argmin}_\theta \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the “optimal” set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Gradient: $g_t = \nabla_\theta \mathcal{L}(\theta)$

Optimizer: SGD, Adam, AdamW, Momentum

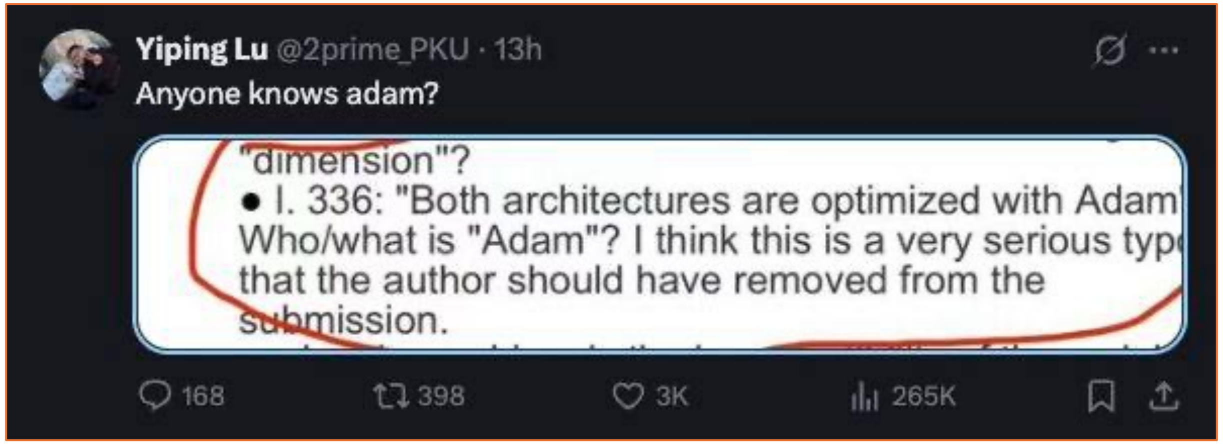
Pre-training: Optimizations

$$\theta = \mathrm{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Gradient: $g_t = \nabla_{\theta} \mathcal{L}(\theta)$

Optimizer: SGD, Adam, AdamW, Momentum



Pre-training: Optimizations

$$\theta = \operatorname{argmin}_\theta \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Gradient (at step t): $g_t = \nabla_\theta \mathcal{L}(\theta_t)$

Optimizer: SGD, Adam, AdamW, Momentum

First momentum: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Second momentum: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Weight Update: $\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda \theta_t \right)$

Pre-training: Optimizations

$$\theta = \text{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Gradient (at step t): $g_t = \nabla_\theta \mathcal{L}(\theta_t)$

Optimizer: SGD, Adam, AdamW, Momentum

First momentum: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Second momentum: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Weight Update: $\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda \theta_t \right)$

Learning Rate (LR)

Weight decay

Pre-training: Optimizations

$$\theta = \operatorname{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - Batching & parallelism

Gradient (at step t): $g_t = \nabla_{\theta} \mathcal{L}(\theta_t)$

Optimizer: SGD, Adam, AdamW, Momentum

First momentum: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Second momentum: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Weight Update: $\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda \theta_t \right)$

Learning Rate Scheduling: η_t

Cosine decay: $\eta_t = \eta_{\min} + 0.5(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{\pi t}{T}))$

Pre-training: Optimizations

$$\theta = \text{argmin}_\theta \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
    - Back-propagation
    - Optimizers
    - Learning rates & schedulers
    - Batching & parallelism

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

Gradient (at step t): $g_t = \nabla_\theta \mathcal{L}(\theta_t)$

Optimizer: SGD, Adam, AdamW, Momentum

First momentum: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Second momentum: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Weight Update: $\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)$

Learning Rate Scheduling: η_t

Cosine decay: $\eta_t = \eta_{\min} + 0.5(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{\pi t}{T}))$

The Llama 3 Herd of Models, Llama Team, AI@Meta, 2024

Pre-training: Optimizations

$$\theta = \text{argmin}_\theta \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning rates & schedulers
  - $\boxed{\text{Batching \& parallelism}}$

Gradient (at step t): $g_t = \nabla_\theta \mathcal{L}(\theta_t)$

Optimizer: SGD, Adam, AdamW, Momentum

First momentum: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Second momentum: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Weight Update: $\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right)$

Learning Rate Scheduling: η_t

Cosine decay: $\eta_t = \eta_{\min} + 0.5(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{\pi t}{T}))$

Batching: $B \subset D$

Mini-batched gradient: $g_t = \frac{1}{|B|} \sum_{\mathbf{x} \in B} \nabla_\theta \mathcal{L}(\mathbf{x}; \theta_t)$

Pre-training: Optimizations

$$\theta = \text{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optimizers
  - Learning ra
  - Batching &

Gradient (at step t): $g_t = \nabla_\theta \mathcal{L}(\theta_t)$

Optimizer: SGD, Adam, AdamW, Momentum

Batching: $B \subset D$

Mini-batched gradient: $g_t = \frac{1}{|B|} \sum_{\mathbf{x} \in B} \nabla_\theta \mathcal{L}(\mathbf{x}; \theta_t)$

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	8	131,072	16	16M	380	38%

Table 4 Scaling configurations and MFU for each stage of Llama 3 405B pre-training. See text and Figure 5 for descriptions of each type of parallelism.

The Llama 3 Herd of Models, Llama Team, AI@Meta, 2024

Pre-training: Optimizations

$$\theta = \text{argmin}_\theta \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

- How do we find the "optimal" set of parameters θ?
  - Back-propagation
  - Optim
  - Learning ra
  - Batching &

Gradient (at step t): $g_t = \nabla_\theta \mathcal{L}(\theta_t)$

W, Momentum

Batching: $B \subset D$

Mini-batched gradient: $g_t = \frac{1}{|B|} \sum_{\mathbf{x} \in B} \nabla_\theta \mathcal{L}(\mathbf{x}; \theta_t)$

Tensor Parallelism

Pipeline Parallelism

Batch size per data-parallel replica

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	8	131,072	16	16M	380	38%

Table 4 Scaling configurations and MFU for each stage of Llama 3 405B pre-training. See text and Figure 5 for descriptions of each type of parallelism.

The Llama 3 Herd of Models, Llama Team, AI@Meta, 2024

Pre-training: Optimizations

$$\theta = \text{argmin}_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

Tensor Parallelism

Pipeline Parallelism

Batch size per data-parallel replica

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	8	131,072	16	16M	380	38%

Table 4 Scaling configurations and MFU for each stage of Llama 3 405B pre-training. See text and Figure 5 for descriptions of each type of parallelism.

Q: How do we calculate #tokens per GPU per optimization step?

The Llama 3 Herd of Models, Llama Team, AI@Meta, 2024

Pre-training: Optimizations

$$\theta = \text{argmin}_\theta \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}(\mathbf{x}; \theta)$$

Tensor Parallelism

Pipeline Parallelism

Batch size per data-parallel replica

GPUs	TP	CP	PP	DP	Seq. Len.	Batch size/DP	Tokens/Batch	TFLOPs/GPU	BF16 MFU
8,192	8	1	16	64	8,192	32	16M	430	43%
16,384	8	1	16	128	8,192	16	16M	400	41%
16,384	8	16	16	8	131,072	16	16M	380	38%

Table 4 Scaling configurations and MFU for each stage of Llama 3 405B pre-training. See text and Figure 5 for descriptions of each type of parallelism.

Q: How do we calculate #tokens per GPU per optimization step?

A: (Seq. Len.) * (Batch size/DP) / (TP * PP)

The Llama 3 Herd of Models, Llama Team, AI@Meta, 2024

Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-training

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, Joel Hestness
Cerebras Systems
{shane.bergsma,joel}@cerebras.net

Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-training, Bergsma et. al., 2025

Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-training

Shane Bergsma, Nolan De

{s

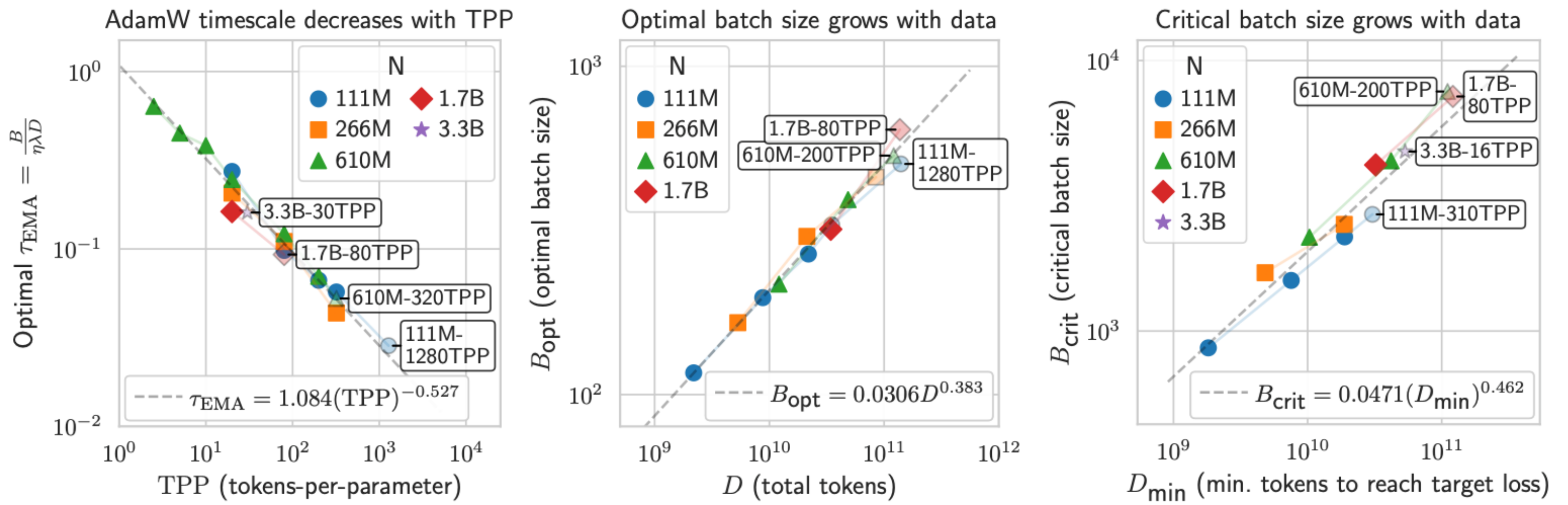
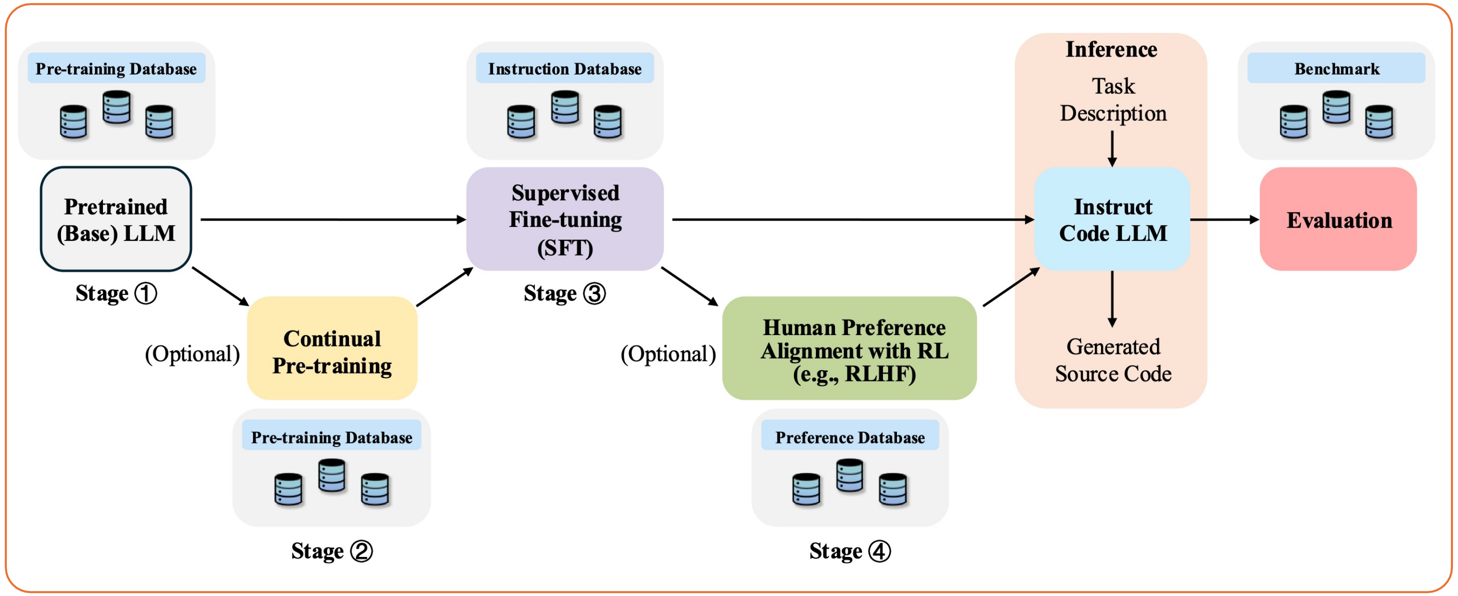


Figure 1: **Hyperparameters and their power lines**: Optimal τ_{EMA} obeys a power law in tokens-per-parameter (*left*), while optimal batch size (*middle*) and critical batch size (*right*) obey power laws in D. Faded markers indicate points not used in fitting — all fits generalize well to larger-scale runs.

Power Lines: Scaling Laws for Weight Decay and Batch Size in LLM Pre-training, Bergsma et. al., 2025

Today's Agenda

- Pre-training stage
  - ~~Model architecture~~
  - ~~Pre-training dataset~~
  - ~~Learning objectives~~
  - ~~Optimization~~
  - Evaluation dataset
- Post-training stage
  - Supervised fine-tuning
  - Reinforcement learning



Evaluation Benchmark

- Coding benchmarks can be used to evaluate LLMs' abilities

Category	Benchmark	Llama 3 8B	Gemma 2 9B	Mistral 7B	Llama 3 70B	Mixtral 8x22B	GPT 3.5 Turbo	Llama 3 405B	Nemotron 4 340B	GPT-4 (0125)	GPT-4o	Claude 3.5 Sonnet
General	MMLU (5-shot)	69.4	**72.3**	61.1	**83.6**	76.9	70.7	87.3	82.6	85.1	89.1	**89.9**
	MMLU (0-shot, CoT)	**73.0**	72.3△	60.5	**86.0**	79.9	69.8	88.6	78.7◁	85.4	**88.7**	88.3
	MMLU-Pro (5-shot, CoT)	**48.3**	–	36.9	**66.4**	56.3	49.2	73.3	62.7	64.8	74.0	**77.0**
	IFEval	**80.4**	73.6	57.6	**87.5**	72.7	69.9	**88.6**	85.1	84.3	85.6	88.0
Code	HumanEval (0-shot)	**72.6**	54.3	40.2	**80.5**	75.6	68.0	89.0	73.2	86.6	90.2	**92.0**
	MBPP EvalPlus (0-shot)	**72.8**	71.7	49.5	**86.0**	78.6	82.0	88.6	72.8	83.6	87.8	**90.5**
Math	GSM8K (8-shot, CoT)	**84.5**	76.7	53.2	**95.1**	88.2	81.6	**96.8**	92.3◇	94.2	96.1	96.4◇
	MATH (0-shot, CoT)	**51.9**	44.3	13.0	**68.0**	54.1	43.1	73.8	41.1	64.5	**76.6**	71.1
Reasoning	ARC Challenge (0-shot)	83.4	**87.6**	74.2	**94.8**	88.7	83.7	**96.9**	94.6	96.4	96.7	96.7
	GPQA (0-shot, CoT)	32.8	–	28.8	**46.7**	33.3	30.8	51.1	–	41.4	53.6	**59.4**
Tool use	BFCL	**76.1**	–	60.4	84.8	–	**85.9**	88.5	86.5	88.3	80.5	**90.2**
	Nexus	**38.5**	30.0	24.7	**56.7**	48.5	37.2	**58.7**	–	50.3	56.1	45.7
Long context	ZeroSCROLLS/QuALITY	81.0	–	–	90.5	–	–	**95.2**	–	**95.2**	90.5	90.5
	InfiniteBench/En.MC	65.1	–	–	78.2	–	–	**83.4**	–	72.1	82.5	–
	NIH/Multi-needle	98.8	–	–	97.5	–	–	98.1	–	**100.0**	**100.0**	90.8
Multilingual	MGSM (0-shot, CoT)	**68.9**	53.2	29.9	**86.9**	71.1	51.4	**91.6**	–	85.9	90.5	**91.6**

Evaluation Benchmark

Scenario	Benchmark	Size	#PL	Date	Link
General	HumanEval [48]	164	Python	2021-07	https://huggingface.co/datasets/openai_humaneval
	HumanEval+ [162]	164	Python	2023-05	https://huggingface.co/datasets/evalplus/humanevalplus
	HumanEvalPack [187]	164	6	2023-08	https://huggingface.co/datasets/bigcode/humanevalpack
	MBPP [17]	974	Python	2021-08	https://huggingface.co/datasets/mbpp
	MBPP+ [162]	378	Python	2023-05	https://huggingface.co/datasets/evalplus/mbppplus
	CoNaLa [297]	596.88K	Python	2018-05	https://huggingface.co/datasets/neulab/conala
	Spider [300]	8,034	SQL	2018-09	https://huggingface.co/datasets/xlangai/spider
	CONCODE [113]	104K	Java	2018-08	https://huggingface.co/datasets/AhmedSSoliman/CONCOD
	ODEX [273]	945	Python	2022-12	https://huggingface.co/datasets/neulab/odex
	CoderEval [299]	460	Python, Java	2023-02	https://github.com/CoderEval/CoderEval
	ReCode [263]	1,138	Python	2022-12	https://github.com/amazon-science/recode
	StudentEval [19]	1,749	Python	2023-06	https://huggingface.co/datasets/wellesley-easel/StudentEval
	BigCodeBench [333]	1,140	Python	2024-06	https://huggingface.co/datasets/bigcode/bigcodebench
	ClassEval [72]	100	Python	2023-08	https://huggingface.co/datasets/FudanSELab/ClassEval
	NaturalCodeBench [314]	402	Python, Java	2024-05	https://github.com/THUDM/NaturalCodeBench
Competitions	APPS [95]	10,000	Python	2021-05	https://huggingface.co/datasets/codeparrot/apps
	CodeContests [151]	13,610	C++, Python, Java	2022-02	https://huggingface.co/datasets/deepmind/code_contests
	LiveCodeBench [188]	713 Updating	Python	2024-03	https://github.com/LiveCodeBench/LiveCodeBench
Data Science	DSP [41]	1,119	Python	2022-01	https://github.com/microsoft/DataScienceProblems
	DS-1000 [136]	1,000	Python	2022-11	https://huggingface.co/datasets/xlangai/DS-1000
	ExeDS [107]	534	Python	2022-11	https://github.com/Jun-jie-Huang/ExeDS
Multilingual	MBXP [16]	12.4K	13	2022-10	https://huggingface.co/datasets/mxeval/mbxp
	Multilingual HumanEval [16]	1.9K	12	2022-10	https://huggingface.co/datasets/mxeval/multi-humaneval
	HumanEval-X [321]	820	Python, C++, Java, JavaScript, Go	2023-03	https://huggingface.co/datasets/THUDM/humaneval-x
	MultiPL-E [39]	161	18	2022-08	https://huggingface.co/datasets/nuprl/MultiPL-E
	xCodeEval [128]	5.5M	11	2023-03	https://github.com/ntunlp/xCodeEval
Reasoning	MathQA-X [16]	5.6K	Python, Java, JavaScript	2022-10	https://huggingface.co/datasets/mxeval/mathqa-x
	MathQA-Python [17]	23,914	Python	2021-08	https://github.com/google-research/google-research
	GSM8K [58]	8.5K	Python	2021-10	https://huggingface.co/datasets/gsm8k
	GSM-HARD [79]	1.32K	Python	2022-11	https://huggingface.co/datasets/reasoning-machines/gsm-hard
	CRUXEval [82]	800	Python	2024-01	https://huggingface.co/datasets/cruxeval-org/cruxeval
Repository	RepoEval [309]	3,573	Python, Java	2023-03	https://paperswithcode.com/dataset/repoeval
	Stack-Repo [239]	200	Java	2023-06	https://huggingface.co/datasets/RepoFusion/Stack-Repo
	Repobench [167]	27k	Python, Java	2023-01	https://github.com/Leolty/repobench
	EvoCodeBench [144]	275	Python	2024-03	https://huggingface.co/datasets/LJ0815/EvoCodeBench
	SWE-bench [123]	2,294	Python	2023-10	https://huggingface.co/datasets/princeton-nlp/SWE-bench
	CrossCodeEval [68]	10K	Python, Java, TypeScript, C#	2023-10	https://github.com/amazon-science/cceval
	SketchEval [308]	20,355	Python	2024-03	https://github.com/nl2code/codes

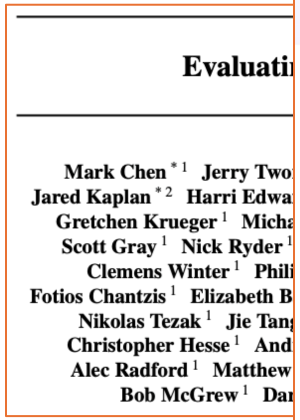
A Survey on Large Language Models for Code Generation, Jiang et al., 2024

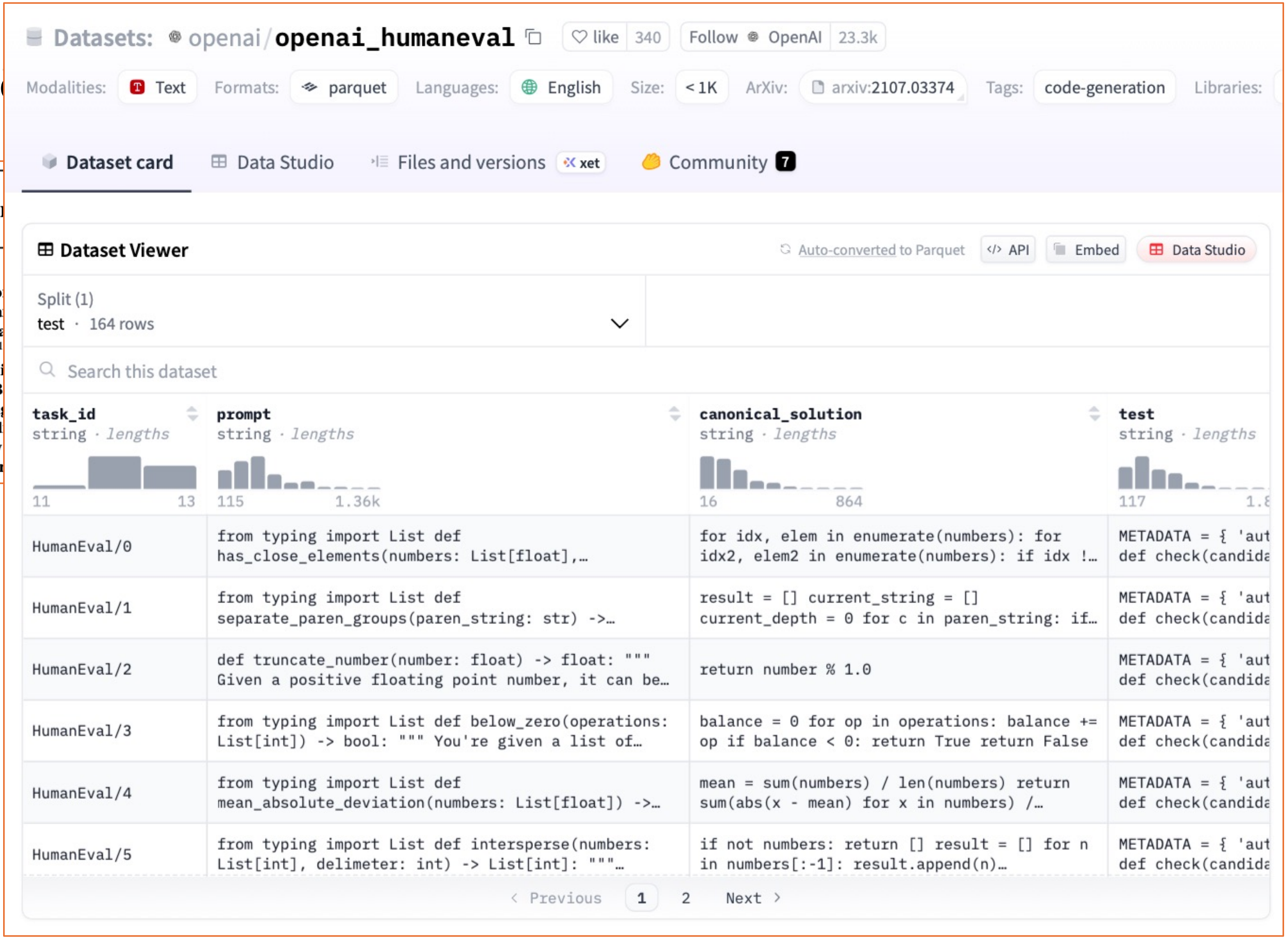
Evaluation Benchmark: HumanEval

Evaluating Large Language Models Trained on Code

Mark Chen [*1] Jerry Tworek [*1] Heewoo Jun [*1] Qiming Yuan [*1] Henrique Ponde de Oliveira Pinto [*1]
Jared Kaplan [*2] Harri Edwards [1] Yuri Burda [1] Nicholas Joseph [2] Greg Brockman [1] Alex Ray [1] Raul Puri [1]
Gretchen Krueger [1] Michael Petrov [1] Heidy Khlaaf [3] Girish Sastry [1] Pamela Mishkin [1] Brooke Chan [1]
Scott Gray [1] Nick Ryder [1] Mikhail Pavlov [1] Alethea Power [1] Lukasz Kaiser [1] Mohammad Bavarian [1]
Clemens Winter [1] Philippe Tillet [1] Felipe Petroski Such [1] Dave Cummings [1] Matthias Plappert [1]
Fotios Chantzis [1] Elizabeth Barnes [1] Ariel Herbert-Voss [1] William Hebgen Guss [1] Alex Nichol [1] Alex Paino [1]
Nikolas Tezak [1] Jie Tang [1] Igor Babuschkin [1] Suchir Balaji [1] Shantanu Jain [1] William Saunders [1]
Christopher Hesse [1] Andrew N. Carr [1] Jan Leike [1] Josh Achiam [1] Vedant Misra [1] Evan Morikawa [1]
Alec Radford [1] Matthew Knight [1] Miles Brundage [1] Mira Murati [1] Katie Mayer [1] Peter Welinder [1]
Bob McGrew [1] Dario Amodei [2] Sam McCandlish [2] Ilya Sutskever [1] Wojciech Zaremba [1]

Evalu





Evaluating Large Language Models Trained on Code, Chen et. al., 2021

Evaluation Benchmark: HumanEval

Evaluating Large Language Models Trained on Code

Datasets: openai/**openai_humaneval** · like 340 · Follow OpenAI 23.3k

task_id string · *lengths* 11↔12 6.1%	prompt string · *lengths* 240↔365 24.4%	canonical_solution string · *lengths* 186↔271 18.9%	test string · *lengths* 455↔624 18.9%	entry_point string · *lengths* 16↔19 10.4%

HumanEval/0	from typing import List
	
	def has_close_elements(numbers: List[float], threshold: float) -> bool:
	""" Check if in given list of numbers, are any two numbers closer to each other than given threshold.
	>>> has_close_elements([1.0, 2.0, 3.0], 0.5)
	False
	>>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
	True
	"""
		for idx, elem in enumerate(numbers):
		for idx2, elem2 in enumerate(numbers):
		if idx != idx2:
		distance = abs(elem - elem2)
		if distance < threshold:
		return True
		
		return False
			METADATA = {
			'author': 'jt',
			'dataset': 'test'
			}
			
			def check(candidate):
			assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
			assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
			assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
			assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
			assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True
			assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
			assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False
				has_close_elements

Evaluating Large Language Models Trained on Code, Chen et. al., 2021

Evaluation Benchmark: MBPP

Program Synthesis with Large Language Models

Jacob Austin* Augustus Odena*

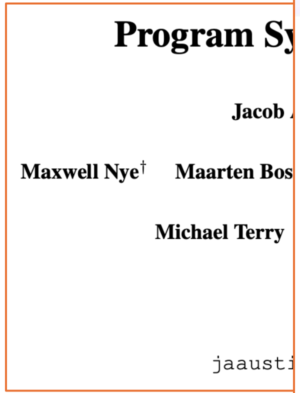
Maxwell Nye† Maarten Bosma Henryk Michalewski David Dohan Ellen Jiang Carrie Cai

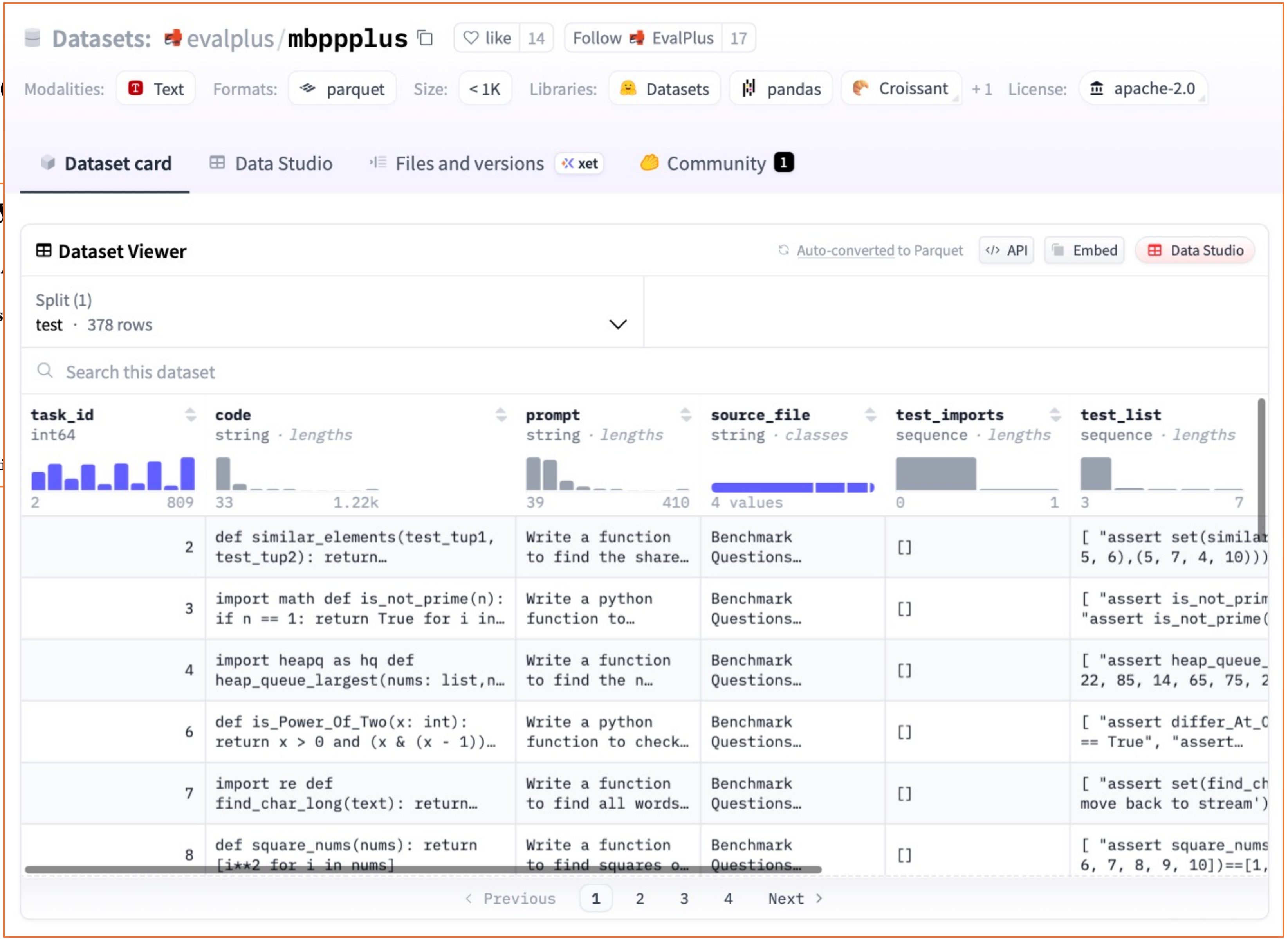
Michael Terry Quoc Le Charles Sutton

Google Research
* denotes equal contribution
jaaustin@google.com, augustusodena@google.com

Program Synthesis with Large Language Models, Austin et. al., 2021

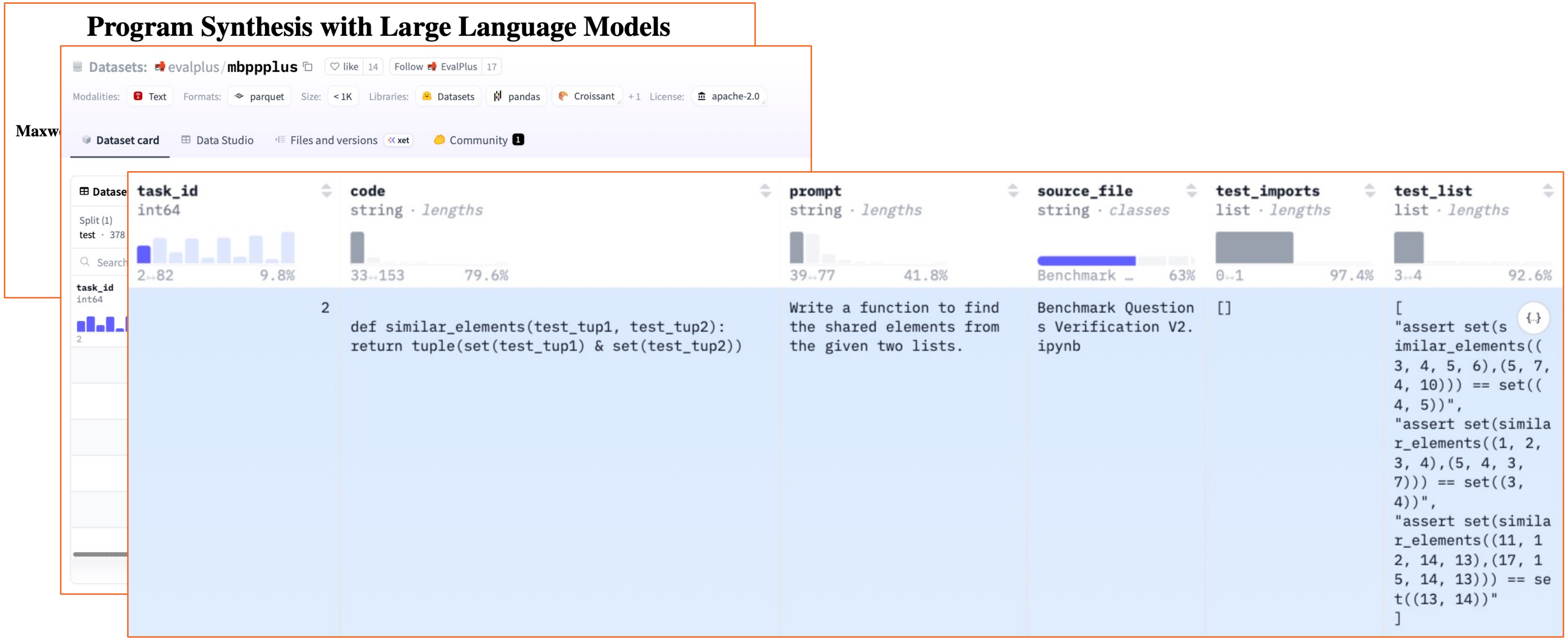
Evalu





Program Synthesis with Large Language Models, Austin et. al., 2021

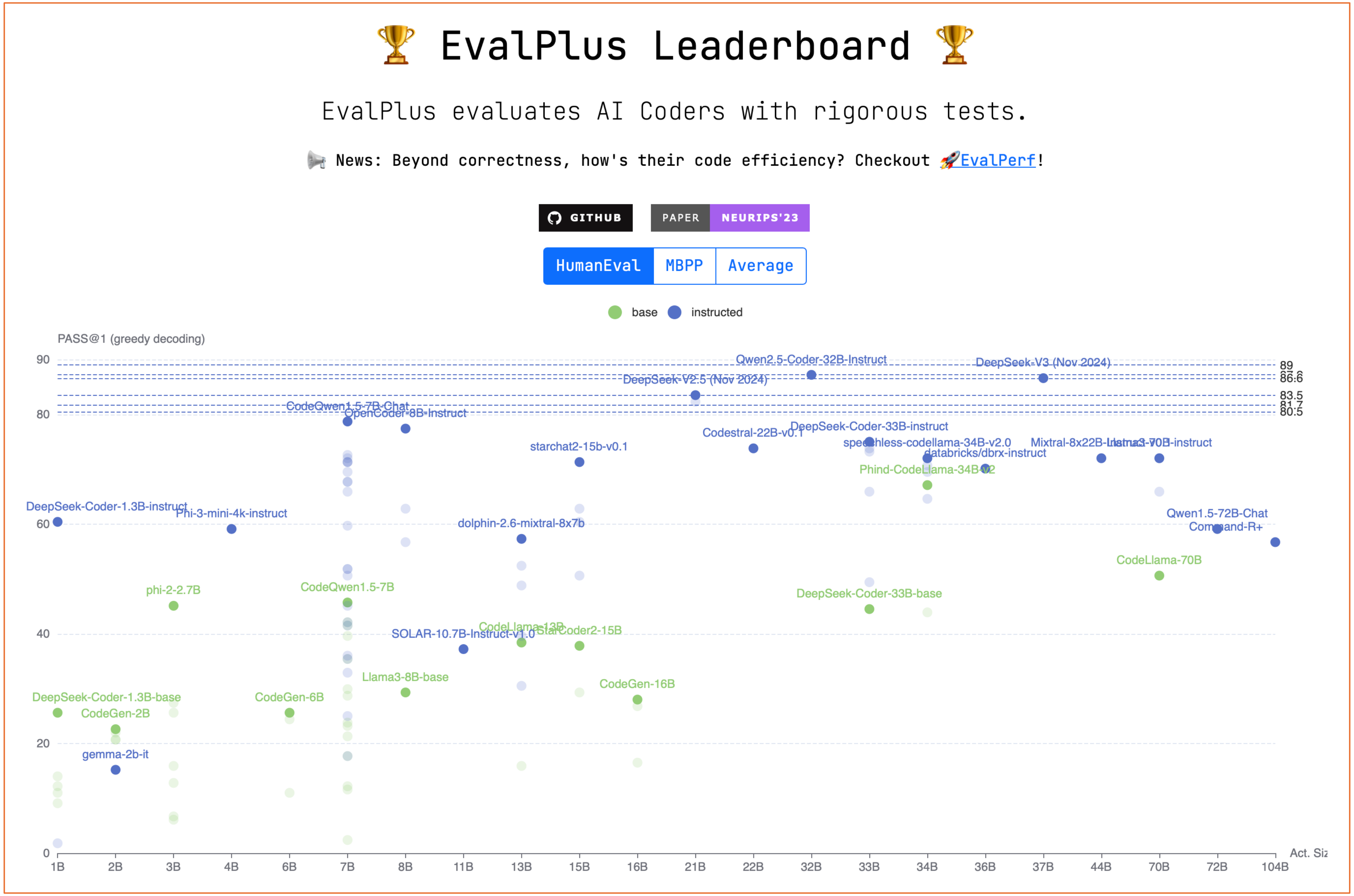
Evaluation Benchmark: MBPP



Program Synthesis with Large Language Models, Austin et. al., 2021

Evaluation Metrics

- PASS@k
  - Standard metric to evaluate code generation models
  - Intuition: practical success rate under multiple tries
  - Procedure:
    - Sample k independent code generations from the model
    - See if at least 1 code snippet satisfies success criteria
  - Success criteria:
    - Syntax, compilation check, no runtime error
    - Pass all test cases (assumes that test cases are present)

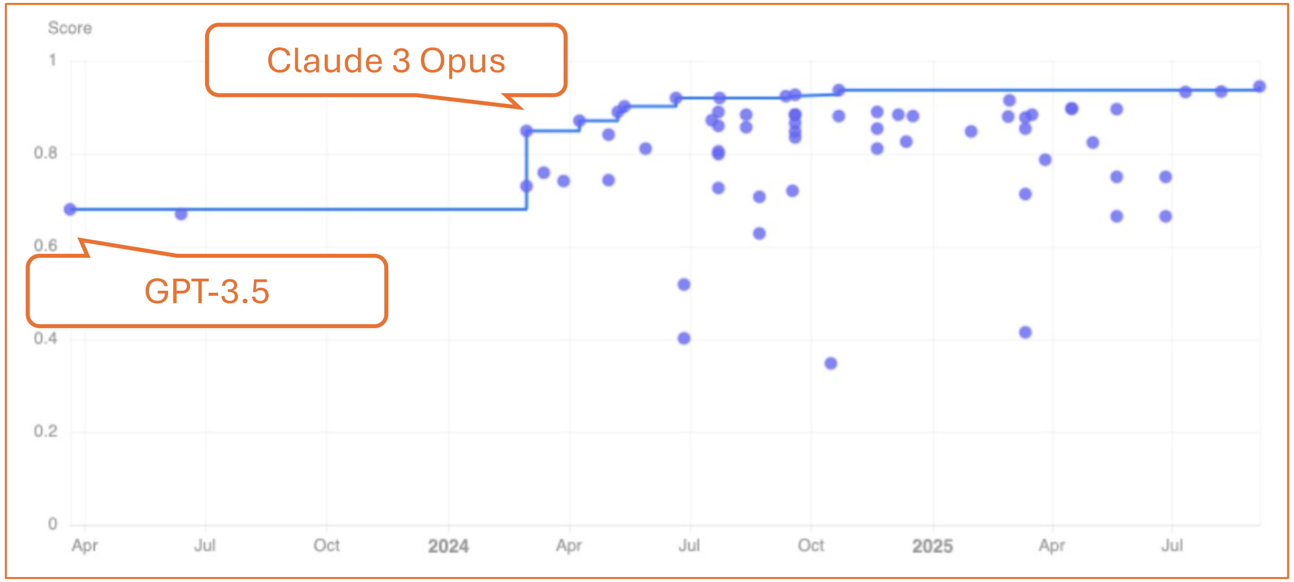


https://evalplus.github.io/leaderboard.html

Table 1. Codex, GPT-Neo, & TabNine evaluations for HumanEval. We find that GPT-J pass@1 is between Codex-85M and Codex-300M performance.

	PASS@k $k = 1$	$k = 10$	$k = 100$
GPT-NEO 125M	0.75%	1.88%	2.97%
GPT-NEO 1.3B	4.79%	7.47%	16.30%
GPT-NEO 2.7B	6.41%	11.27%	21.37%
GPT-J 6B	11.62%	15.74%	27.74%
TABNINE	2.58%	4.35%	7.59%
CODEX-12M	2.00%	3.62%	8.58%
CODEX-25M	3.21%	7.1%	12.89%
CODEX-42M	5.06%	8.8%	15.55%
CODEX-85M	8.22%	12.81%	22.4%
CODEX-300M	13.17%	20.37%	36.27%
CODEX-679M	16.22%	25.7%	40.95%
CODEX-2.5B	21.36%	35.42%	59.5%
CODEX-12B	28.81%	46.81%	72.31%

2021



2025

https://llm-stats.com/benchmarks/humaneval

CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution

Alex Gu* *gua@mit.edu*
MIT CSAIL
Baptiste Rozière *broz@meta.com*
Meta AI
Hugh Leather *hleather@meta.com*
Meta AI
Armando Solar-Lezama *asolar@csail.mit.edu*
MIT CSAIL
Gabriel Synnaeve *gab@meta.com*
Meta AI
Sida I. Wang *sida@meta.com*
Meta AI

CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution

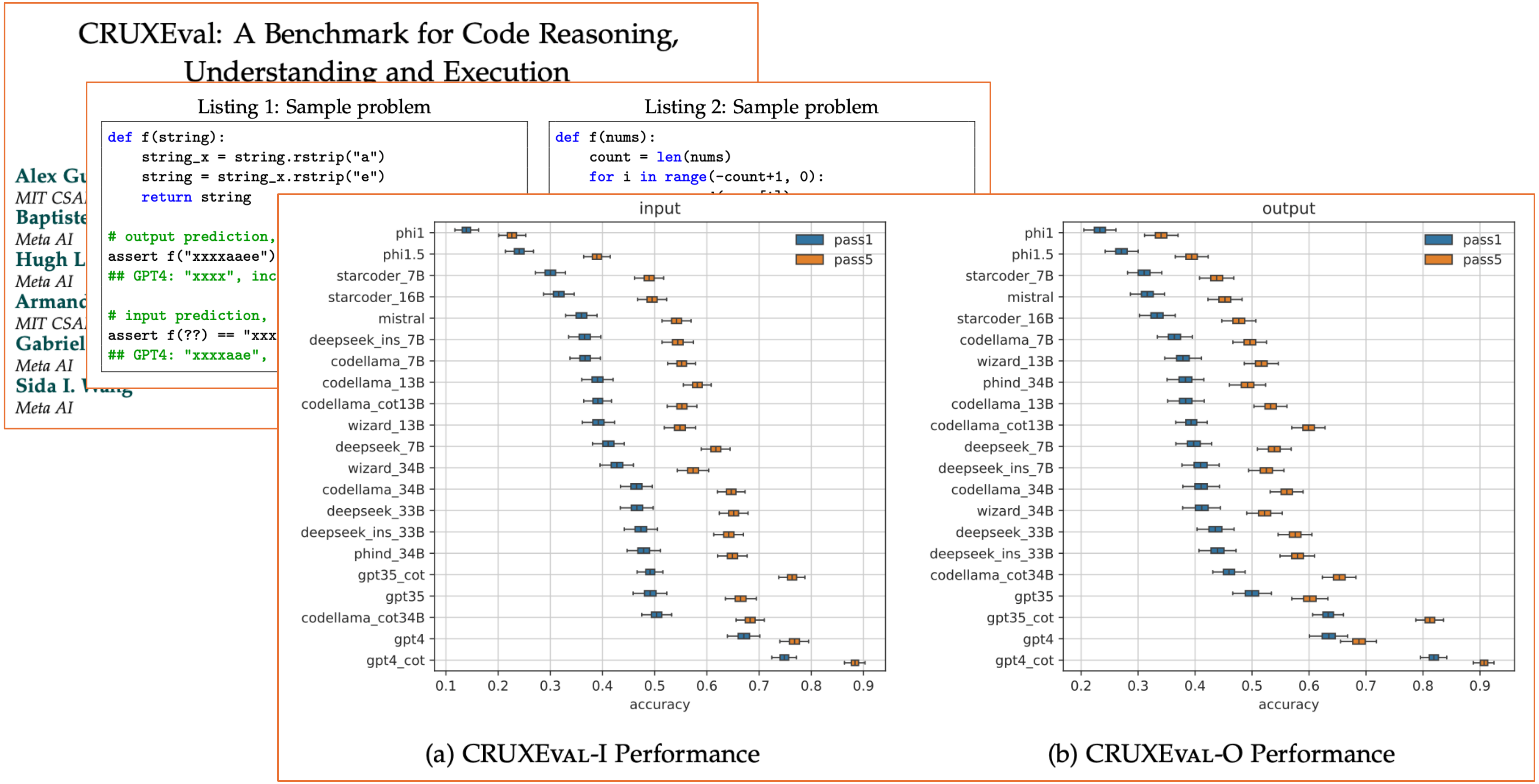
Alex Gu* *gua@mit.edu*
MIT CSAIL
Baptiste Rozière *broz@meta.com*
Meta AI
Hugh Leather *hleather@meta.com*
Meta AI
Armando Solar-Lezama
MIT CSAIL
Gabriel Synnaeve
Meta AI
Sida I. Wang
Meta AI

Listing 1: Sample problem

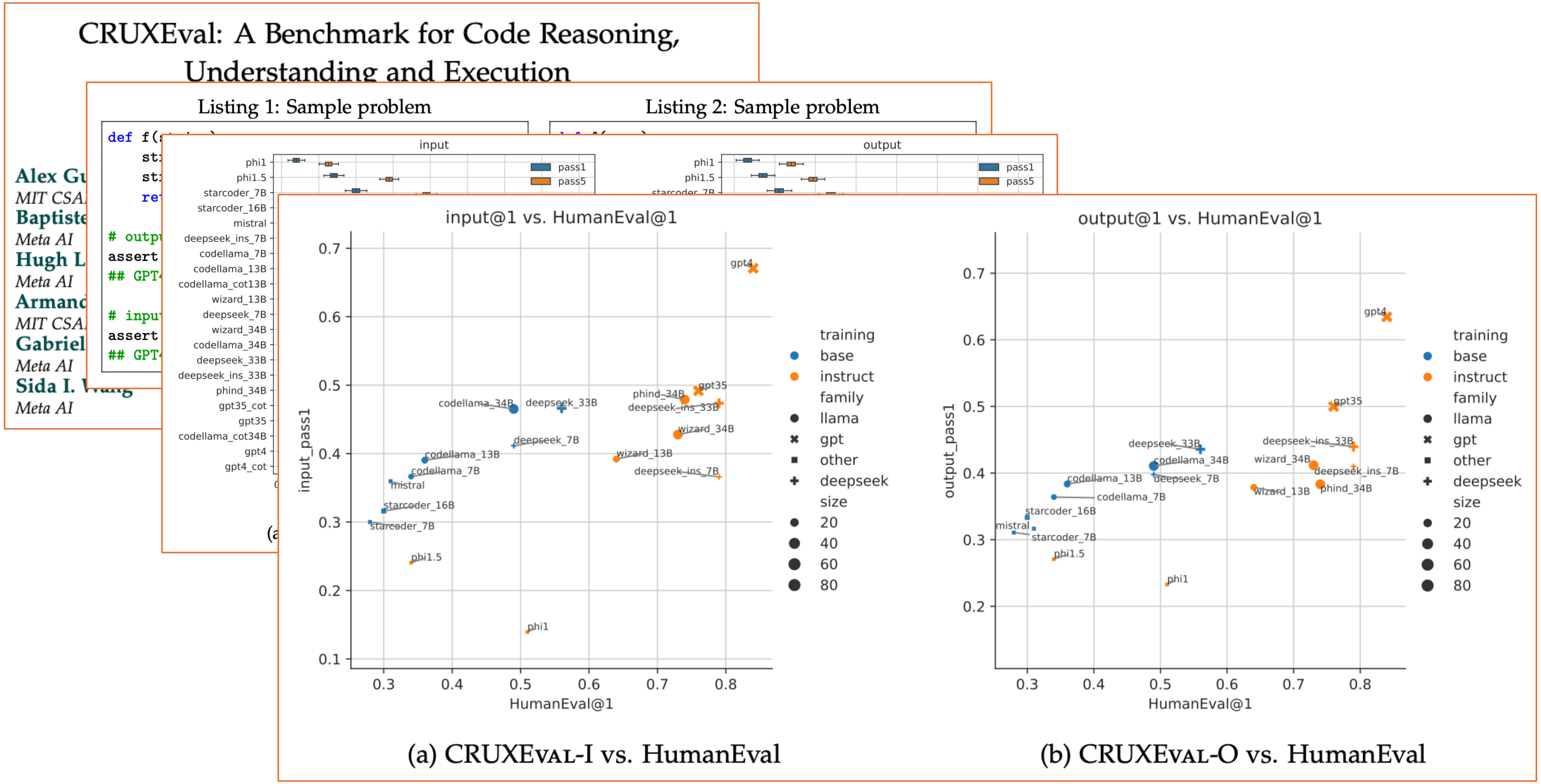
```
def f(string):
    string_x = string.rstrip("a")
    string = string_x.rstrip("e")
    return string

# output prediction, CRUXEval-O
assert f("xxxxaaee") == ??
## GPT4: "xxxx", incorrect

# input prediction, CRUXEval-I
assert f(??) == "xxxxaa"
## GPT4: "xxxxaae", correct
```

Listing 2: Sample problem

```
def f(nums):
    count = len(nums)
    for i in range(-count+1, 0):
        nums.append(nums[i])
    return nums
# output prediction, CRUXEval-O
assert f([2, 6, 1, 3, 1]) == ??
# GPT4: [2, 6, 1, 3, 1, 6, 1, 3, 1], incorrect

# input prediction, CRUXEval-I
assert f(??) == [2, 6, 1, 3, 1, 6, 3, 6, 6]
# GPT4: [2, 6, 1], incorrect
```



CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution
Listing 1: Sample problem
def f(string):
string_x = string.rstrip("a")
string = string_x.rstrip("e")
return string
output prediction,
assert f("xxxxaaee")
GPT4: "xxxx", inc
input prediction,
assert f(??) == "xxx
GPT4: "xxxxaae",
Listing 2: Sample problem
def f(nums):
count = len(nums)
for i in range(-count+1, 0):
MIT CSAI
Meta AI
Meta AI
MIT CSAI
Meta AI
Sida I.
Meta AI
input
pass1
pass5
phi1
phi1.5
starcoder_7B
starcoder_16B
mistral
deepseek_ins_7B
codellama_7B
codellama_13B
codellama_cot13B
wizard_13B
deepseek_7B
wizard_34B
codellama_34B
deepseek_33B
deepseek_ins_33B
phind_34B
gpt35_cot
gpt35
codellama_cot34B
gpt4
gpt4_cot
0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
accuracy
output
pass1
pass5
phi1
phi1.5
starcoder_7B
mistral
starcoder_16B
codellama_7B
wizard_13B
phind_34B
codellama_13B
codellama_cot13B
deepseek_7B
deepseek_ins_7B
codellama_34B
wizard_34B
deepseek_33B
deepseek_ins_33B
codellama_cot34B
gpt35
gpt35_cot
gpt4
gpt4_cot
0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
accuracy
(a) CRUXEval-I Performance
(b) CRUXEval-O Performance



(a) CRUXEval-I vs. HumanEval

(b) CRUXEval-O vs. HumanEval

swebench.com

SWE-bench

Leaderboards

BENCHMARKS

- SWE-bench
- SWE-bench Lite
- SWE-bench Multilingual
- SWE-bench Multimodal
- SWE-bench Bash Only
- SWE-bench Verified

ABOUT

- Paper
- Blog
- Docs
- Contact
- Citations
- Press
- Submit

SWE-BENCH FAMILY

- SWE-agent

Leaderboards

There's an all-new, challenging SWE-bench **Multimodal**, containing software issues described with images. Learn more here.

Bash Only | Verified | Lite | Full | Multimodal

Bash Only evaluates all LMs with a minimal agent on SWE-bench Verified (details)

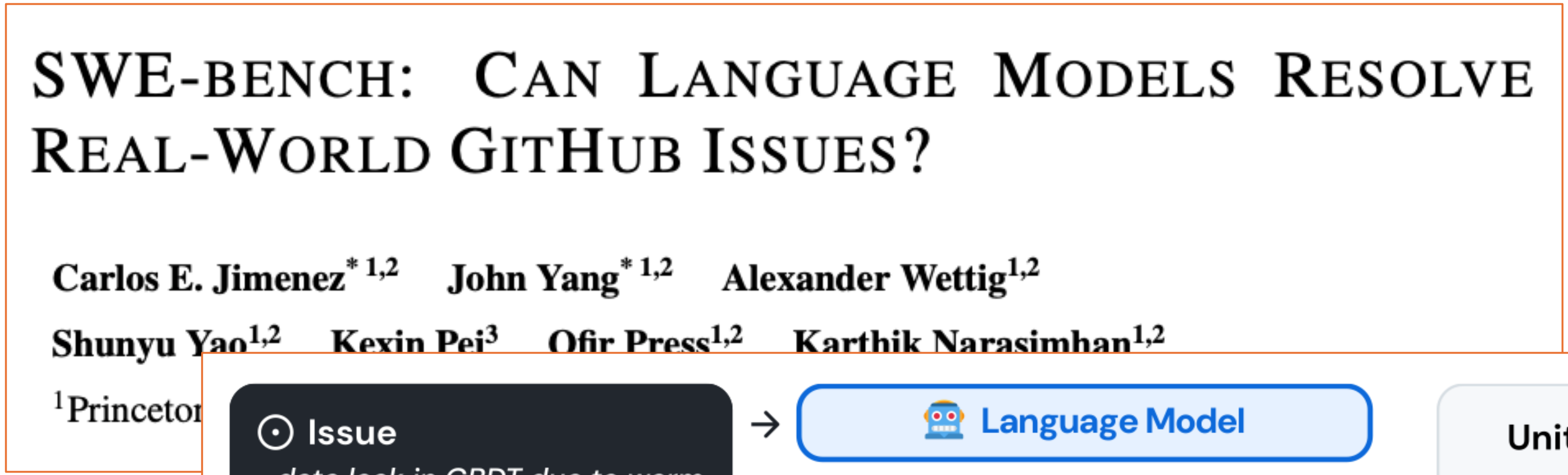
Filters: Open Scaffold ▾ All Tags ▾

Model	% Resolved	Avg. $	Org	Date	Release
🆕 ✅ Claude 4.5 Sonnet (20250929)	70.60	$0.56		2025-09-29	1.13.3
🆕 ✅ Claude 4 Opus (20250514)	67.60	$1.13		2025-08-02	1.0.0
🆕 ✅ GPT-5 (2025-08-07) (medium reasoning)	65.00	$0.28		2025-08-07	1.7.0
🆕 ✅ Claude 4 Sonnet (20250514)	64.93	$0.37		2025-07-26	1.0.0
🆕 ✅ GPT-5 mini (2025-08-07) (medium reasoning)	59.80	$0.04		2025-08-07	1.7.0
🆕 ✅ o3 (2025-04-16)	58.40	$0.33		2025-07-26	1.0.0
🆕 ✅ Qwen3-Coder 480B/A35B Instruct	55.40	$		2025-08-02	1.0.0
🆕 ✅ GLM-4.5 (2025-08-22)	54.20	$0.30		2025-08-22	1.9.1
🆕 ✅ Gemini 2.5 Pro (2025-05-06)	53.60	$0.29		2025-07-26	1.0.0
🆕 ✅ Claude 3.7 Sonnet (20250219)	52.80	$0.35		2025-07-20	0.0.0
🆕 ✅ o4-mini (2025-04-16)	45.00	$0.21		2025-07-26	1.0.0

SWE-bench **Bash Only** uses the SWE-bench Verified dataset with the mini-SWE-agent environment for all models [Post].

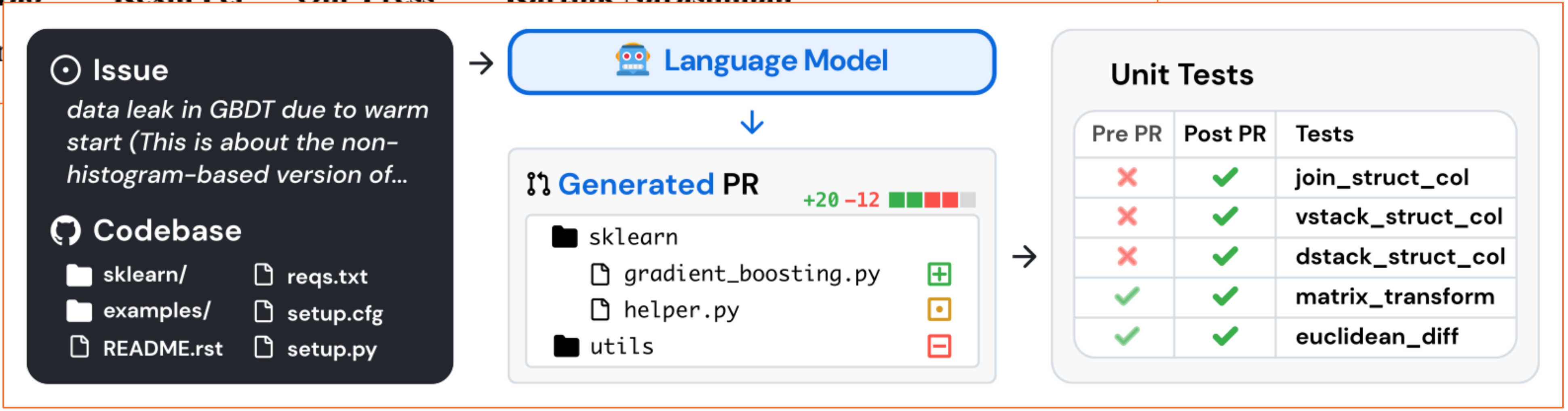
SWE-bench: Can Language Models Resolve Real-World GitHub Issues?

Carlos E. Jimenez[*1,2] **John Yang**[*1,2] **Alexander Wettig**[1,2]
Shunyu Yao[1,2] **Kexin Pei**[3] **Ofir Press**[1,2] **Karthik Narasimhan**[1,2]
[1]Princeton University [2]Princeton Language and Intelligence [3]University of Chicago

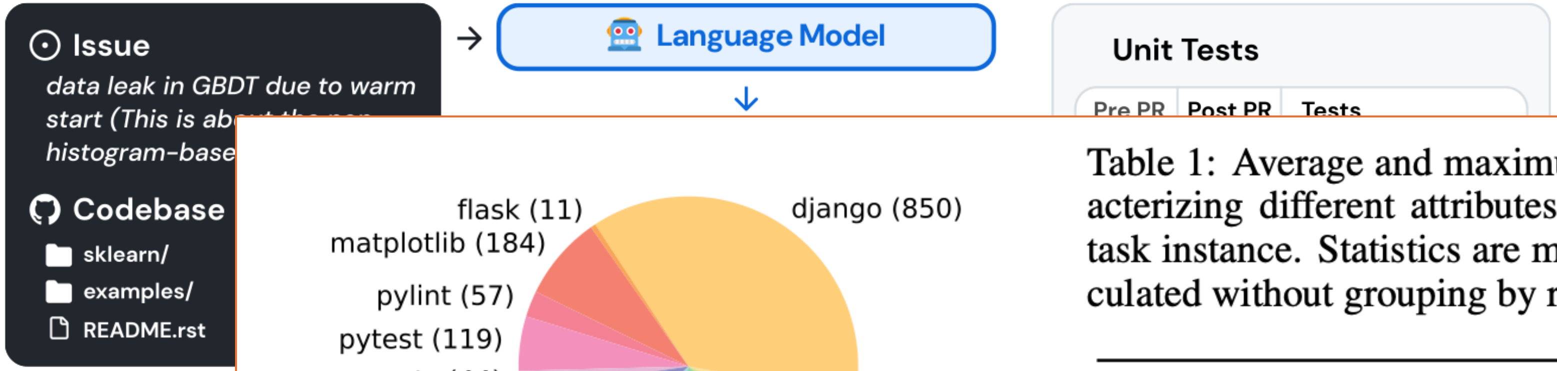


SWE-bench: Can Language Models Resolve Real-World GitHub Issues?

Carlos E. Jimenez[*1,2] John Yang[*1,2] Alexander Wettig[1,2]
Shunyu Yao[1,2] Kexin Pei[3] Ofir Press[1,2] Karthik Narasimhan[1,2]
[1]Princeton



SWE-bench: Can Language Models Resolve



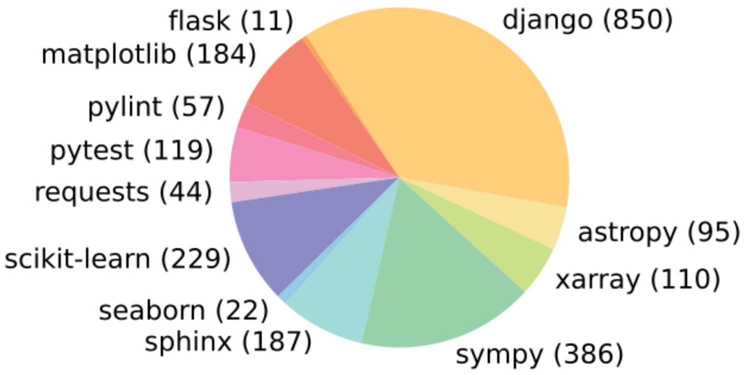
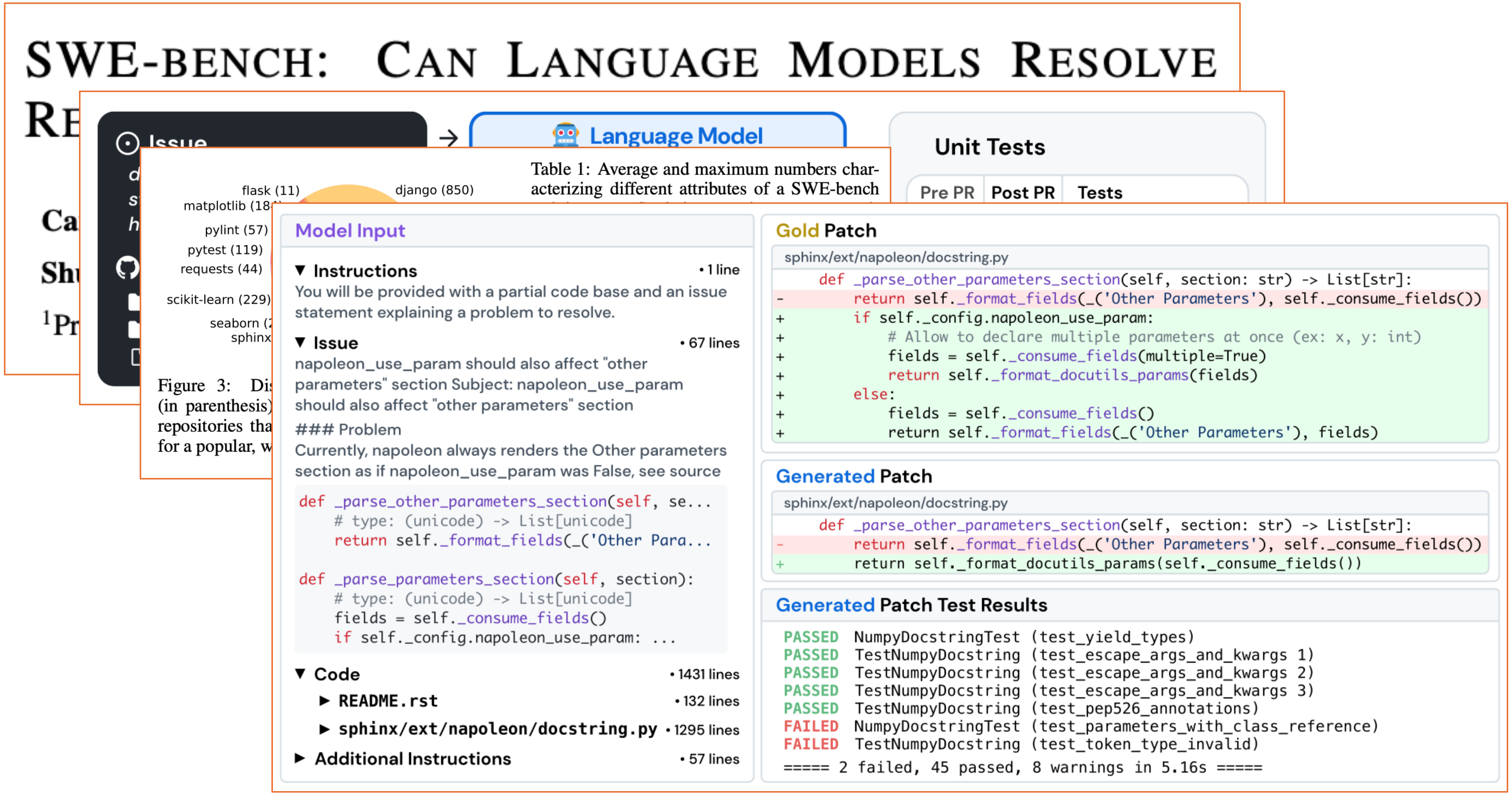


Figure 3: Distribution of SWE-bench tasks (in parenthesis) across 12 open source GitHub repositories that each contains the source code for a popular, widely downloaded PyPI package.

Table 1: Average and maximum numbers characterizing different attributes of a SWE-bench task instance. Statistics are micro-averages calculated without grouping by repository.

		Mean	Max
Issue Text	Length (Words)	195.1	4477
Codebase	# Files (non-test)	3,010	5,890
	# Lines (non-test)	438K	886K
Gold Patch	# Lines edited	32.8	5888
	# Files edited	1.7	31
	# Func. edited	3	36
Tests	# Fail to Pass	9.1	1633
	# Total	120.8	9459


SWE-BENCH: CAN LANGUAGE MODELS RESOLVE
Issue
Language Model
Unit Tests
Pre PR
Post PR
Tests
Table 1: Average and maximum numbers characterizing different attributes of a SWE-bench
flask (11)
django (850)
matplotlib (18
pylint (57)
pytest (119)
requests (44)
scikit-learn (229)
seaborn (
sphinx
Figure 3: Di
(in parenthesis)
repositories tha
for a popular, w
Model Input
Instructions
• 1 line
You will be provided with a partial code base and an issue statement explaining a problem to resolve.
Issue
• 67 lines
napoleon_use_param should also affect "other parameters" section Subject: napoleon_use_param should also affect "other parameters" section
Problem
Currently, napoleon always renders the Other parameters section as if napoleon_use_param was False, see source
def _parse_other_parameters_section(self, se...
type: (unicode) -> List[unicode]
return self._format_fields(_('Other Para...
def _parse_parameters_section(self, section):
# type: (unicode) -> List[unicode]
fields = self._consume_fields()
if self._config.napoleon_use_param: ...
Code
• 1431 lines
README.rst
• 132 lines
sphinx/ext/napoleon/docstring.py
• 1295 lines
Additional Instructions
• 57 lines
Gold Patch
sphinx/ext/napoleon/docstring.py
def _parse_other_parameters_section(self, section: str) -> List[str]:
- return self._format_fields(_('Other Parameters'), self._consume_fields())
+ if self._config.napoleon_use_param:
+ # Allow to declare multiple parameters at once (ex: x, y: int)
+ fields = self._consume_fields(multiple=True)
+ return self._format_docutils_params(fields)
+ else:
+ fields = self._consume_fields()
+ return self._format_fields(_('Other Parameters'), fields)
Generated Patch
sphinx/ext/napoleon/docstring.py
def _parse_other_parameters_section(self, section: str) -> List[str]:
- return self._format_fields(_('Other Parameters'), self._consume_fields())
+ return self._format_docutils_params(self._consume_fields())
Generated Patch Test Results
PASSED NumpyDocstringTest (test_yield_types)
PASSED TestNumpyDocstring (test_escape_args_and_kwargs 1)
PASSED TestNumpyDocstring (test_escape_args_and_kwargs 2)
PASSED TestNumpyDocstring (test_escape_args_and_kwargs 3)
PASSED TestNumpyDocstring (test_pep526_annotations)
FAILED NumpyDocstringTest (test_parameters_with_class_reference)
FAILED TestNumpyDocstring (test_token_type_invalid)
===== 2 failed, 45 passed, 8 warnings in 5.16s =====

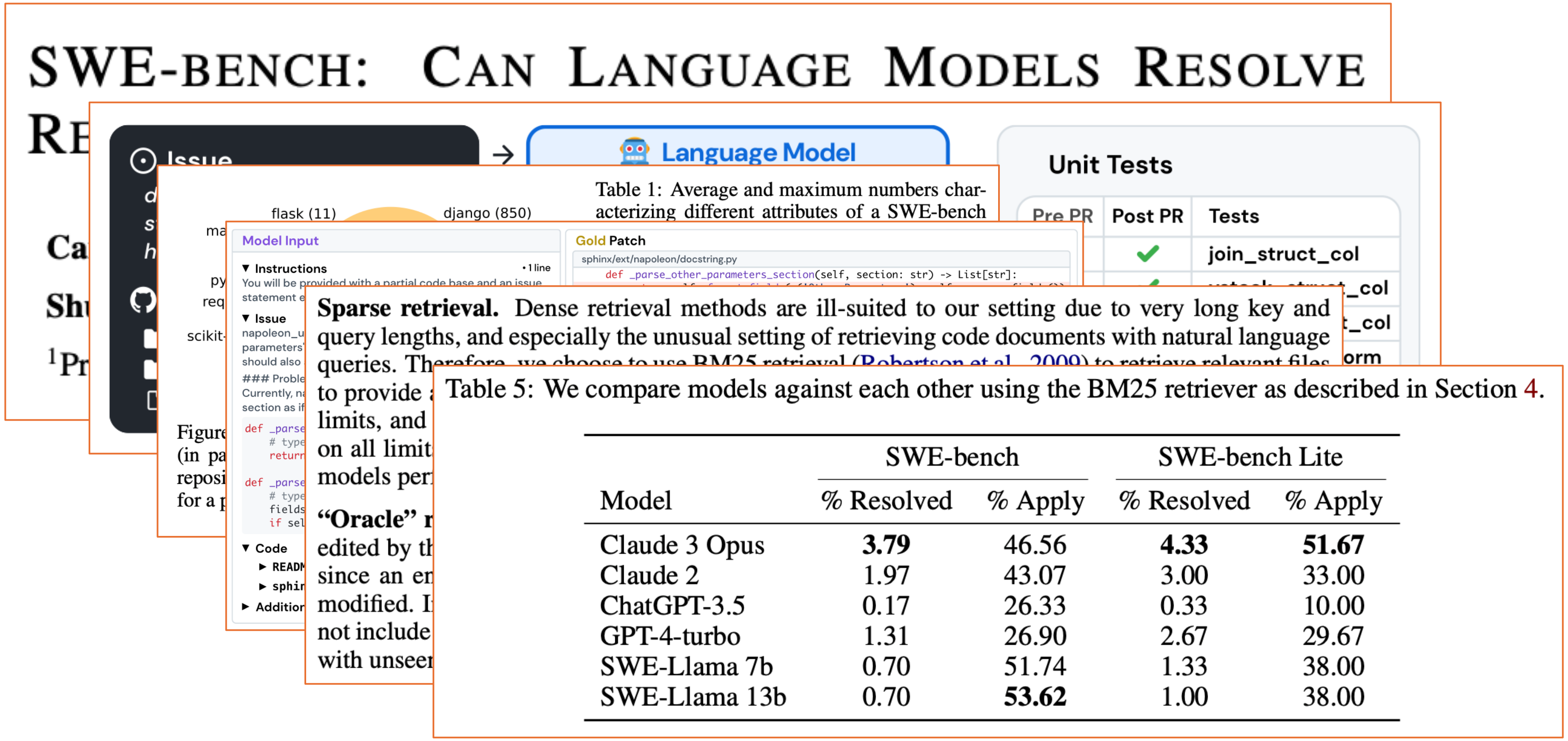


Table 5: We compare models against each other using the BM25 retriever as described in Section 4.

	SWE-bench		SWE-bench Lite	
Model	% Resolved	% Apply	% Resolved	% Apply
Claude 3 Opus	**3.79**	46.56	**4.33**	**51.67**
Claude 2	1.97	43.07	3.00	33.00
ChatGPT-3.5	0.17	26.33	0.33	10.00
GPT-4-turbo	1.31	26.90	2.67	29.67
SWE-Llama 7b	0.70	51.74	1.33	38.00
SWE-Llama 13b	0.70	**53.62**	1.00	38.00

gpt-oss-120b & gpt-oss-20b Model Card

OpenAI

August 5, 2025

Table 3: Evaluations across multiple benchmarks and reasoning levels.

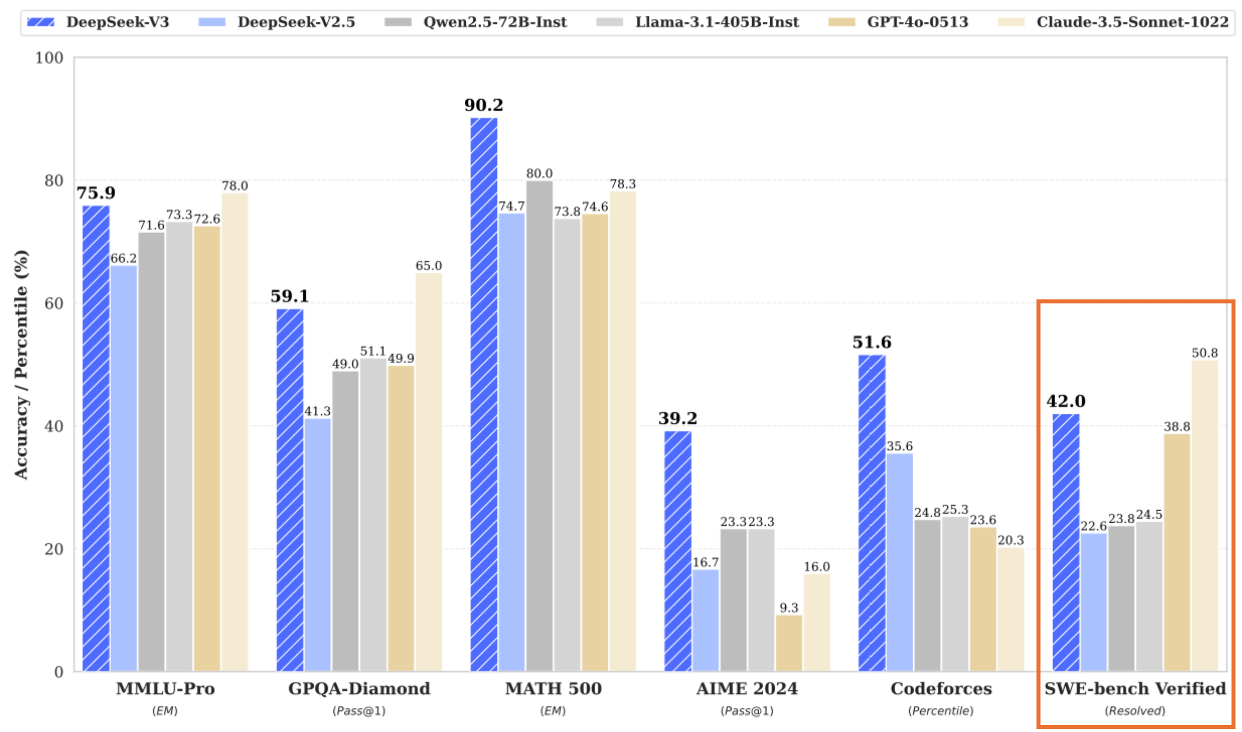
	gpt-oss-120b			gpt-oss-20b		
Benchmark (Accuracy (%))	**low**	**medium**	**high**	**low**	**medium**	**high**
AIME 2024 (no tools)	56.3	80.4	95.8	42.1	80.0	92.1
AIME 2024 (with tools)	75.4	87.9	96.6	61.2	86.0	96.0
AIME 2025 (no tools)	50.4	80.0	92.5	37.1	72.1	91.7
AIME 2025 (with tools)	72.9	91.6	97.9	57.5	90.4	98.7
GPQA Diamond (no tools)	67.1	73.1	80.1	56.8	66.0	71.5
GPQA Diamond (with tools)	68.1	73.5	80.9	58.0	67.1	74.2
HLE (no tools)	5.2	8.6	14.9	4.2	7.0	10.9
HLE (with tools)	9.1	11.3	19.0	6.3	8.8	17.3
MMLU	85.9	88.0	90.0	80.4	84.0	85.3
SWE-Bench Verified	47.9	52.6	62.4	37.4	53.2	60.7
Tau-Bench Retail	49.4	62.0	67.8	35.0	47.3	54.8
Tau-Bench Airline	42.6	48.6	49.2	32.0	42.6	38.0
Aider Polyglot	24.0	34.2	44.4	16.6	26.6	34.2
MMMLU (Average)	74.1	79.3	81.3	67.0	73.5	75.7
Benchmark (Score (%))	**low**	**medium**	**high**	**low**	**medium**	**high**
HealthBench	53.0	55.9	57.6	40.4	41.8	42.5
HealthBench Hard	22.8	26.9	30.0	9.0	12.9	10.8
HealthBench Consensus	90.6	90.8	89.9	84.9	83.0	82.6
Benchmark (Elo)	**low**	**medium**	**high**	**low**	**medium**	**high**
Codeforces (no tools)	1595	2205	2463	1366	1998	2230
Codeforces (with tools)	1653	2365	2622	1251	2064	2516

deepseek

DeepSeek-V3 Technical Report

DeepSeek-AI

research@deepseek.com



Bash Only | Verified | Lite | Full | Multimodal

Bash Only evaluates all LMs with a minimal agent on SWE-bench Verified (details)

Filters: Open Scaffold ▾ All Tags ▾

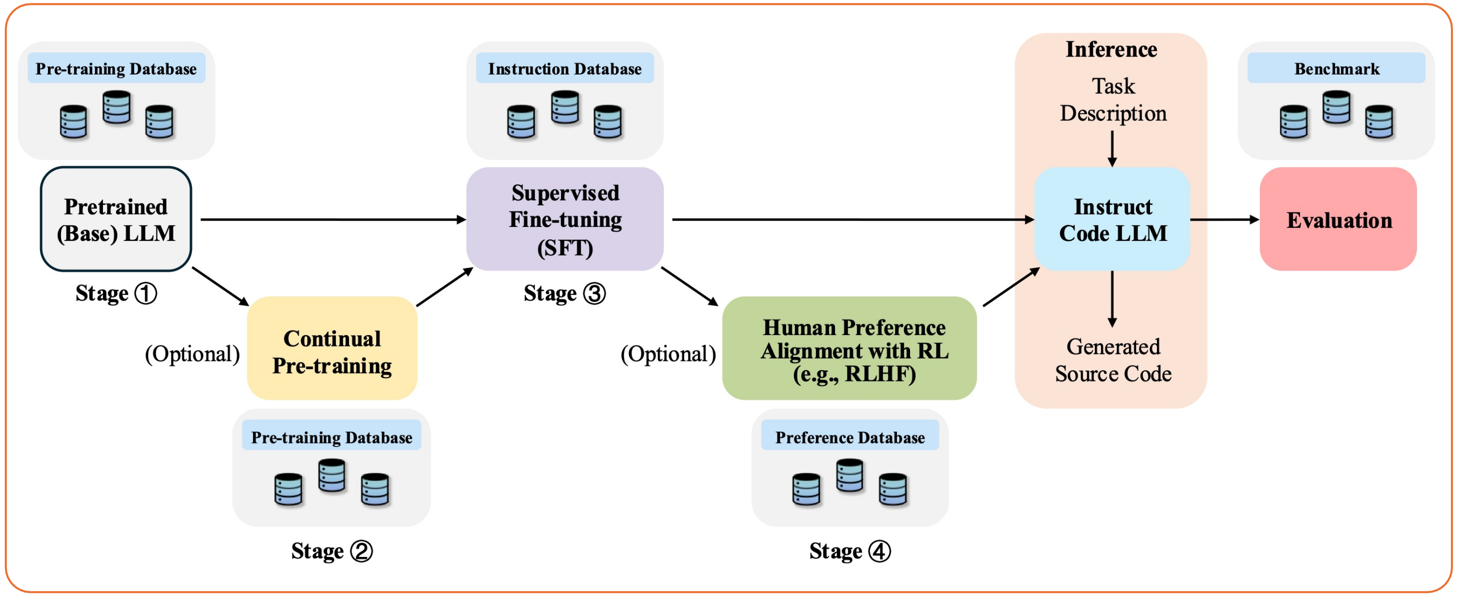
Model	% Resolved	Avg. $	Org	Date	Release
🆕 ✅ Claude 4.5 Sonnet (20250929)	70.60	$0.56	A\	2025-09-29	1.13.3
🆕 ✅ Claude 4 Opus (20250514)	67.60	$1.13	A\	2025-08-02	1.0.0
🆕 ✅ GPT-5 (2025-08-07) (medium reasoning)	65.00	$0.28		2025-08-07	1.7.0
🆕 ✅ Claude 4 Sonnet (20250514)	64.93	$0.37	A\	2025-07-26	1.0.0
🆕 ✅ GPT-5 mini (2025-08-07) (medium reasoning)	59.80	$0.04		2025-08-07	1.7.0
🆕 ✅ o3 (2025-04-16)	58.40	$0.33		2025-07-26	1.0.0
🆕 ✅ Qwen3-Coder 480B/A35B Instruct	55.40	$		2025-08-02	1.0.0
🆕 ✅ GLM-4.5 (2025-08-22)	54.20	$0.30		2025-08-22	1.9.1
🆕 ✅ Gemini 2.5 Pro (2025-05-06)	53.60	$0.29		2025-07-26	1.0.0
🆕 ✅ Claude 3.7 Sonnet (20250219)	52.80	$0.35	A\	2025-07-20	0.0.0
🆕 ✅ o4-mini (2025-04-16)	45.00	$0.21		2025-07-26	1.0.0

Evaluating Coding Language Models

- High-level Task
  - Function implementation, Resolve GitHub issues, Fixing vulnerability, etc.
- Mid-level Task
  - In-place code completion, Patch generation, Bash operation, etc.
- Low-level Task
  - Next-token prediction
- Evaluation Dataset
  - MBPP, HumanEval, BigCodeBench, ClassEval, NaturalCodeBench, etc.
  - CRUXEval, SWE-Bench (different variants), etc.
- Evaluation Metric
  - Pass@k, %Resolved, $Cost, Token Cost, Time, BLEU score, etc.

Today’s Agenda

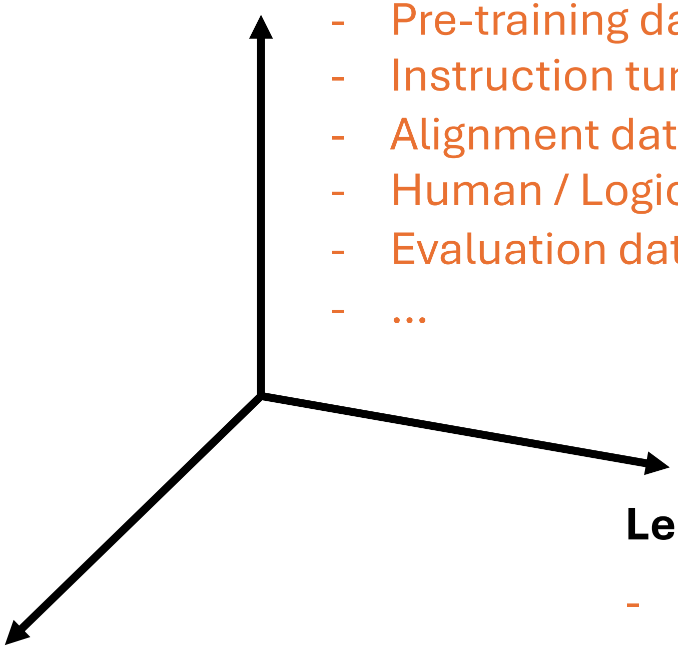
- Pre-training stage
  - ~~Model architecture~~
  - ~~Pre-training dataset~~
  - ~~Learning objectives~~
  - ~~Optimization~~
  - ~~Evaluation dataset~~
- Post-training stage
  - Supervised fine-tuning
  - Reinforcement learning



How to obtain a “good enough” LLM

Dataset

- Pre-training dataset / Fine-tuning dataset
- Instruction tuning dataset
- Alignment dataset
- Human / Logical feedback dataset
- Evaluation dataset
- …



Learning Procedure

- Optimization objectives
- Learning algorithm (SFT, RL, etc.)
- Continual learning, Curriculum learning
- Staged learning
- …

Model Architecture

- Encoder-decoder models
- Decoder-only models
- Hyper-parameter tuning
- …

Logistics – Week 7

- Assignment 3: Coding Agents
  - Due: Oct 23
- Oral presentation sign up sheet
  - Sent out during the weekend
  - Oral presentation starting on Week 9
- Forming groups for your final projects!
  - Sign up form will be sent out on Thursday
  - Form a group of 2-3 before Next Thursday (Oct 16)