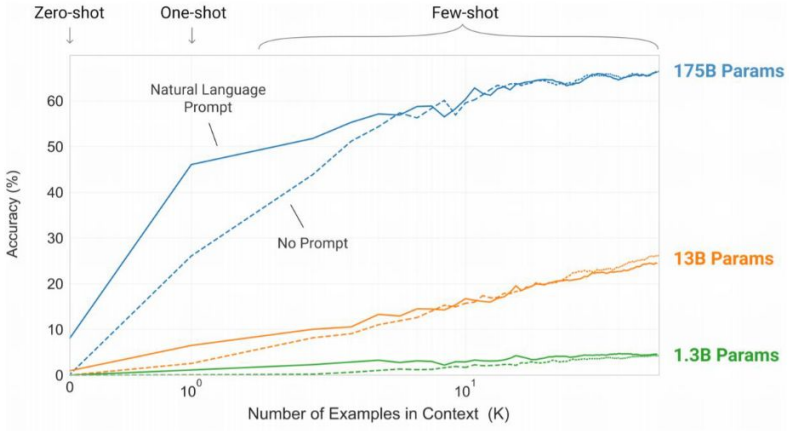
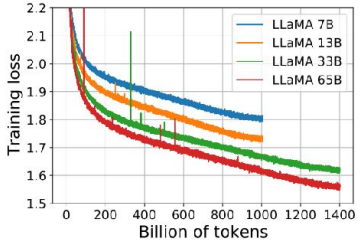
Recall: Large Language Models (LLMs)

Large-Scale Pre-Training: GPT-3



Large Language Models: Where Are We Now?

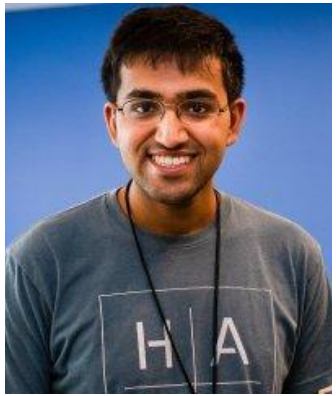
- Since GPT-3:
  - Many open-source LLMs, most recently LLaMa
  - Realization that these models converge better with *far more* tokens than parameters
    - Chinchilla, from DeepMind, empirically proved a series of scaling laws that minimize test loss for a given compute budget (but still tend to produce overly large models)



Should the focus only be on building bigger models?

CAT-LM Training Language Models on Aligned Code And Tests


Nikitha Rao*


Kush Jain*


Uri Alon


Claire Le Goues


Vincent Hellendoorn

*equal contribution

S3D Software and Societal Systems Department

Carnegie Mellon University

nikitharao@cmu.edu

Testing is important but time consuming

Automate test generation with LLMs!

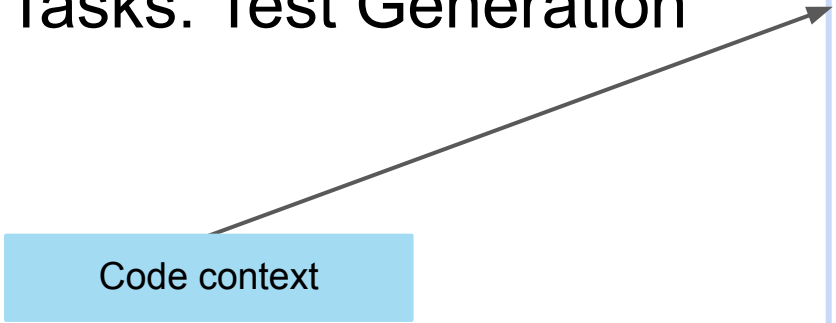
Code example

```python
def squared_norm(x):
    """Squared Euclidean or Frobenius norm of x.
    Returns the Euclidean norm when x is a vector, the Frobenius norm when x
    is a matrix (2-d array). Faster than norm(x) ** 2.
    """
    x = _ravel(x)
    if np.issubdtype(x.dtype, np.integer):
        warnings.warn('Array type is integer, np.dot may overflow. '
                      'Data should be float type to avoid this issue',
                      UserWarning)
    return np.dot(x, x)
```

Test example

```python
def test_norm_squared_norm():
    X = np.random.RandomState(42).randn(50, 63)
    X *= 100        # check stability
    X += 200
    assert_almost_equal(np.linalg.norm(X.ravel()), norm(X))
    assert_almost_equal(norm(X) ** 2, squared_norm(X), decimal=6)
    assert_almost_equal(np.linalg.norm(X), np.sqrt(squared_norm(X)), decimal=6)
    # Check the warning with an int array and np.dot potential overflow
    assert_warns_message(UserWarning, 'Array type is integer, np.dot may '
                         'overflow. Data should be float type to avoid this issue',
                         squared_norm, X.astype(int))
```

Tasks: Test Generation




```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
```

Tasks: Test Generation

First test generation

```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
public class BankTest {
    @Test
    public void FirstTest() {...}



}
```

Tasks: Test Generation

Last test generation

```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
public class BankTest {
    @Test
    public void FirstTest() {...}
    ...
    @Test
    public void Test_k() {...}
    ...


    @Test
    public void LastTest() {...}


}
```

Tasks: Test Generation

```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
public class BankTest {
    @Test
    public void FirstTest() {...}
    ...
    @Test
    public void Test_k() {...}
    ...


    @Test
    public void LastTest() {...}
    @Test
    public void ExtraTest() {...}
}
```

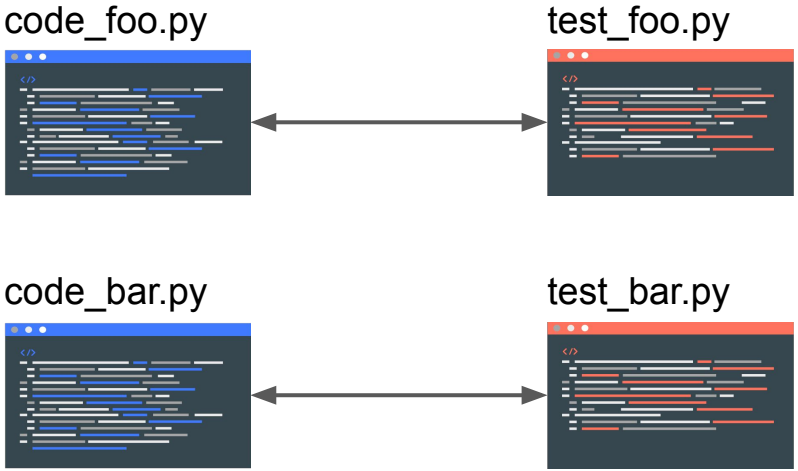
Extra test generation

Tasks: Test Completion

Test completion

```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
public class BankTest {
    @Test
    public void FirstTest() {...}
    ...
    @Test
    public void Test_k() {
        assertNotNull(Bank());
    }
    ...
    @Test
    public void LastTest() {...}
    @Test
    public void ExtraTest() {...}
}
```

Current models **cannot** consider the code under test



Key Insight:

Existing models are trained on separate code and test files, and can **never** learn the dependency between them.

Goal:

Demonstrate the importance of incorporating domain knowledge when building models.

Solution: Use Aligned Code-Test File Pairs

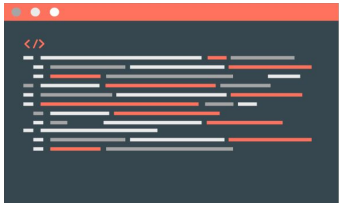
Code File

+

<|codetestpair|>

+

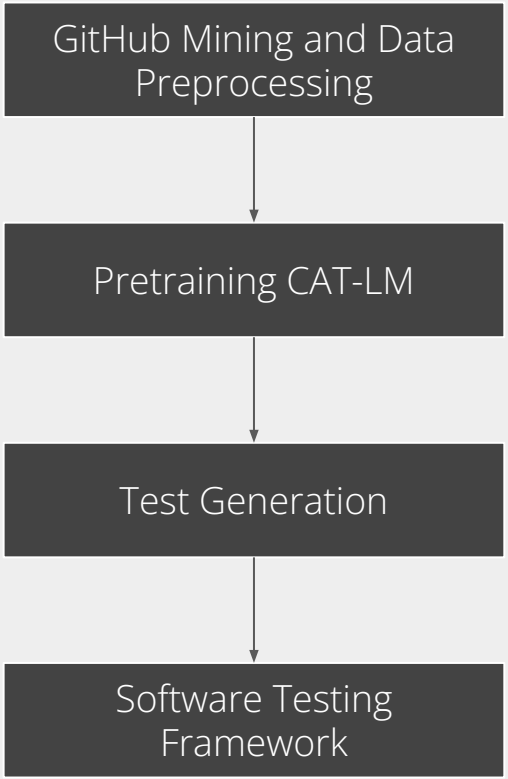
Test File



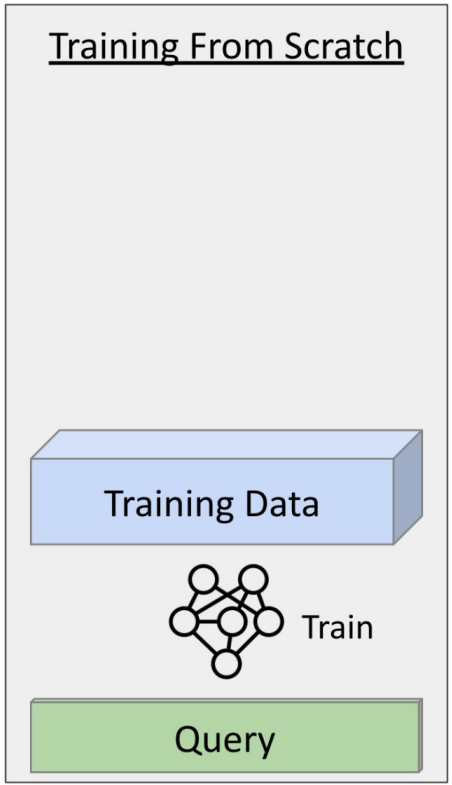
Code-Test File Pair

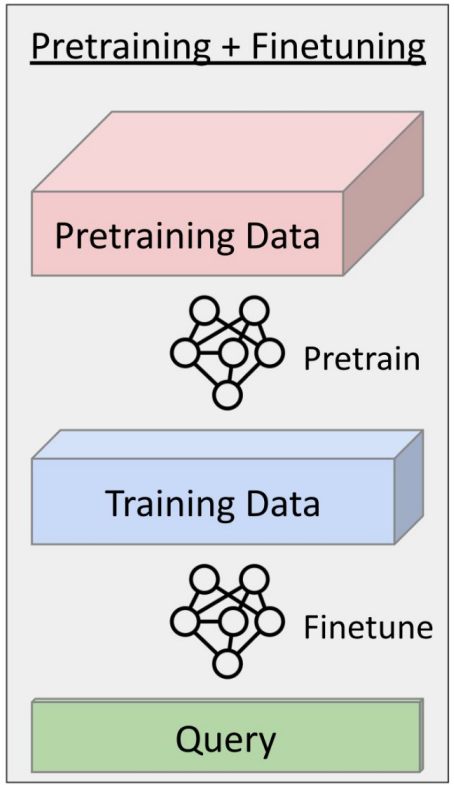
Overview

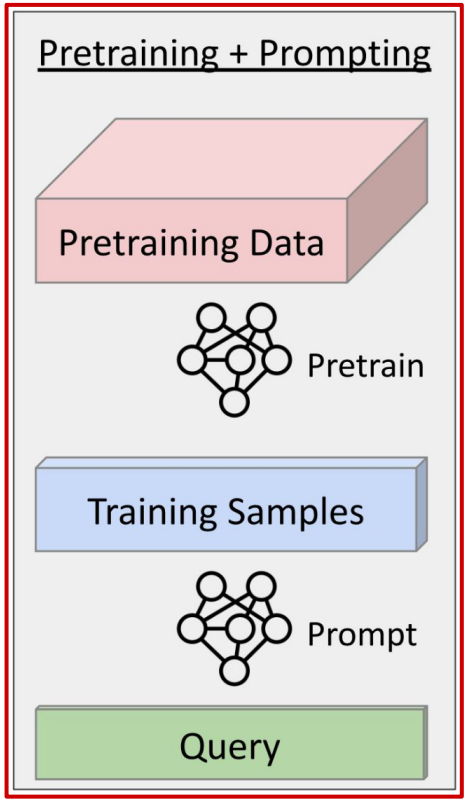

LM


GitHub Mining and Data Preprocessing
Pretraining CAT-LM
Test Generation
Software Testing Framework

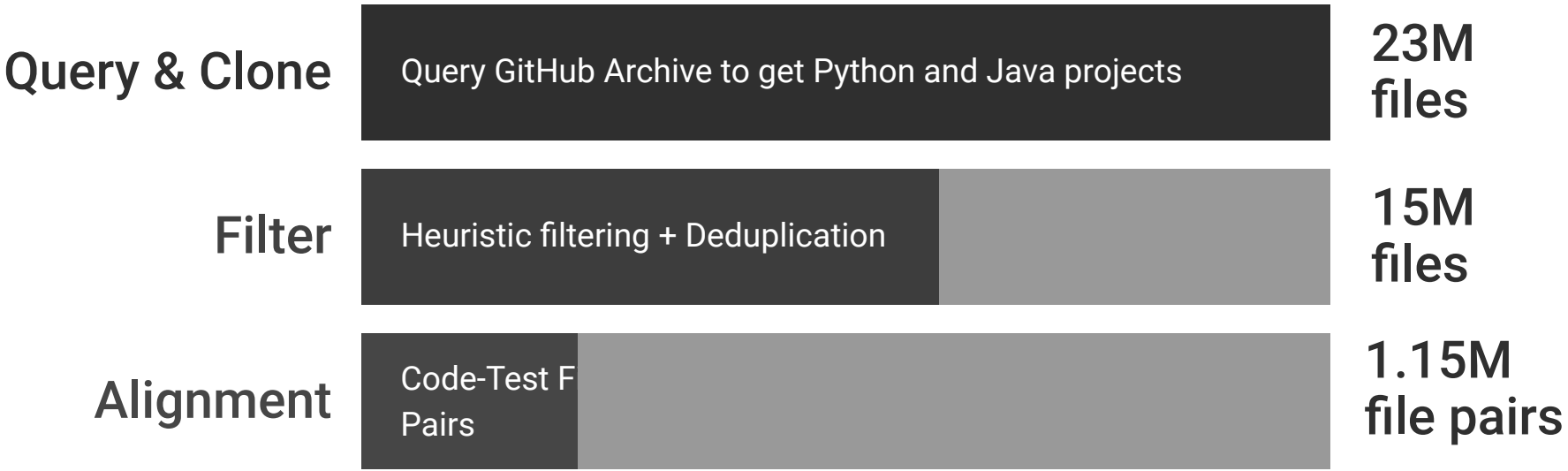
Background: Modeling Paradigms



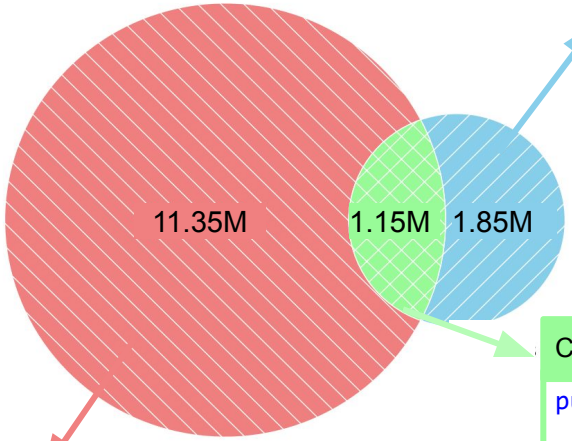




Data Mined from GitHub for Pretraining 🐱 LM



1.15M Code-Test File Pairs Found



Test Files

```
public class AppTest {
    @Test
    public void homePage() {
    ...
    }
}
```

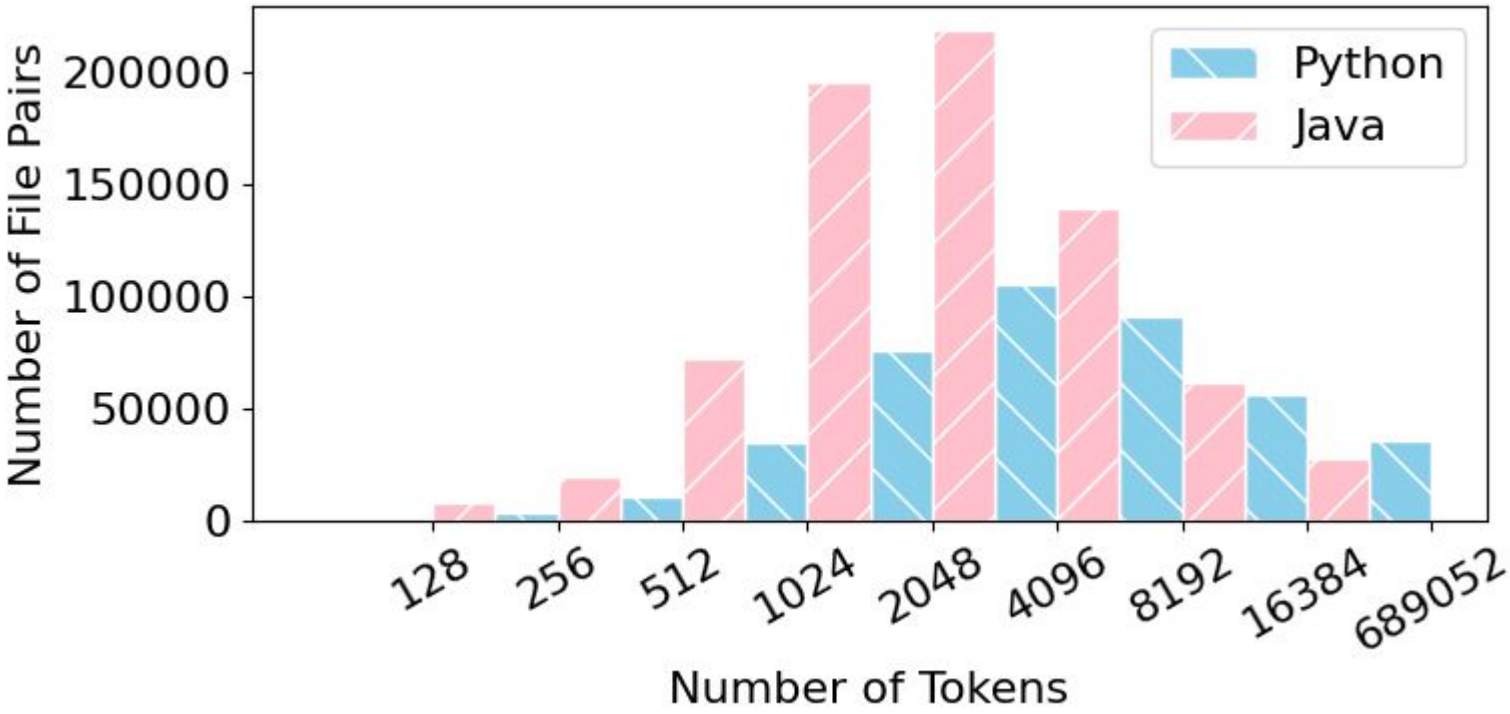
Code Files

```
public class UserController {
    public String getAllUsers() {
    ...
    }
}
```

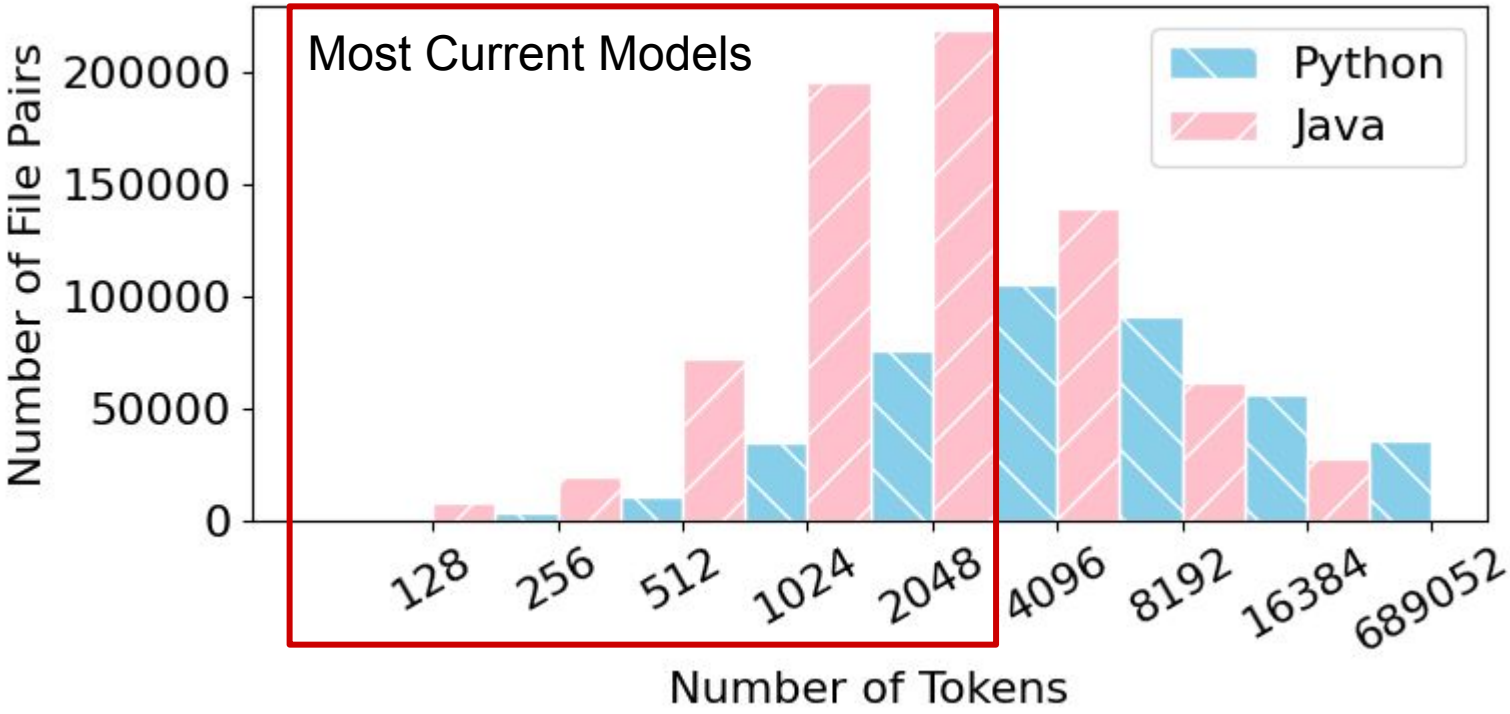
Code-Test File Pairs

```
public class Bank {
    public String customerSummary() {
    ...
    }
}
<|codetestpair|>
public class BankTest {
    @Test
    public void customerSummary() {
    ...
    }
}
```

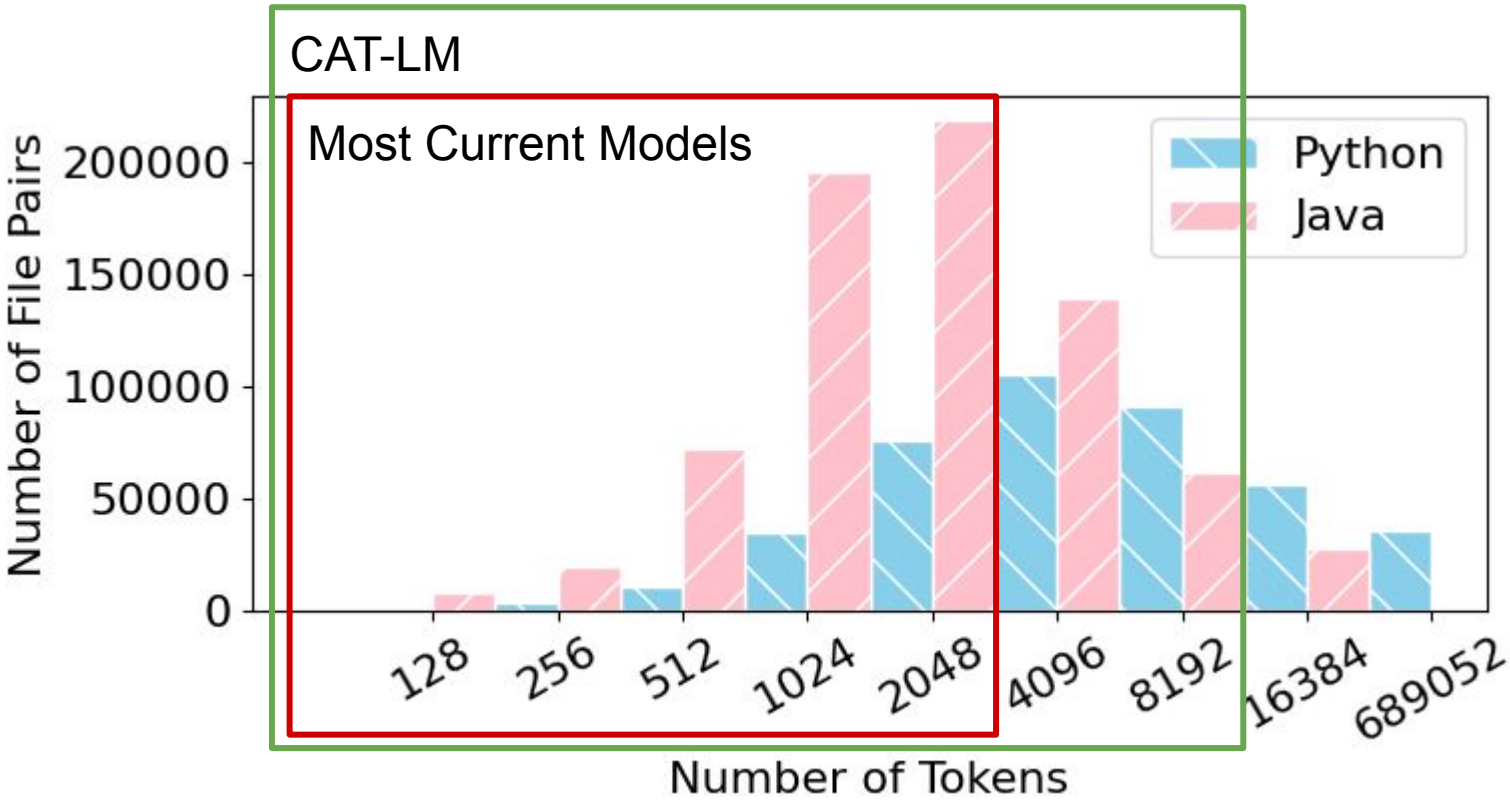
Distribution of Tokens in Code-Test File Pairs



Need for long context windows



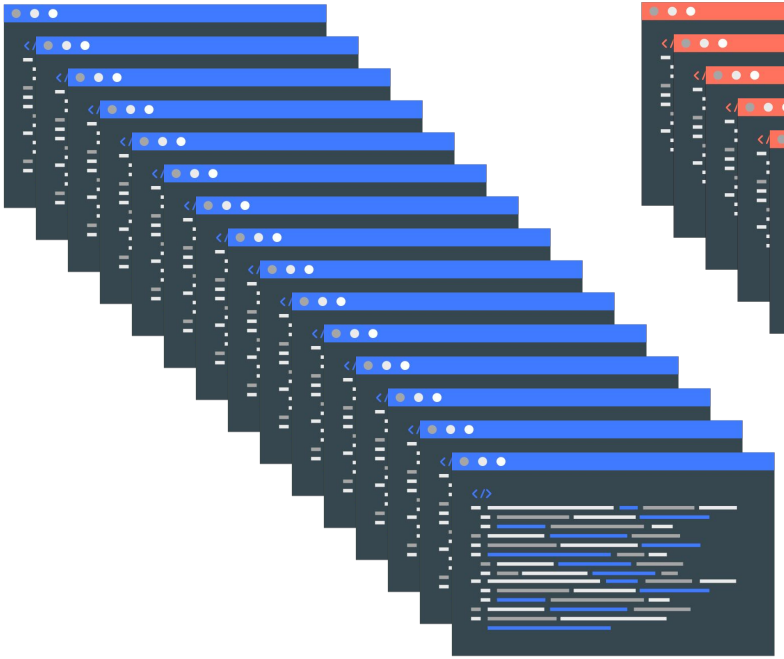
Long context windows → 16x more compute



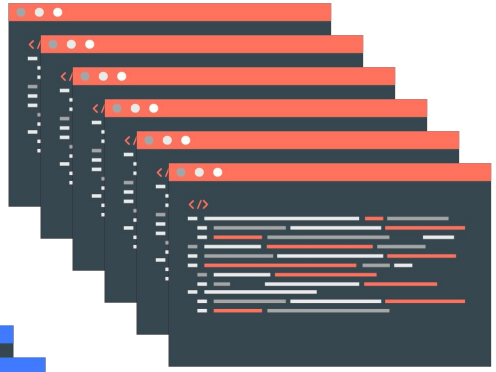
Introducing a Novel Pretraining Signal for Test Generation: Using Code Context when Available

1.15M Code-Test File Pairs

11M Code files



2M Test files

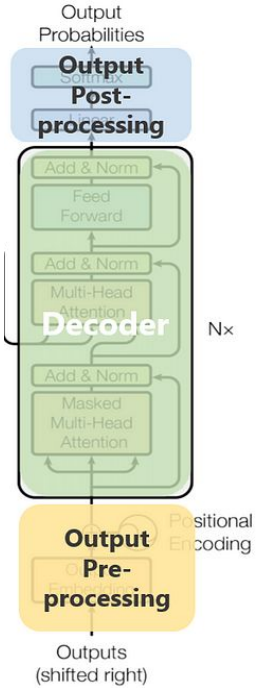


Pretraining LM

Code Files

Test Files

Code-Test File Pairs



Auto-Regressive Transformer
2.7 Billion Parameters

LM

Pretraining  LM

Model & Training Details

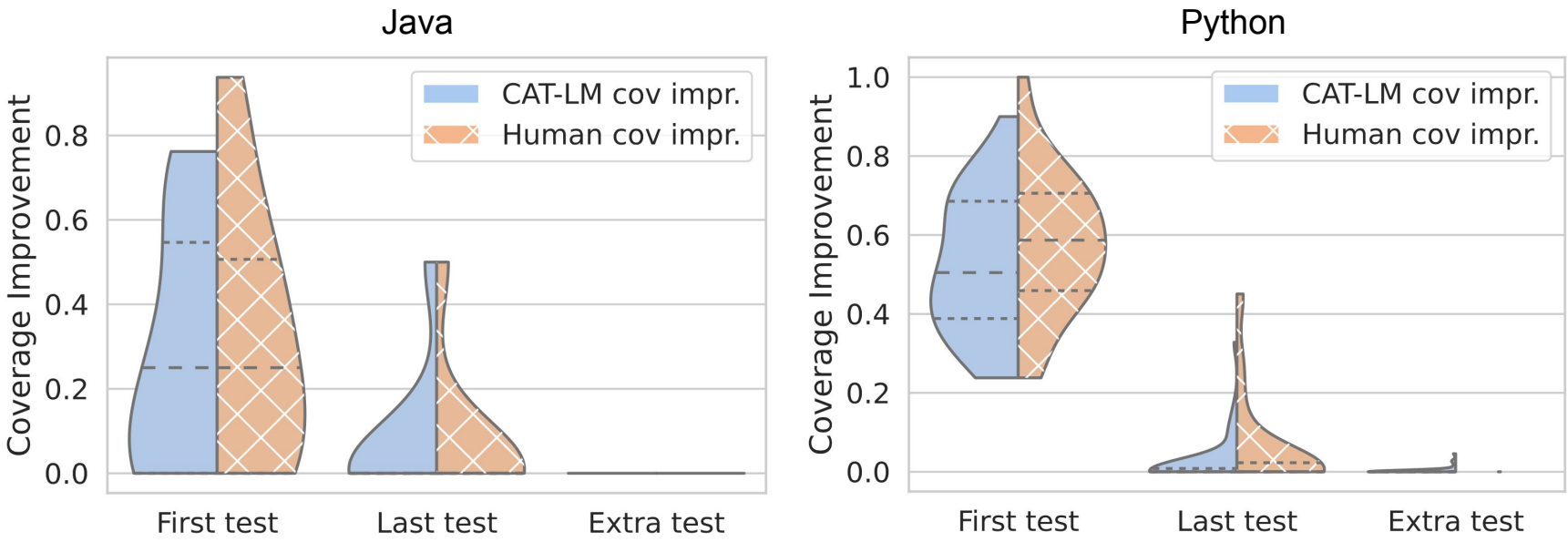
Compute (2.7B parameters)

- $25,000 cloud credits
- Megatron toolkit + Deepspeed
- Efficient training with Data Parallelism
- Chinchilla scaling laws

Large context window (8K)

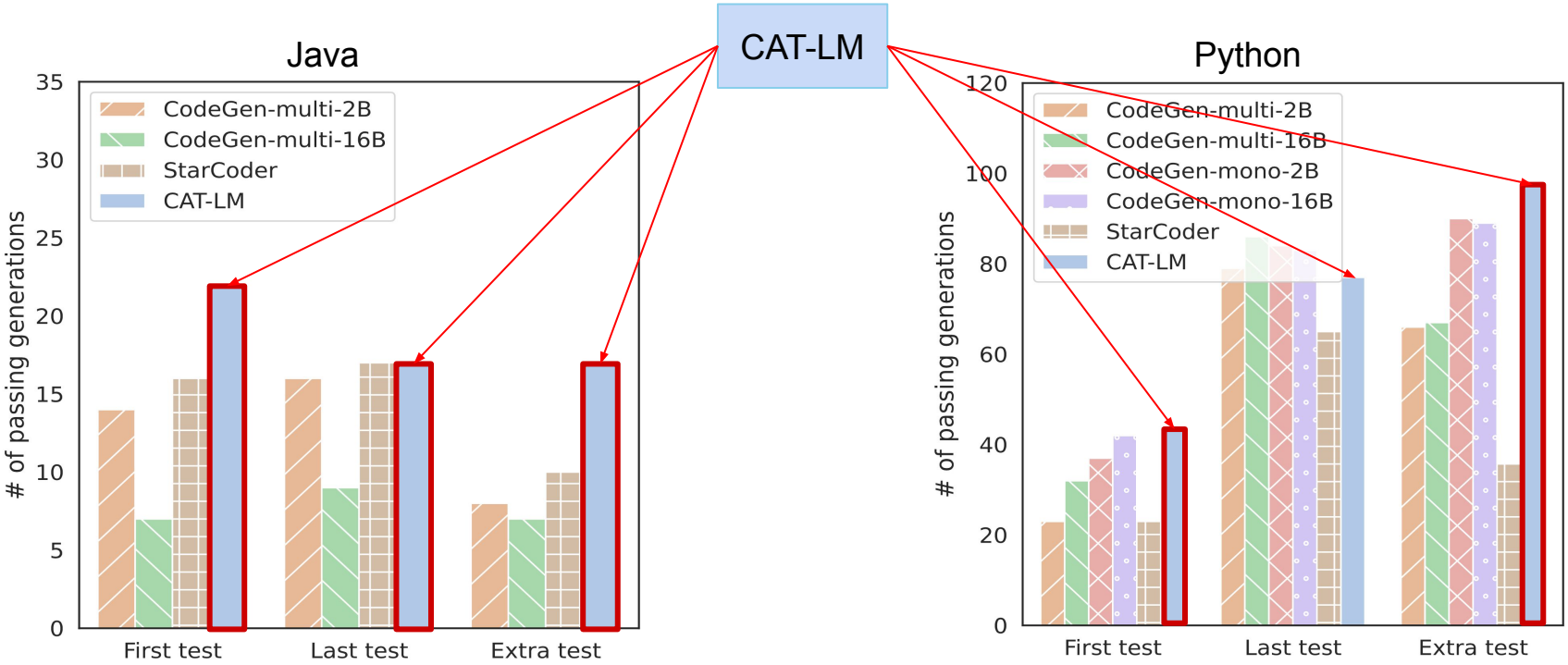
- Flash Attention
- Faster training

Tests generated by 🐱LM are comparable to humans



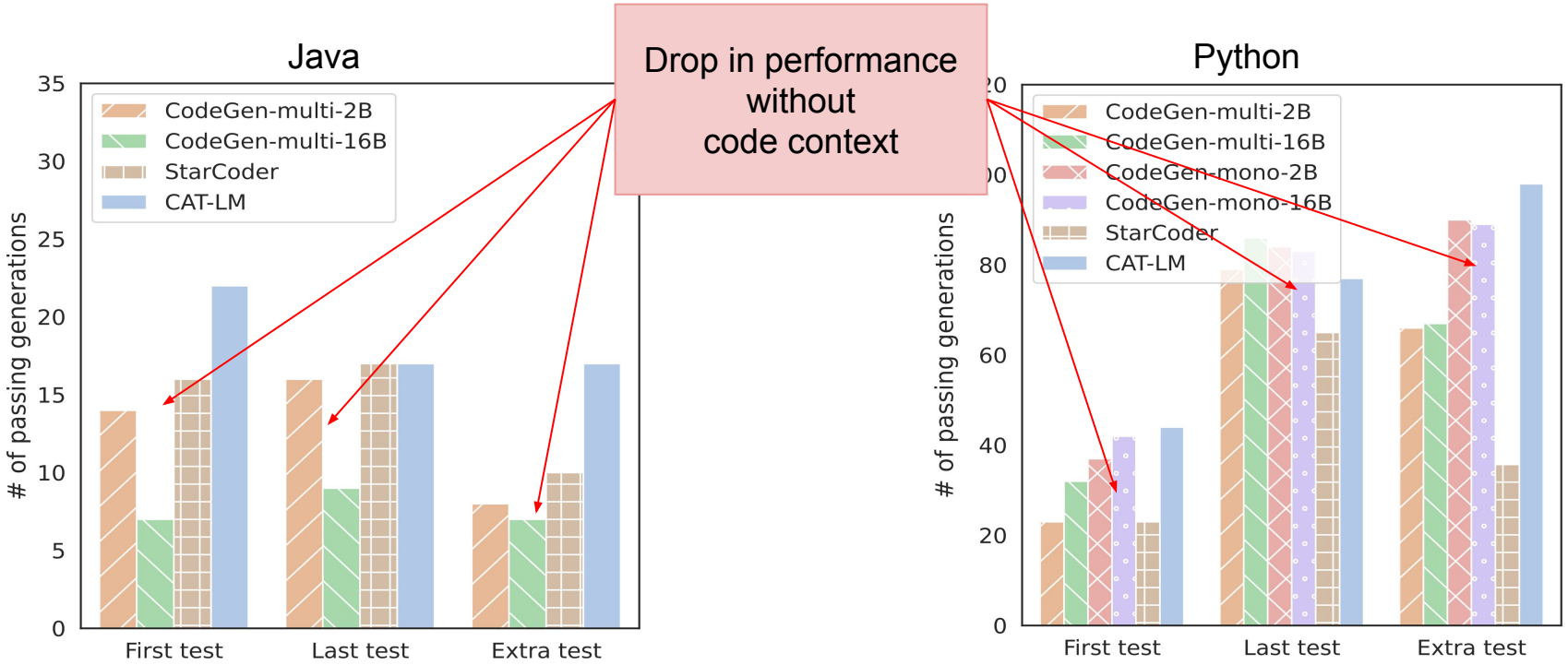
Coverage distribution is almost the same as humans!

LM generates more passing tests on average



CAT-LM outperforms much larger models trained on a lot more data!

Code context helps LM generate better tests

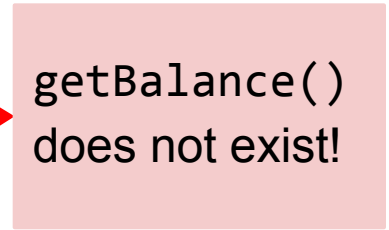


Code context helps CAT-LM generate more valid tests despite being under trained!

CodeGen generates highly readable but incorrect tests

CodeGen: Example of first test generated

```
@Test
public void testBank() {
     Bank bank = new Bank();
     assertEquals(0, bank.getBalance(), DOUBLE_DELTA);
     bank.deposit(100)
     assertEquals(100, bank.getBalance(), DOUBLE_DELTA);
     bank.withdraw(50)
     assertEquals(50, bank.getBalance(), DOUBLE_DELTA);
}
```

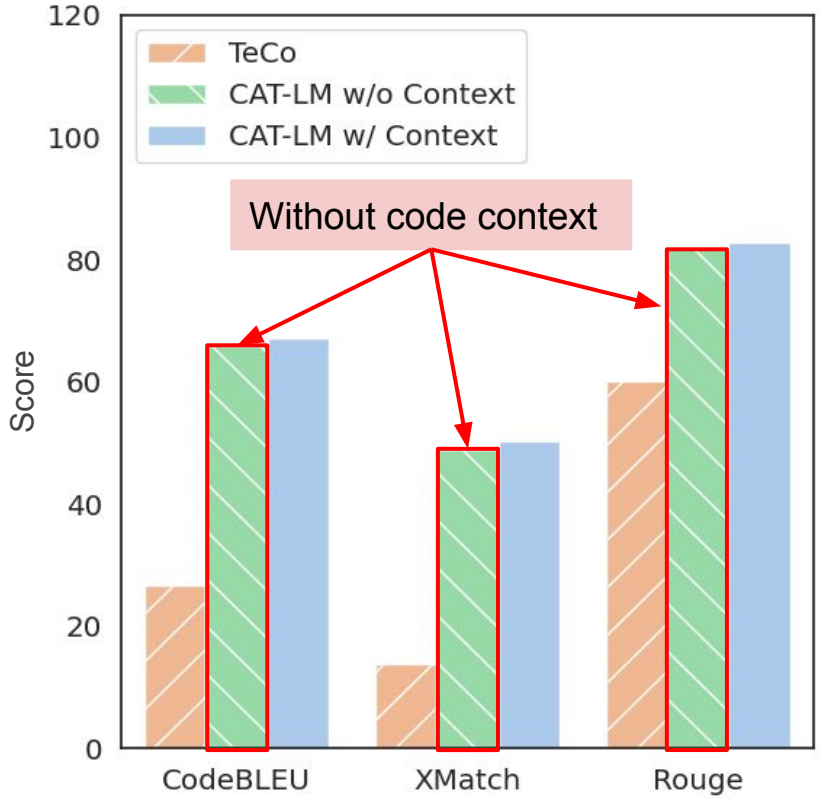


LM generates high quality tests with readable asserts

LM: Example of first test generated

```
@Test
public void testCustomerSummary() {
     Bank bank = new Bank();
     assertEquals(“Customer Summary”,
          “Customer Summary”,
          bank.customerSummary());
}
```

Test context is more helpful for test completion



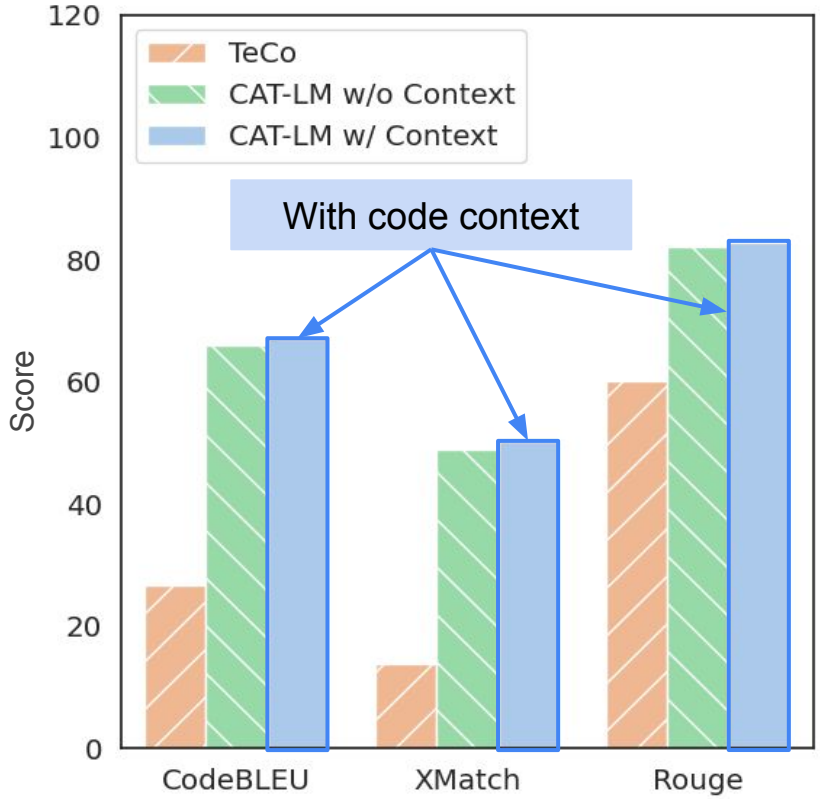

```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
public class BankTest {
    @Test
    public void FirstTest() {...}
    ...
    @Test
    public void Test_k() {
        ...
        assertNotNull(Bank());
    }
    ...
}
```

Code context

Test context

Test completion

Test context is more helpful for test completion




```
public class Bank {
    public String methodName() {...}
    ...
}
<|codetestpair|>
public class BankTest {
    @Test
    public void FirstTest() {...}
    ...
    @Test
    public void Test_k() {
        ...
        assertNotNull(Bank());
    }
    ...
}
```

Code context

Test context

Test completion

LM: Takeaways

CAT-LM outperforms much larger models trained on a lot more data.

Incorporating domain knowledge can fundamentally improve and help build more powerful models. (mapping between code-test files)

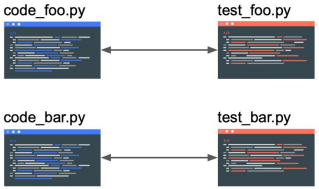
LM is on HuggingFace! Try it out

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained("nikitharao/catlm")
model = AutoModelForCausalLM.from_pretrained("nikitharao/catlm")
```

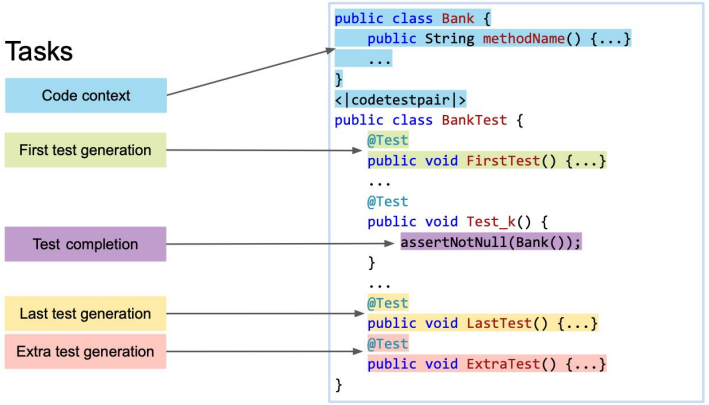
CAT-LM Training Language Models on Aligned Code And Tests

Current models fail to consider the code under test context!



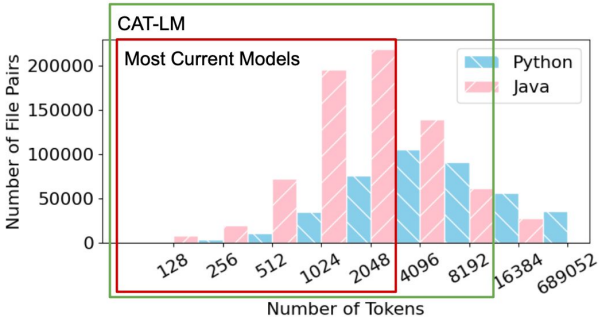
Key Insight:
We demonstrate the importance of incorporating domain knowledge when training models.

Tasks



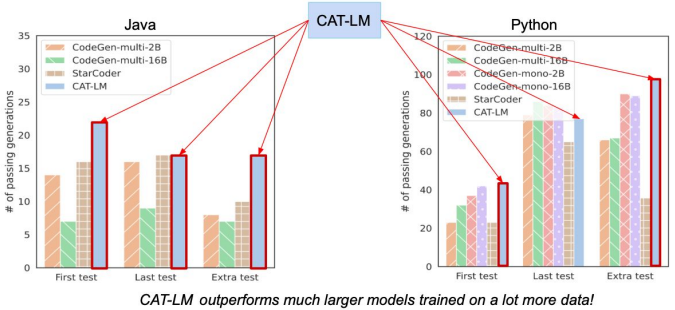
Scan me!

Long context windows → 16x more compute



10

LM generates more passing tests on average



CAT-LM outperforms much larger models trained on a lot more data!

21

Nikitha Rao | nikitharao@cmu.edu

Pretraining your own LLM from scratch can be hard,
Can we instead tune LLMs to be more aligned with our goals?

Yes! Using Alignment techniques

What is Alignment?

The process of training models to generate outputs that align with human preferences, intentions, and values, such as being more truthful or less toxic.

What are some desired goals when generating code?

1. Should follow natural language instructions
2. Generated code should execute
3. Generated code should be correct, check using tests

Overview

1. Alignment with Natural Language
   - *CodeT5+*
   - *OctoPack*
2. Alignment using Execution Signal
   - *LEVER*
   - *Self Debug*
3. Alignment with Tests
   - *CodeT*
   - *RLTF*

Overview

1. **Alignment with Natural Language**
   - *CodeT5+*
   - *OctoPack*
2. Alignment using Execution Signal
   - *LEVER*
   - *Self Debug*
3. Alignment with Tests
   - *CodeT*
   - *RLTF*

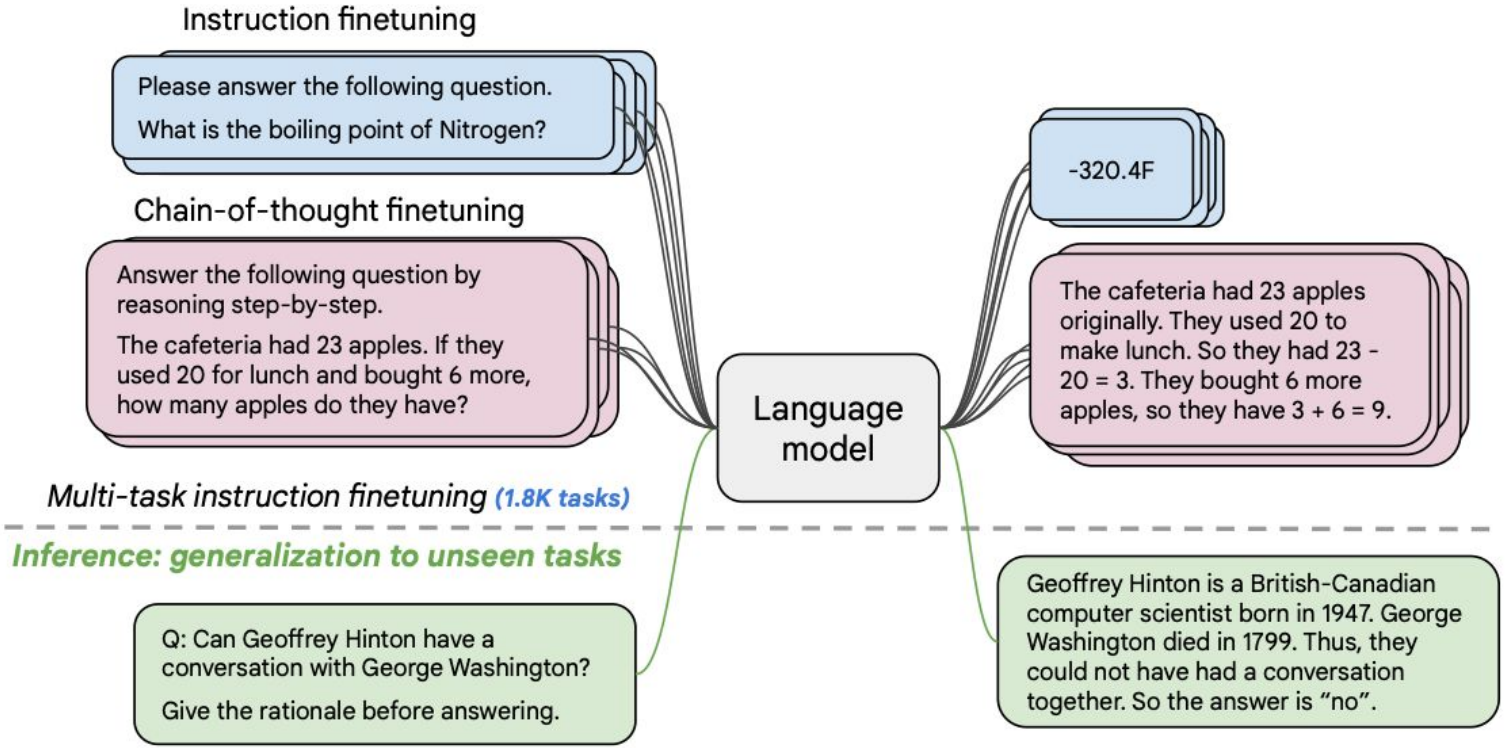
CodeT5+: Motivation

Current LLMs are designed to perform well for specific tasks and are limited by their

- Architecture
  - Encoder only or Decoder only models are limited to certain applications.
  - Unified Encoder-Decoder models have suboptimal performance on some tasks.

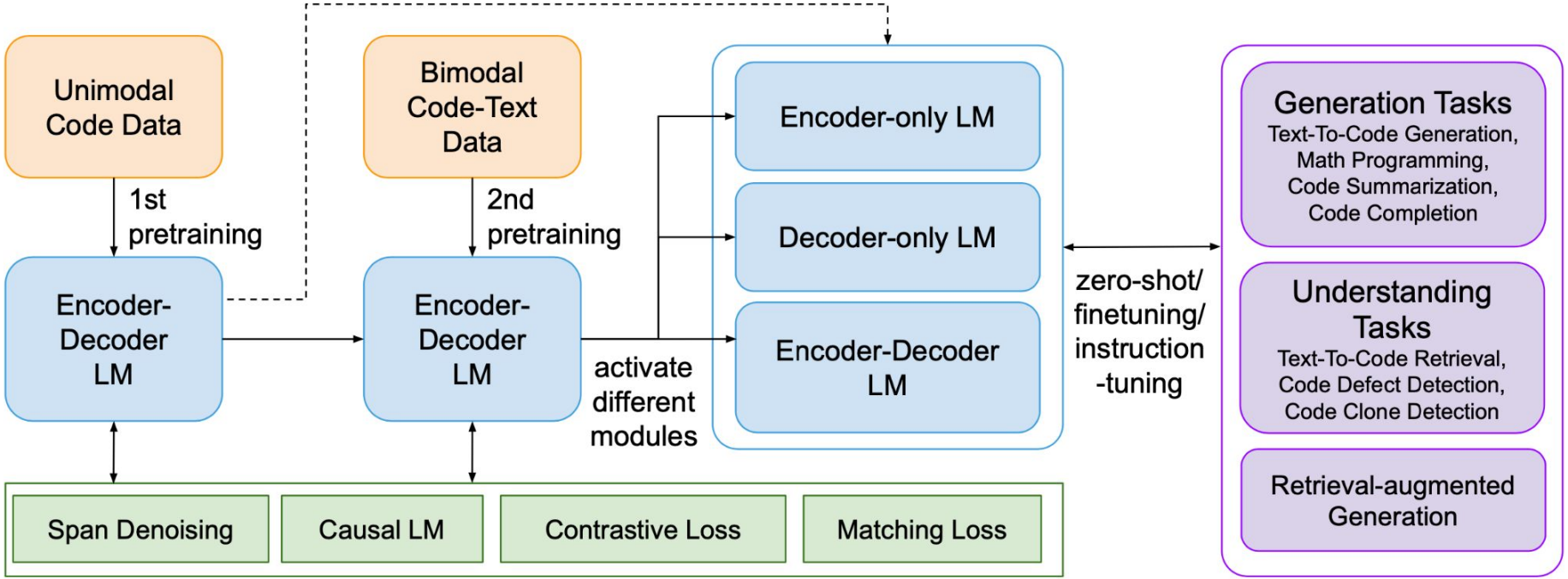
- Pretraining signal
  - Specific objectives like span denoising, next token prediction.
  - May lead to degraded performance for certain downstream tasks when finetuning

Recall: Instruction Tuning



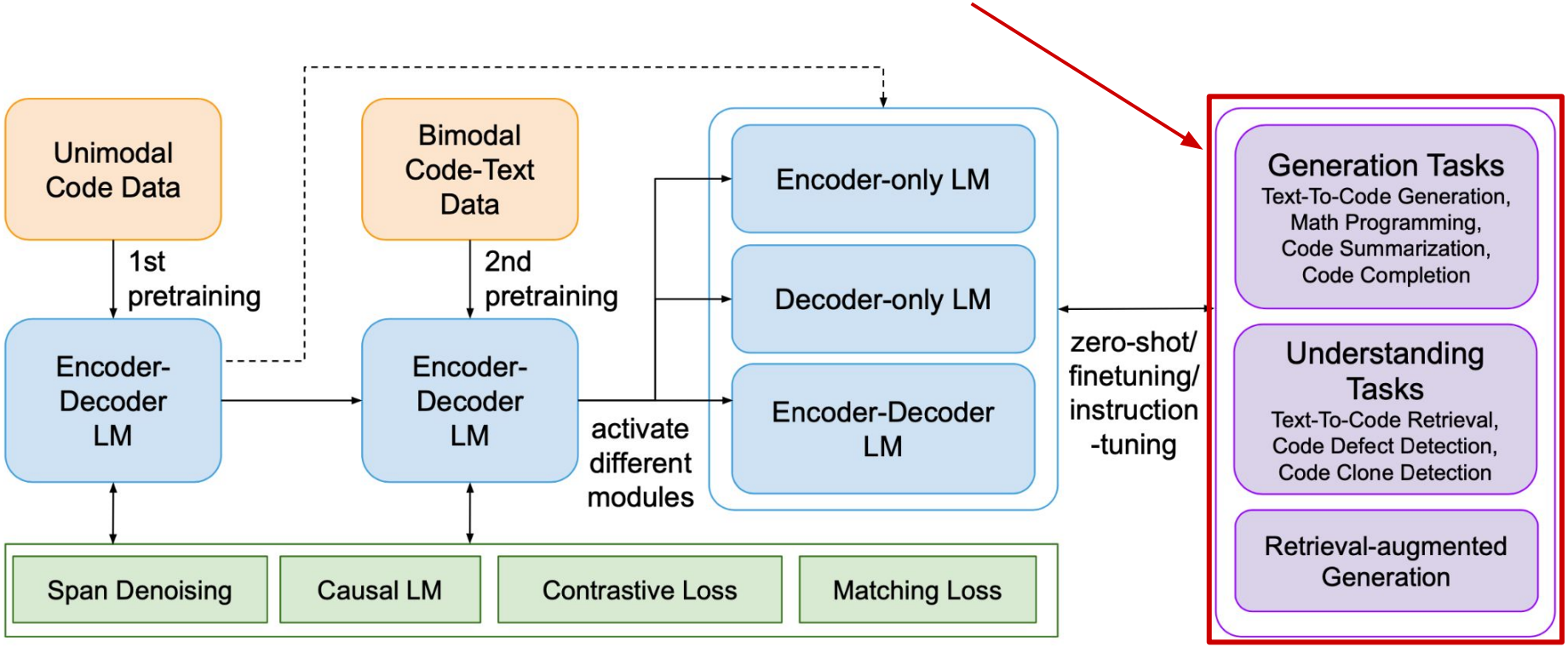
FlanT5 (Chung et al., 2022), https://arxiv.org/pdf/2210.11416.pdf

CodeT5+: Overview

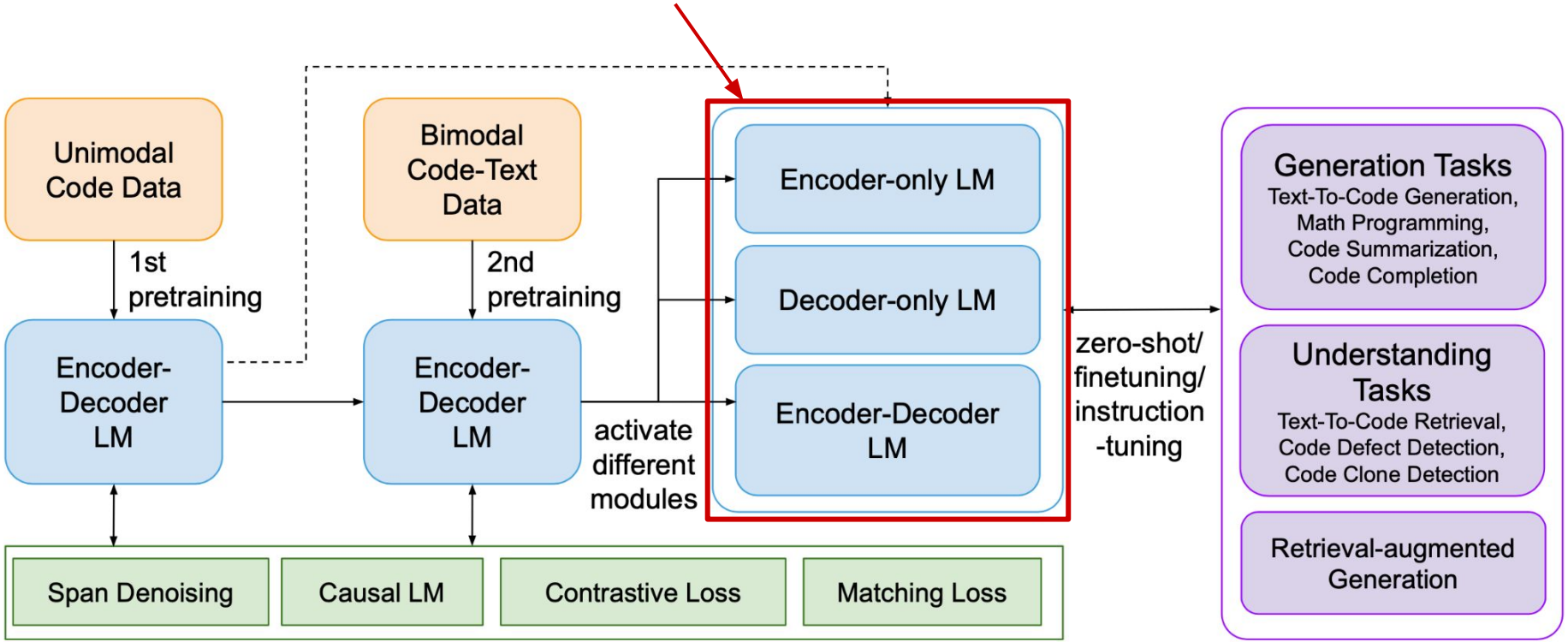


CodeT5+, https://arxiv.org/abs/2305.07922

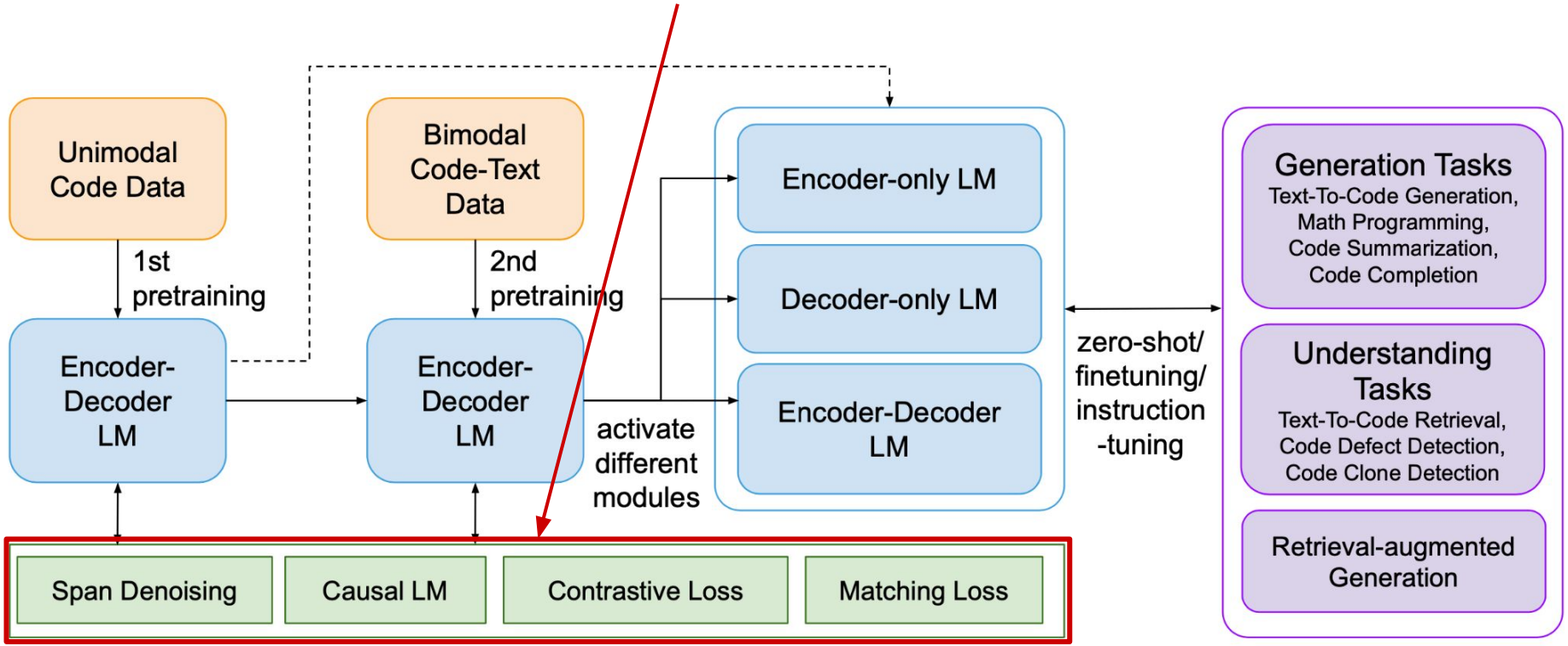
CodeT5+: Supports various downstream tasks



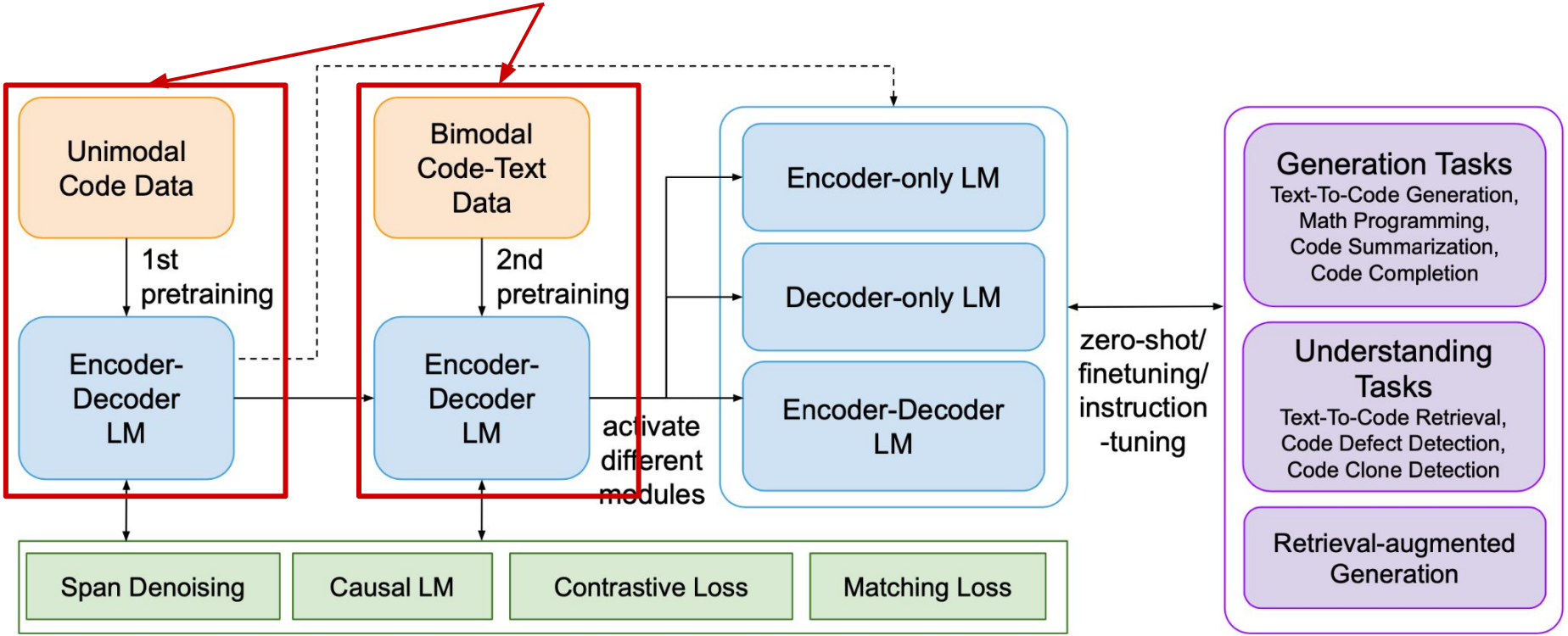
CodeT5+: Can operate is different modes



CodeT5+: Uses several different pretraining tasks



CodeT5+: Has two pretraining stages



Stage 1: Unimodal Pretraining

Goal: Train model to recover code contexts at different scales

Data: Code from GitHub

Tasks:

- Span Denoising (15% masked tokens)
- Causal LM
  - Partial programs
  - Complete programs

Stage 2: Bimodal Pretraining

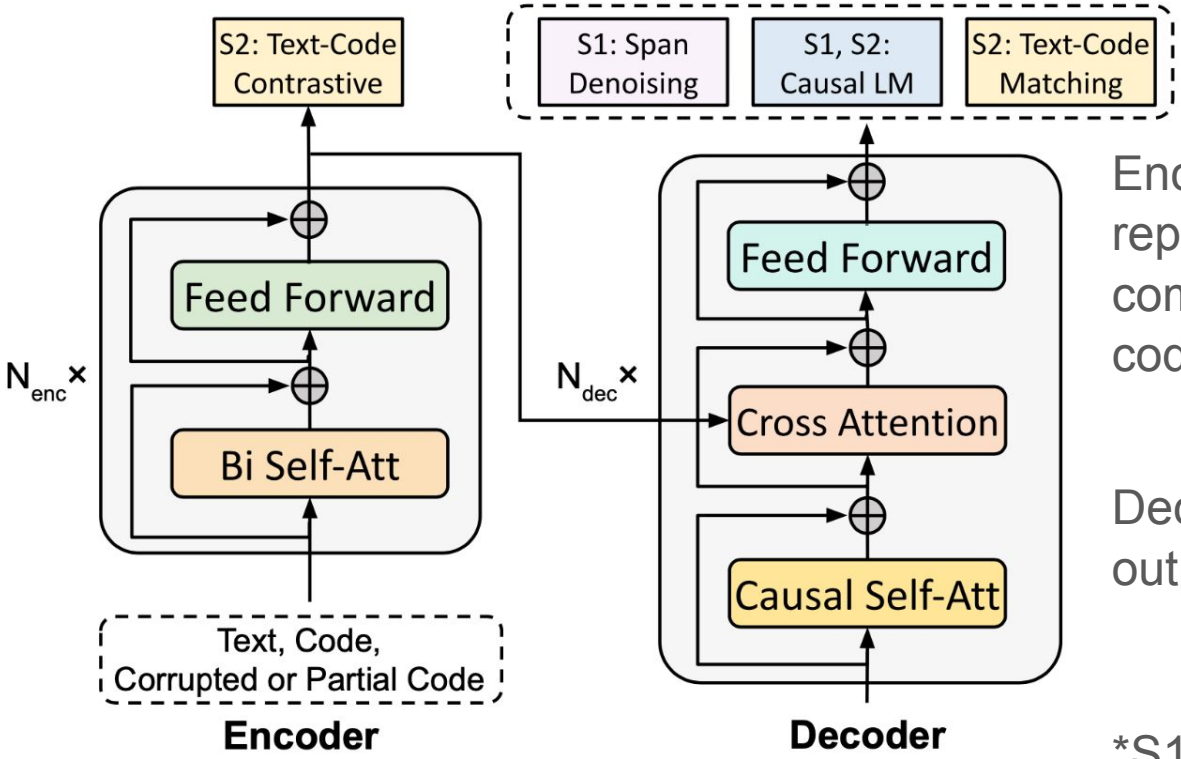
Goal: Train model for cross-modal understanding and generation

Data: CodeSearchNet (Docstring & Code)

Tasks:

- Contrastive Learning (align feature space of code and text representation)
- Text-Code Matching (predict if semantics match)
- Text-Code Causal LM (text-to-code and code-to-text generation)

CodeT5+: Model Architecture



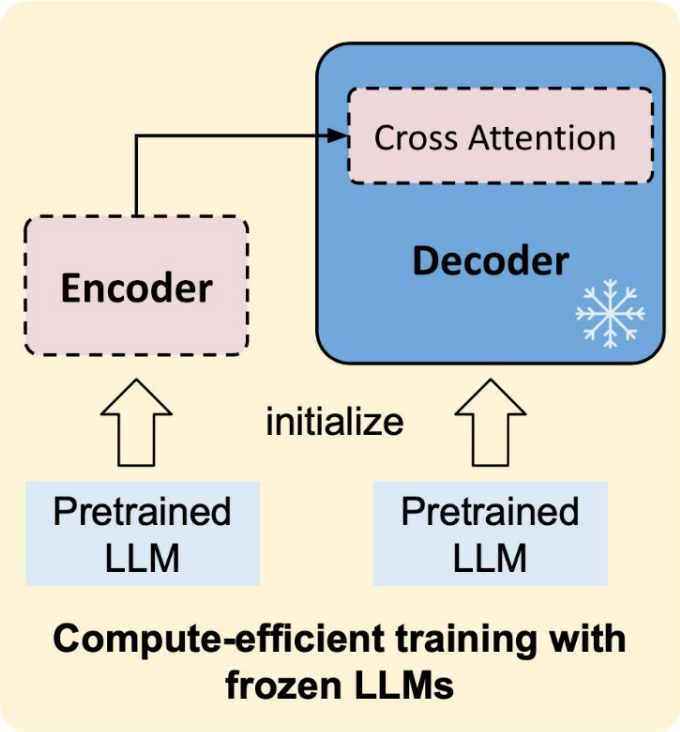
Encoder: encodes contextual representations from either complete, partial or span-masked code/text sequences

Decoder: generates different outputs based on pretraining task

*S1: Stage1 and S2: Stage2

CodeT5+: Compute Efficient Training

- Shallow encoder and deep decoder, initialised with pretrained CodeGen-mono weights.
- Only encoder and cross attention layers are trainable
- Decoder weights are frozen



InstructCodeT5+: Tuned with over 20K generated instruction data

Instruction: Create a SQL query to get the list of employee names and ids with a monthly income greater than 4,000.
Input: n/a
Output:
```
SELECT id, name FROM Employees WHERE
monthly_income > 4000;
```

Instruction: Write a code to add two numbers without using the \"+\" operator.
Input:
```
num1 = 2
num2 = 7
```
Output:
```
num1 = 2
num2 = 7
# Without using \"+\" operator
sum = num1 - (-num2)
```

Instruction: Optimize the given Python program to improve the speed of execution.

Input:
```
def calc( num1, num2):
    result = 0
    for i in range(num1, num2):
       result += ((i+1) * (i+2))
    return result
```

Output:
```
def calc(num1, num2):
    result = (num1 + num2) * (num2
- num1 + 1) // 2
    return result
```

Dataset generated by having text-davinci-003 generate novel tasks with task instruction, inputs (if any) and expected outputs. (Dataset released part of Code Alpaca)

Code Alpaca: https://github.com/sahil280114/codealpaca

CodeT5+: Evaluation

Task: Zero shot text to code generation

Benchmark: HumanEval (164 python problems)

Finding: InstructCodeT5+ 16B achieves SOTA performance against other open code LLMs

Model	Model size	pass@1	pass@10	pass@100
Closed-source models				
LaMDA	137B	14.0	-	47.3
AlphaCode	1.1B	17.1	28.2	45.3
MIM	1.3B	22.4	41.7	53.8
MIM	2.7B	30.7	48.2	69.6
PaLM	8B	3.6	-	18.7
PaLM	62B	15.9	-	46.3
PaLM	540B	26.2	-	76.2
PaLM-Coder	540B	36.0	-	88.4
Codex	2.5B	21.4	35.4	59.5
Codex	12B	28.8	46.8	72.3
code-cushman-001	-	33.5	54.3	77.4
code-davinci-002	-	47.0	**74.9**	**92.1**
GPT-3.5	-	48.1	-	-
GPT-4	-	**67.0**	-	-
Open-source models				
GPT-Neo	2.7B	6.4	11.3	21.4
GPT-J	6B	11.6	15.7	27.7
GPT-NeoX	20B	15.4	25.6	41.2
InCoder	1.3B	8.9	16.7	25.6
InCoder	6B	15.2	27.8	47.0
CodeGeeX	13B	22.9	39.6	60.9
LLaMA	7B	10.5	-	36.5
LLaMA	13B	15.8	-	52.5
LLaMA	33B	21.7	-	70.7
LLaMA	65B	23.7	-	**79.3**
Replit	3B	21.9	-	-
StarCoder	15B	33.6	-	-
CodeGen-mono	2B	23.7	36.6	57.0
CodeGen-mono	6B	26.1	42.3	65.8
CodeGen-mono	16B	29.3	49.9	75.0
CodeT5+	220M	12.0	20.7	31.6
CodeT5+	770M	15.5	27.2	42.7
CodeT5+	2B	24.2	38.2	57.8
CodeT5+	6B	28.0	47.2	69.8
CodeT5+	16B	30.9	51.6	76.7
InstructCodeT5+	16B	**35.0**	**54.5**	77.9

CodeT5+: Additional Evaluation

Tasks:

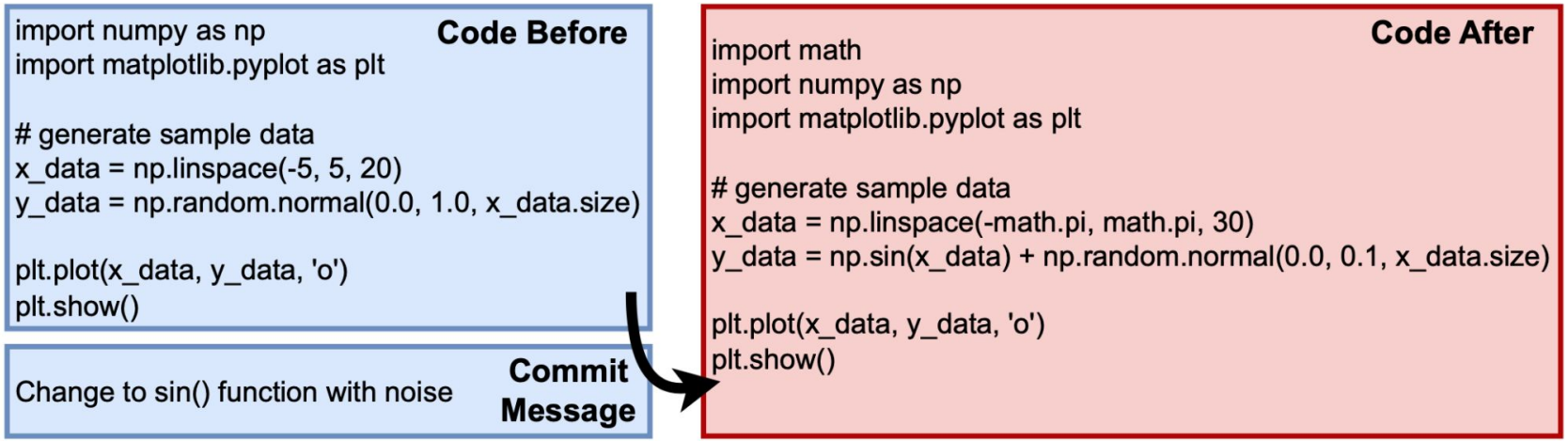
- Math Programming (MathQA-Python, GSM8K-Python)
- Code Summarization (CodeSearchNet)
- Line-level Code Completion (PY150, JavaCorpus)
- Text-to-Code Retrieval (MRR on CodeXGLUE)

Finding: CodeT5+ (220M and 770M) outperforms other baseline models (some much larger) across all other tasks.

Problem with instruction tuning models on generated data

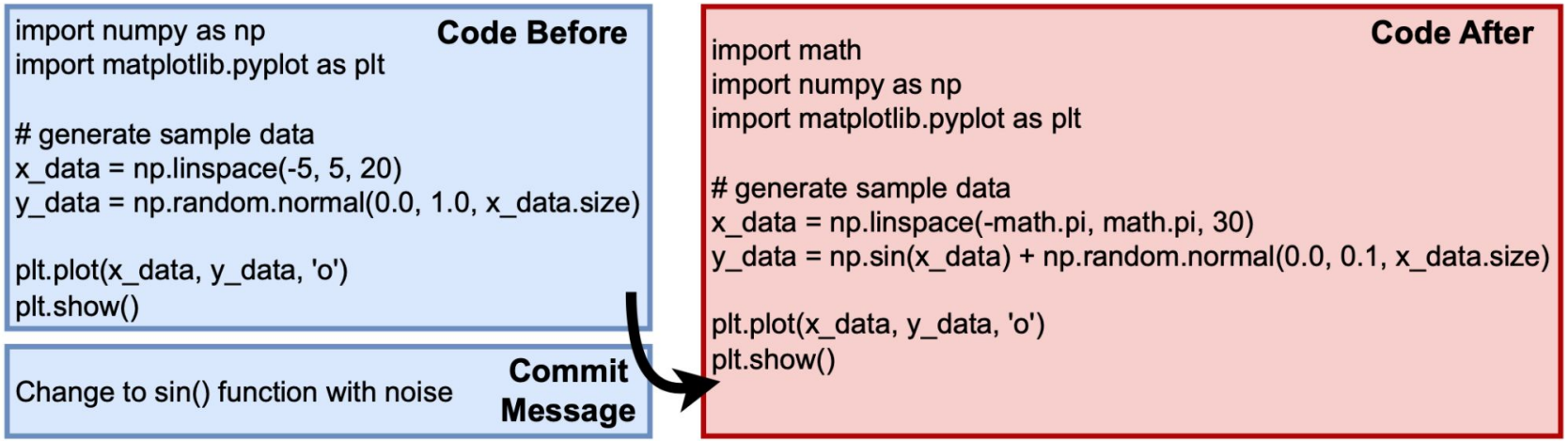
- Synthetic data generated by OpenAI models
- Can reinforce hallucinations
- Closed source APIs can change and have unpredictable availability

Solution: Use git commit messages as instructions!



OctoPack, https://arxiv.org/abs/2308.07124

OctoPack: Uses git commit messages as instructions



CommitPack: 4TB dataset of permissively licensed code commits across 350 programming languages for pretraining!

CommitPackFT: 2GB filtered variant containing high-quality code instructions for finetuning.

Evaluation

HumanEvalPack

Languages: Python, JavaScript, Java, Go, C++, Rust
Subtasks: HumanEvalFix, HumanEvalExplain, HumanEvalSynthesize
Metric: Pass@k
Creation: Humans

Fix Code

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = elem - elem2
                if distance < threshold:
                    return True

    return False

def check(has_close_elements):
    assert has_close_elements([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) ==
True
    assert has_close_elements([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) ==
False
    assert has_close_elements([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
    assert has_close_elements([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
    assert has_close_elements([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) ==
True
    assert has_close_elements([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
    assert has_close_elements([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False

check(has_close_elements)

Fix bugs in has_close_elements.
```

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:
    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
```

Explain Code

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool: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

Provide a concise natural language description of the function using
at most 213 characters.
```

```
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
```

```
Check if in given list of numbers, are any...
...
Write functional code in Python according to the description.
```

```
from typing import List


def has_close_elements(numbers: List[float], threshold: float) ->
bool:
    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
```

Synthesize Code

```
Write a Python function `has_close_elements(numbers: List[float],
threshold: float) -> bool` to solve the following problem:
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
```

```
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
```

Model Input

Target Output

HumanEvalPack:

1. HumanEvalFix: Fix subtle bug in code function given unit tests. Manually added the bugs across all languages, 984 total bugs. Buggy code still runs, but tests fail.

2. HumanEvalExplain: Generate NL explanation for given code function. Then regenerate code using NL explanation. Allows you to evaluate regenerated code with pass@k instead of BLEU.

3. HumanEvalSythesize: the original HumalEval benchmark

OctoPack: Models

1. OctoCoder: Instruction tuned StarCoder (15B)
2. OctoGeeX: Instruction tuned CodeGeeX2 (6B)

Data used: CommitPackFT (5k samples) + OASST (~8.5k samples)

OASST: diverse dataset of multi-turn chat dialogues (mostly NL, some Code) with some filters.

OctoPack: Evaluation & Findings

- OctoCoder performs best among all permissive models.
- Instruction tuning generalizes to unseen programming languages. Non zero scores for Go and Rust which are not present in BLOOMZ's instruction data.
- OctoCoder has a 38% relative improvement over StarCoder on the HumanEvalSynthesize (original HumanEval) benchmark.

Model (↓)	Python	JavaScript	Java	Go	C++	Rust	Avg.
HUMANEVALFIX							
Non-permissive models							
InstructCodeT5+†	2.7	1.2	4.3	2.1	0.2	0.5	1.8
WizardCoder†	31.8	29.5	30.7	30.4	18.7	13.0	25.7
GPT-4	47.0	48.2	50.0	50.6	47.6	43.3	47.8
Permissive models							
BLOOMZ	16.6	15.5	15.2	16.4	6.7	5.7	12.5
StarChat-β	18.1	18.1	24.1	18.1	8.2	3.6	11.2
CodeGeeX2*	15.9	14.7	18.0	13.6	4.3	6.1	12.1
StarCoder	8.7	15.7	13.3	20.1	15.6	6.7	13.4
OCTOGEEX*	28.1	27.7	30.4	27.6	22.9	9.6	24.4
OCTOCODER	**30.4**	**28.4**	**30.6**	**30.2**	**26.1**	**16.5**	**27.0**
HUMANEVALEXPLAIN							
Non-permissive models							
InstructCodeT5+†	20.8	0.0	0.0	0.0	0.1	0.0	3.5
WizardCoder†	32.5	33.0	27.4	26.7	28.2	16.9	27.5
GPT-4	64.6	57.3	51.2	58.5	38.4	42.7	52.1
Permissive models							
BLOOMZ	14.7	8.8	12.1	8.5	0.6	0.0	7.5
StarChat-β	25.4	21.5	24.5	18.4	17.6	13.2	20.1
CodeGeeX2*	0.0	0.0	0.0	0.0	0.0	0.0	0.0
StarCoder	0.0	0.0	0.0	0.0	0.0	0.0	0.0
OCTOGEEX*	30.4	24.0	24.7	**21.7**	21.0	**15.9**	22.9
OCTOCODER	**35.1**	**24.5**	**27.3**	21.1	**24.1**	14.8	**24.5**
HUMANEVALSYNTHESIZE							
Non-permissive models							
InstructCodeT5+†	37.0	18.9	17.4	9.5	19.8	0.3	17.1
WizardCoder†	57.3	49.5	36.1	36.4	40.9	20.2	40.1
GPT-4	86.6	82.9	81.7	72.6	78.7	67.1	78.3
Permissive models							
BLOOMZ	15.6	14.8	18.4	8.4	6.5	5.5	11.5
StarChat-β	33.5	31.4	26.7	25.5	26.6	14.0	26.3
CodeGeeX2*	35.9	32.2	30.8	22.5	29.3	18.1	28.1
StarCoder	33.6	30.8	30.2	17.6	31.6	21.8	27.6
OCTOGEEX*	44.7	33.8	36.9	21.9	32.3	15.7	30.9
OCTOCODER	**46.2**	**39.2**	**38.2**	**30.4**	**35.6**	**23.4**	**35.5**

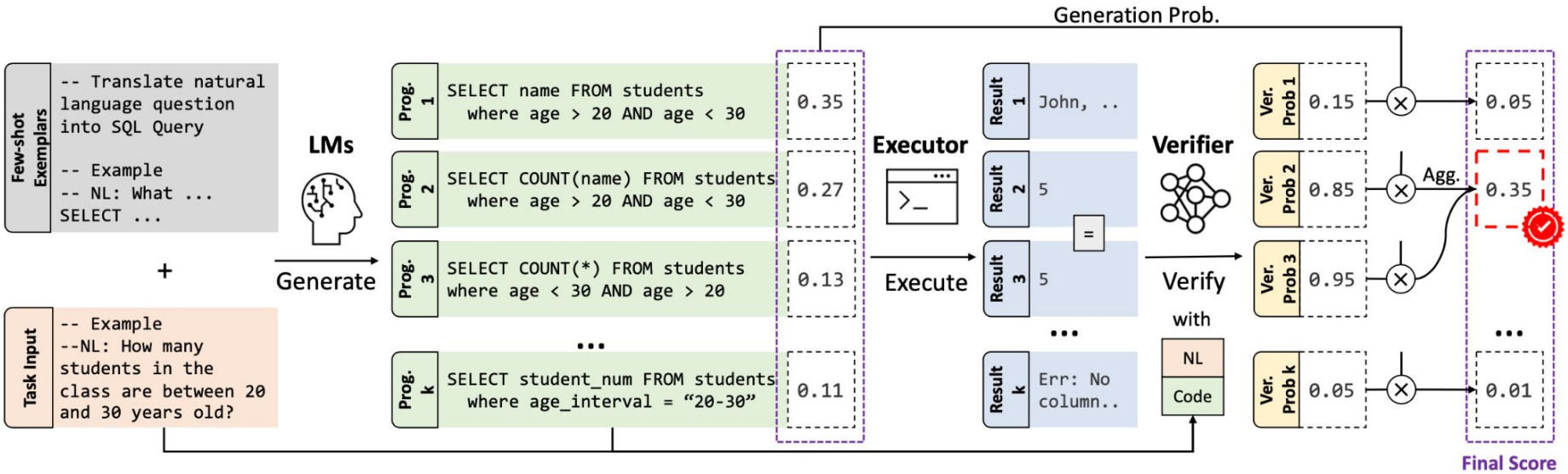
Overview

1. Alignment with Natural Language
   - *CodeT5+*
   - *OctoPack*

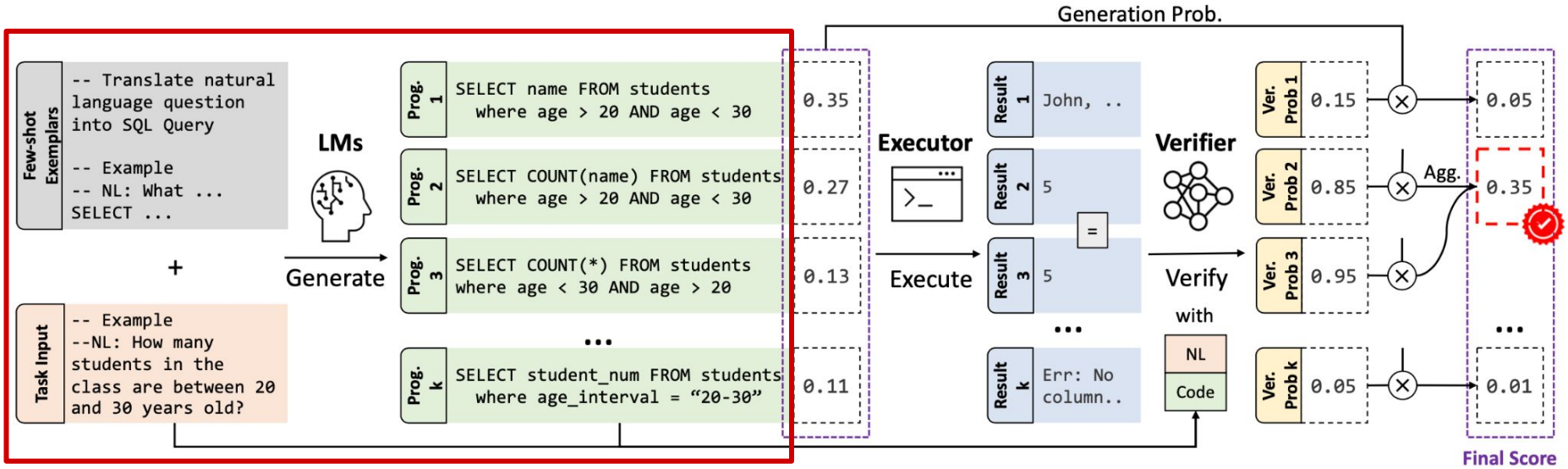
2. **Alignment using Execution Signal**
   - *LEVER*
   - *Self Debug*

3. Alignment with Tests
   - *CodeT*
   - *RLTF*

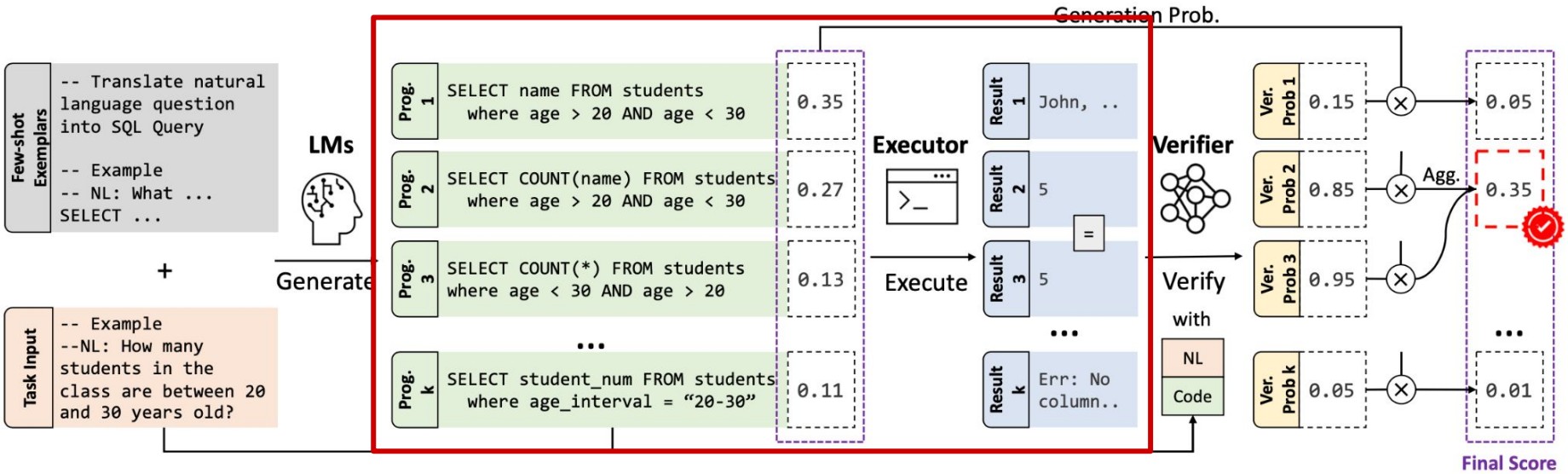
LEVER: Learning to Verify Language-to-Code Generation with Execution



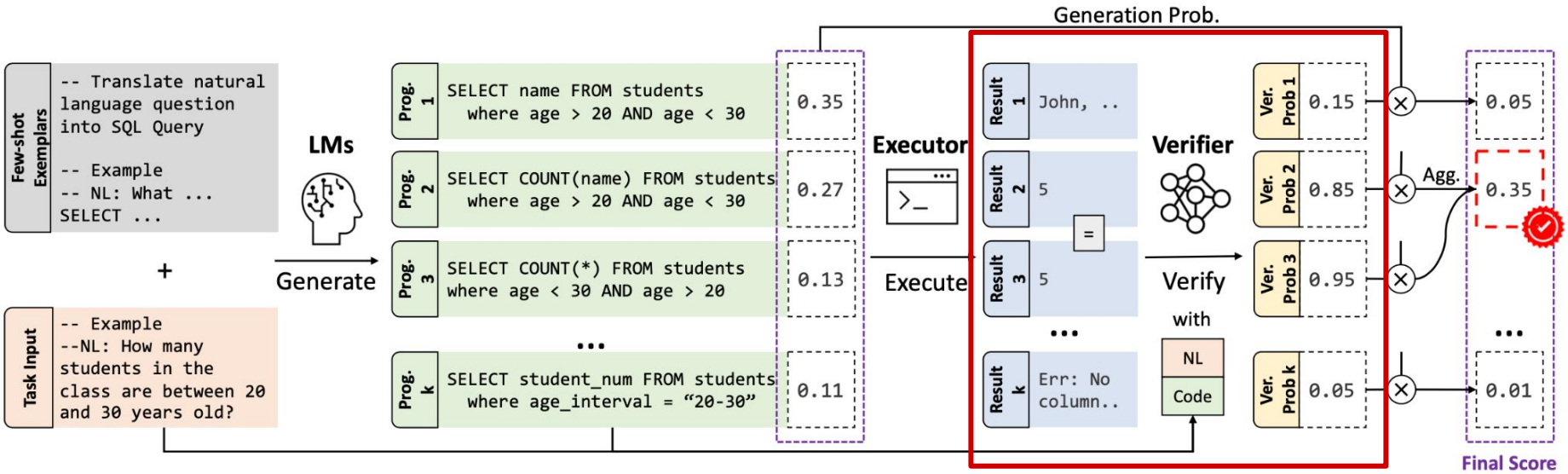
Generation: sample programs based on task input and few-shot examples



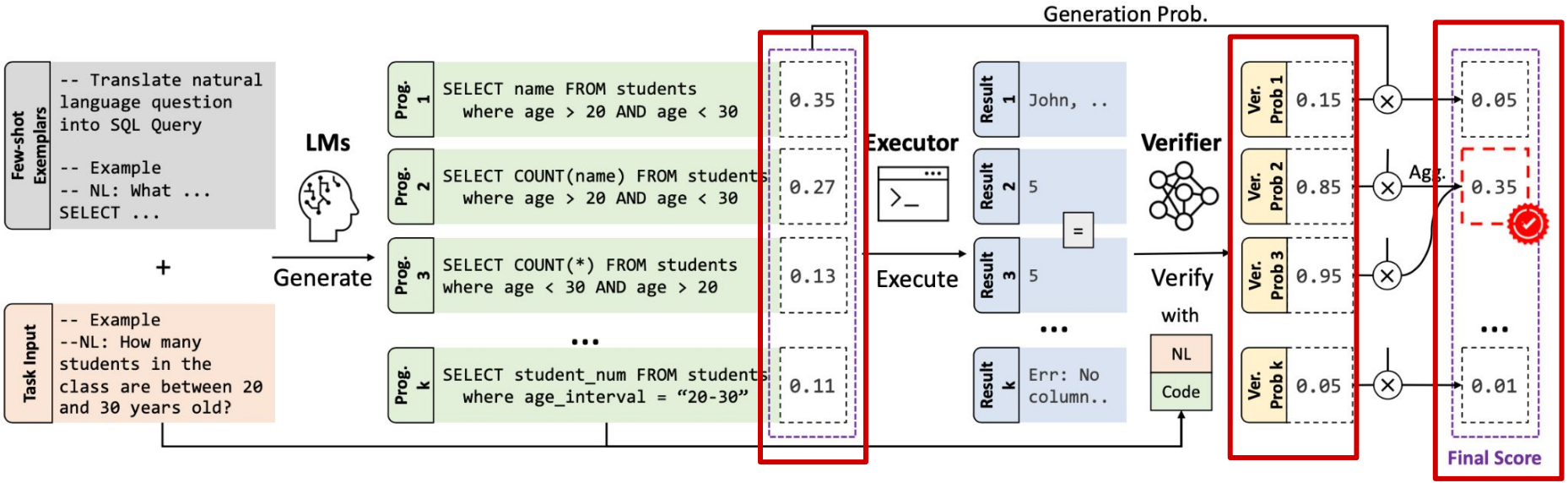
Execution: Obtain execution results with program executors



Verification: using a learned verifier to output the probability of the program being correct based on the NL, program and execution results.



Programs are reranked by combining the verification score with the LLM generation probability, and marginalizing over programs with the same execution results



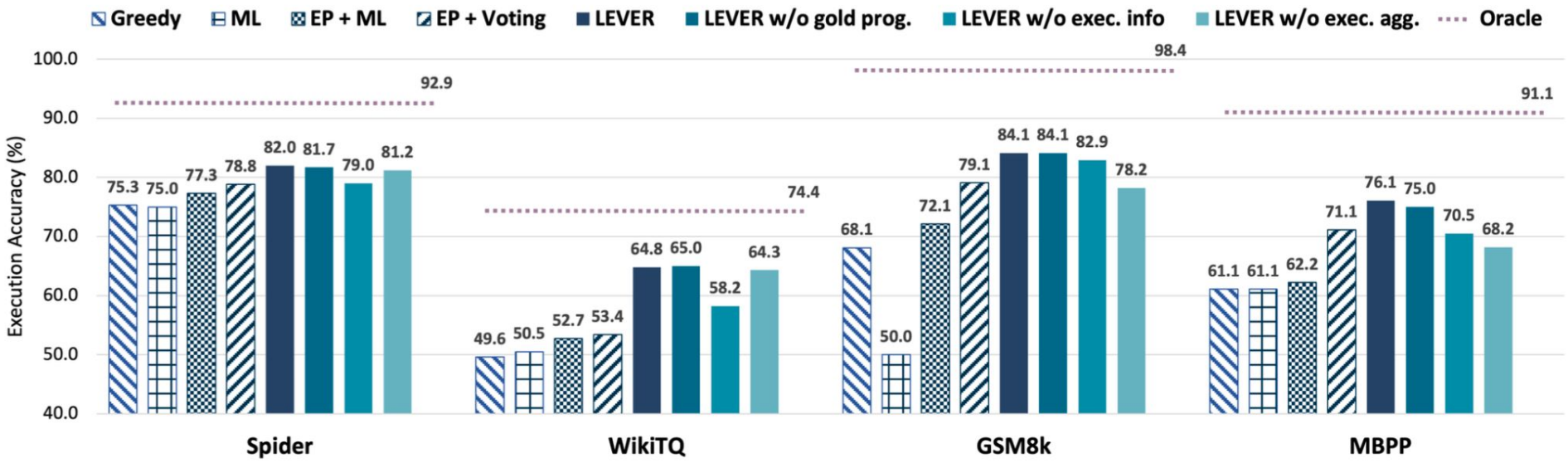
LEVER: Evaluation

Tasks:

- Spider (NL to SQL)
- WikiTableQuestions (NL to SQL)
- GSM8k (Math problems to Python)
- MBPP (NL to Python)

Base Models: Codex (code-davinci-002), InCoder, CodeGen

LEVER: Evaluation

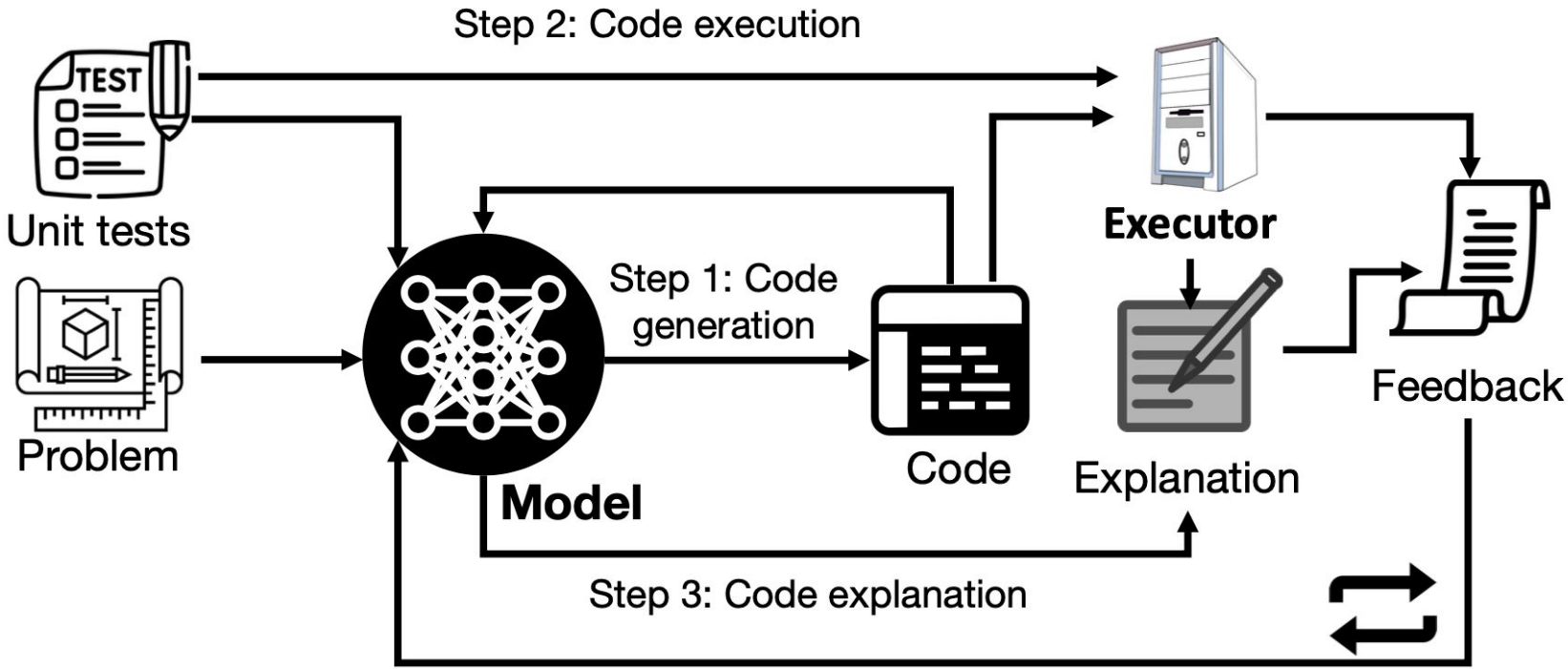


Finding: LEVER consistently improves the performance of all code LLMs on all tasks, even outperforms various finetuned models

LEVER assumes that given enough samples,
at least one program is correct.

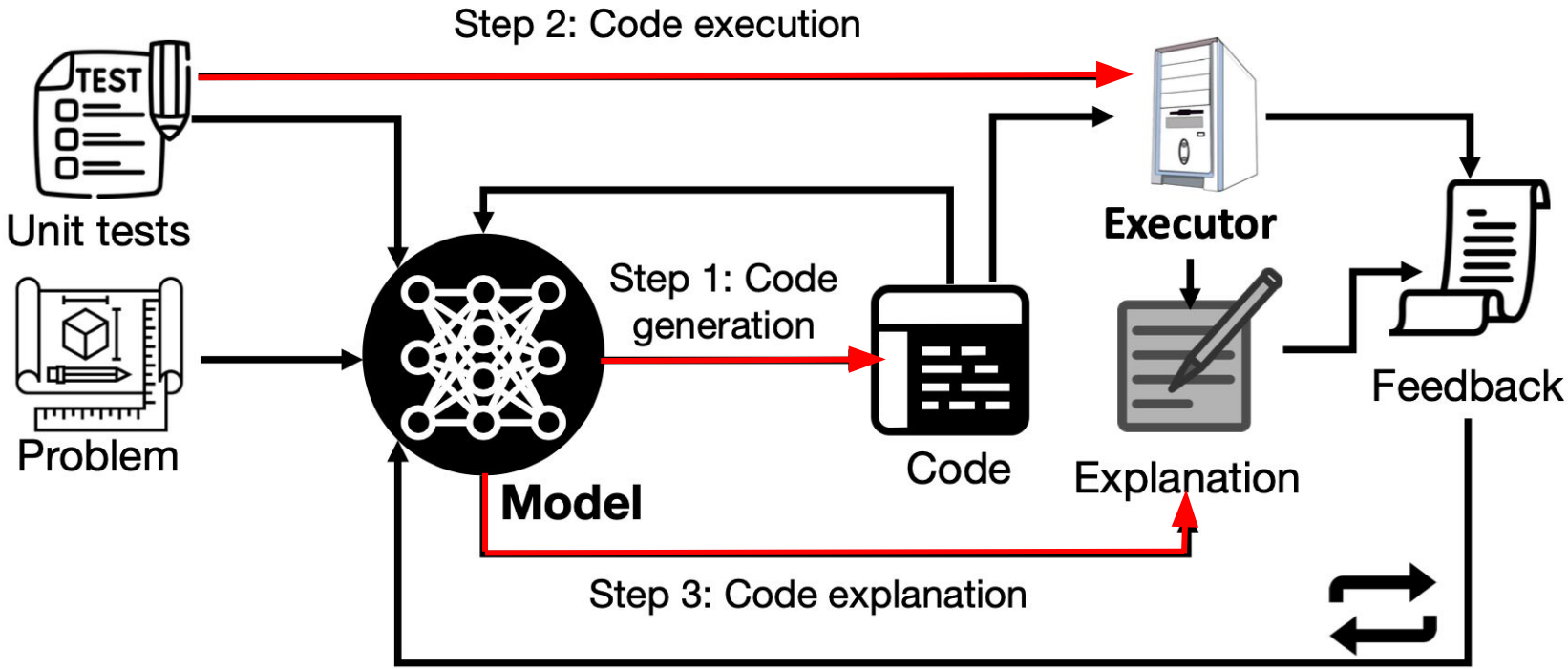
Can we instead teach the LLM to debug its generated code?

Self Debug: Iterative debugging of generated code

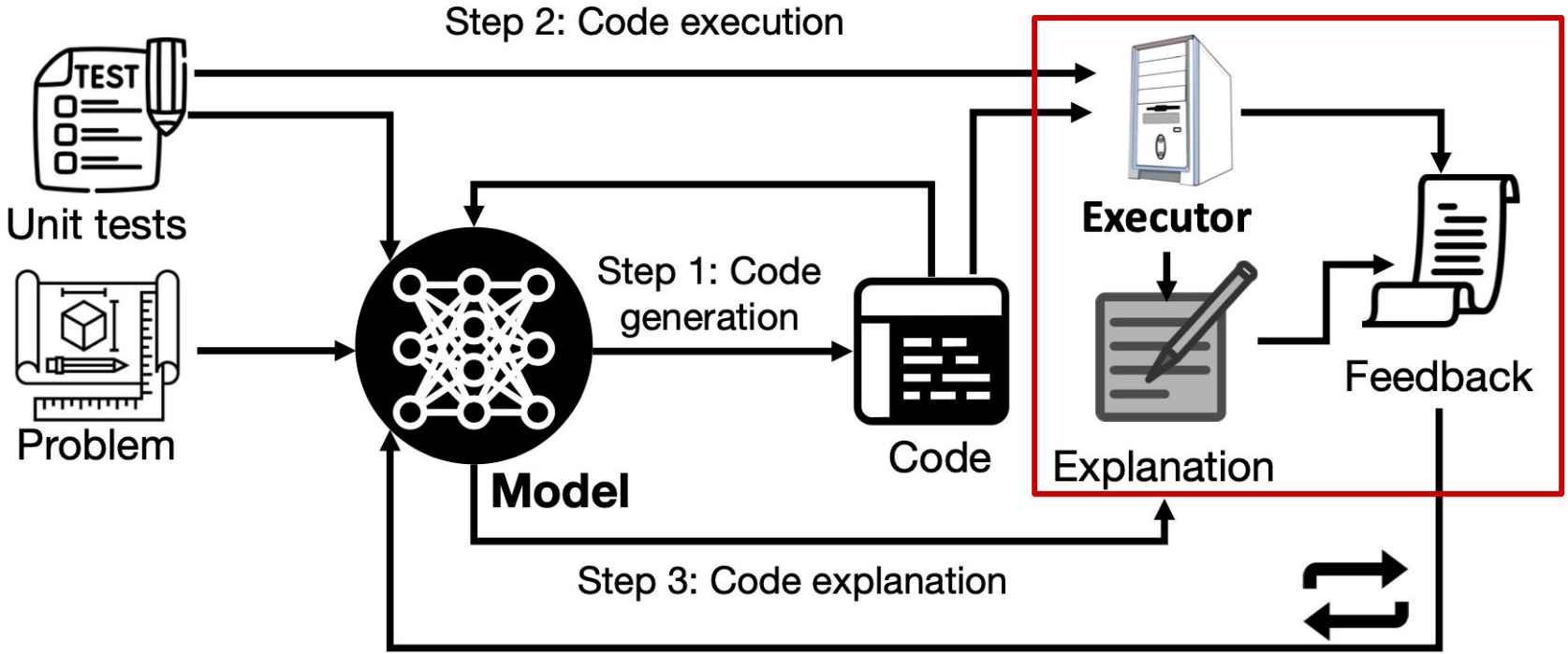


Self Debug,

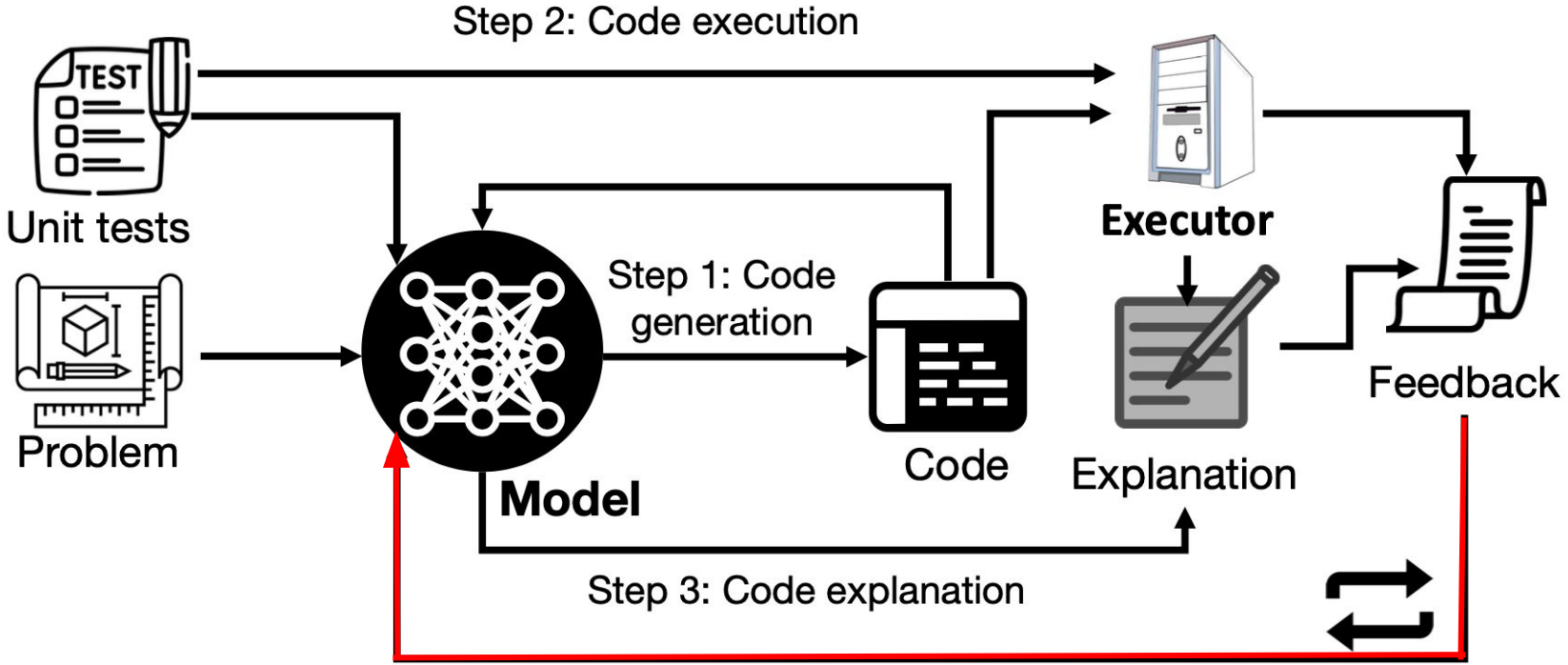
Generate the code and it’s explanation and execute it



The code explanation and execution results constitute the feedback message, based on which the model infers code correctness.



The feedback message is sent back to the model to perform more debugging steps. Feedback can be based on explanation alone (rubber duck debugging).



Self Debug: SQL Example

Step 1: Question explanation

Infer the return type of the question.

[Question]

Which customers have both "On Road" and "Shipped" as order status? List the customer names.

[Question Explanation]

"List the customer names" returns 1 column. The question returns the customer names who have both "On Road" and "Shipped" as order status. So the question returns 1 column.

Self Debug: SQL Example

Step 2: Code explanation

Summarize the return type of the execution.

[SQL]

SELECT ustomers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id
WHERE orders.order_status = "On Road" OR orders.order_status = "Shipped"

Execution:

| George |

[SQL Explanation]

The execution of the SQL query above would return a table with 1 column.
The first column, "customers.customer_name" would contain the customer names.
With "customers JOIN orders", the table would contain the data about customers with orders.
With "WHERE orders.order_status = 'On Road' OR orders.order_status = 'Shipped'", the table filters the records to only include customers who have order status "On Road" or "Shipped".
So the SQL query returns a table with 1 column, the customer names who have the order status "On Road" or "Shipped".

Step 3: Self-Debugging with explanation

[Question]

[Explanation summary]

As in your explanation, the SQL query returns a table with 1 column, the customer names who have the order status "On Road" or "Shipped". The question returns 1 column, the customer names who have both "On Road" and "Shipped" as order status.

[Feedback]

So the SQL prediction above is wrong. Please fix the SQL.

[Revised SQL #1]

SELECT customers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id WHERE orders.order_status = "On Road" AND orders.order_status = "Shipped"

[Explanation summary #1]

As in your explanation, the SQL query returns an empty table, because the "WHERE" clause contains mutually exclusive conditions. The question returns 1 column, the customer names who have order status of both "On Road" and "Shipped".

[Feedback #1]

So the SQL prediction above is wrong. Please fix the SQL.

[Revised SQL #2]

SELECT customers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id WHERE orders.order_status = "On Road" INTERSECT SELECT customers.customer_name FROM customers JOIN orders ON customers.customer_id = orders.customer_id WHERE orders.order_status = "Shipped"

[Explanation summary #2]

As in your explanation, the SQL query returns a table with 1 column, the customer names who have the order status "On Road" and "Shipped". The question returns 1 column, the customer names who have order status of both "On Road" and "Shipped". So the question returns 1 column.

[Feedback #2]

So the SQL prediction above is correct!

Self Debug Evaluation: Codex with prior ranking techniques

(a) Results on the Spider development set.

	Spider (Dev)
w/ training	
T5-3B + N-best Reranking	80.6
LEVER (Ni et al., 2023)	81.9
Prompting only w/o debugging	
Coder-Reviewer	74.5
MBR-Exec	75.2
SELF-DEBUGGING (this work)	
Codex	81.3
+ Expl.	**84.1**

(b) Results on MBPP dataset.

	n samples
Prior work	
MBR-Exec	63.0 ($n = 25$)
Reviewer	66.9 ($n = 25$)
LEVER	68.9 ($n = 100$)
SELF-DEBUGGING (this work)	
Codex	72.2 ($n = 10$)
Simple	73.6
UT	75.2
UT + Expl.	**75.6**

Finding: Self-Debugging improves overall performance on both tasks

Self Debug: Evaluation with different formats

(a) Results on the Spider development set.

Spider	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	81.3	71.1	73.2	64.7
Simple	81.3	**72.2**	73.4	**64.9**
+Expl.	**84.1**	**72.2**	**73.6**	**64.9**

(b) Results on TransCoder.

TransCoder	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	80.4	89.1	77.3	70.0
Simple	89.3	91.6	80.9	72.9
UT	91.6	**92.7**	88.8	76.4
+ Expl.	**92.5**	**92.7**	**90.4**	**76.6**
+ Trace.	87.9	92.3	89.5	73.6

(c) Results on MBPP.

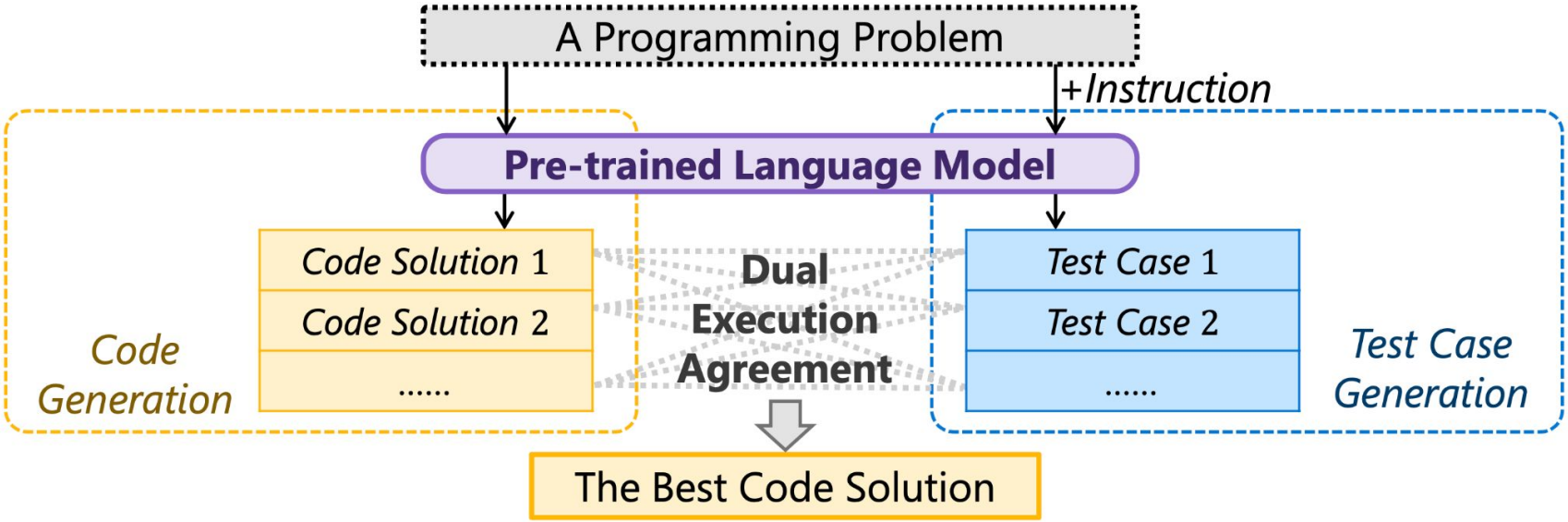
MBPP	Codex	GPT-3.5	GPT-4	StarCoder
Baseline	61.4	67.6	72.8	47.2
Simple	68.2	70.8	78.8	50.6
UT	69.4	72.2	**80.6**	52.2
+ Expl.	69.8	**74.2**	80.4	52.2
+ Trace.	**70.8**	72.8	80.2	**53.2**

Finding: All models benefit from richer feedback for Self-Debugging, especially when the execution information is present in the feedback.

Overview

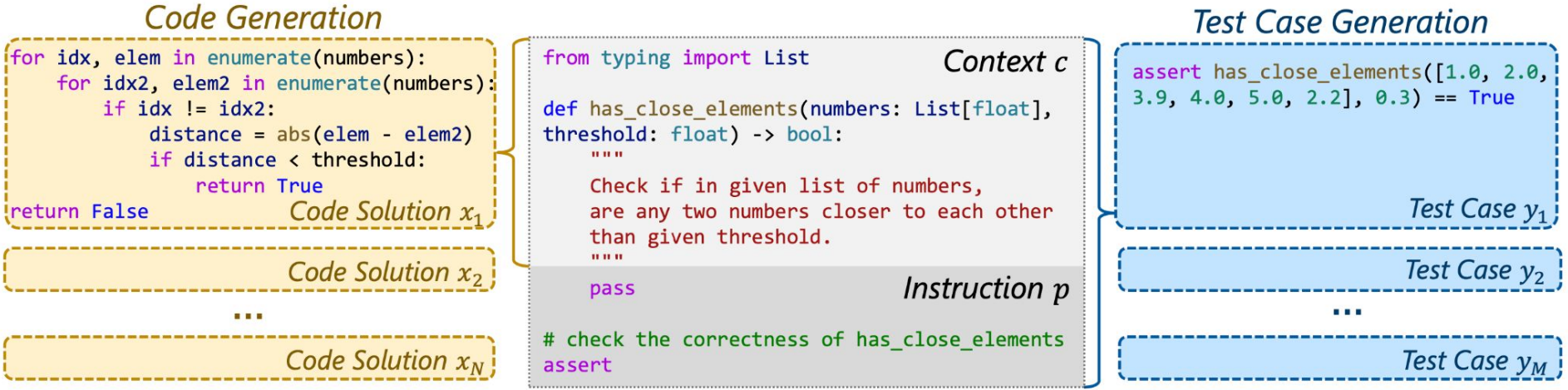
1. Alignment with Natural Language
   - *CodeT5+*
   - *OctoPack*
2. Alignment using Execution Signal
   - *LEVER*
   - *Self Debug*
3. **Alignment with Tests**
   - *CodeT*
   - *RLTF*

CodeT: Code Generation with Generated Tests



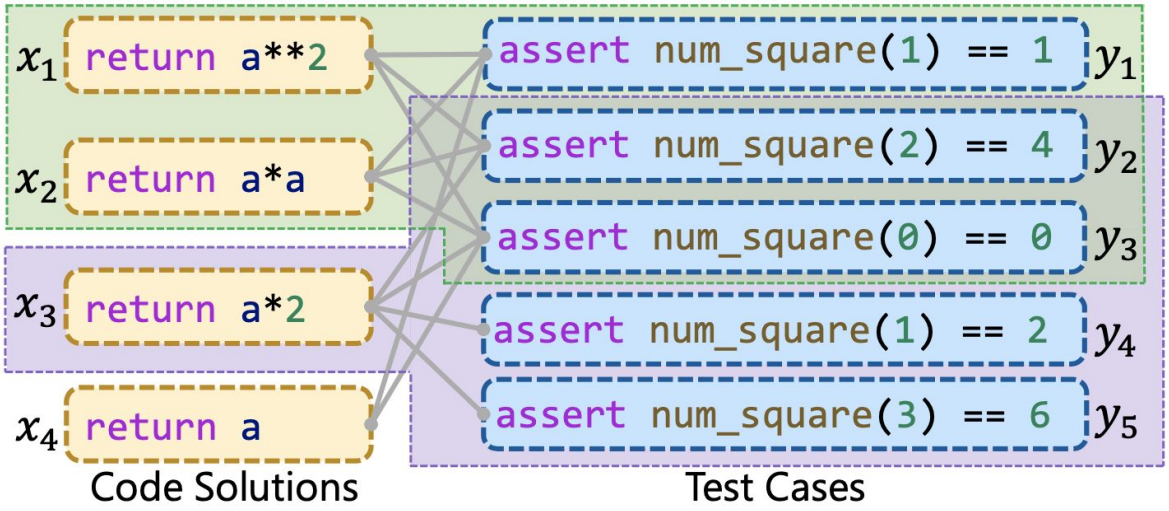
CodeT,

CodeT: Example from HumanEval



Note: Example input-output cases removed from context

CodeT: Dual Execution Agreement



S_x - code that pass same tests

S_y - all tests a code can pass

Consensus set, S

$S = \{(x,y)|x \in S_x, y \in S_y\}$

Score: $f(S) = |S_x||S_y|$

Key Idea: the more pairs that agree with the hypothetical functionality, the more likely this functionality is correct.

Best code solution selected from the consensus set with highest score.

CodeT: Evaluation

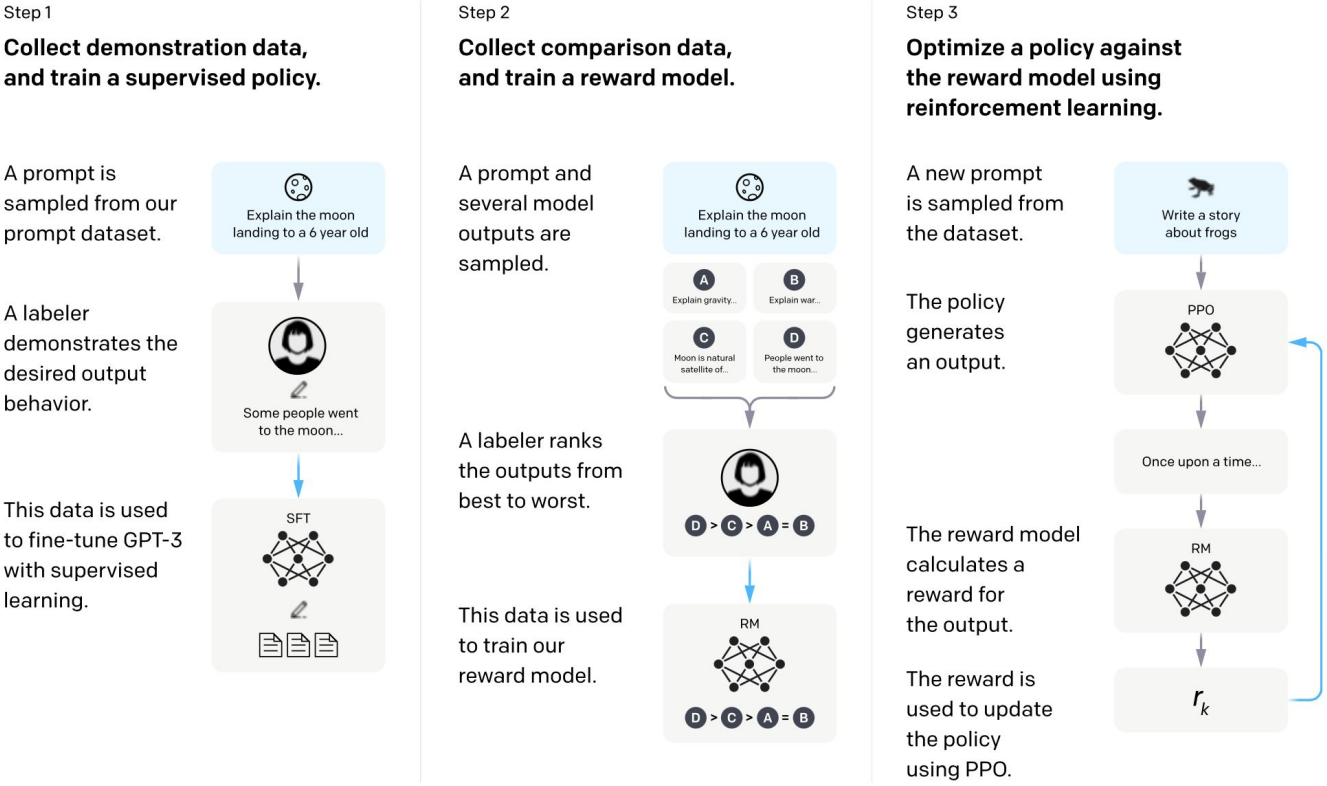
Methods	Baseline			AlphaCode-C			CodeT		
k	1	10	100	1	2	10	1	2	10
HumanEval									
code-cushman-001	33.5	54.3	77.4	39.6	46.4	63.8	44.5 11.0	50.1	65.7 11.4
code-davinci-001	39.0	60.6	84.1	41.6	50.7	75.6	50.2 11.2	58.9	75.8 15.2
code-davinci-002	47.0	74.9	92.1	55.1	64.1	84.4	65.8 18.8	75.1	86.6 11.7
InCoder-6B	16.4 15.2	28.3 27.8	47.5 47.0	17.7	23.8	34.8	20.6 4.2	27.6	37.1 8.8
CodeGen-Mono-16B	29.7 29.3	50.3 49.9	73.7 75.0	27.3	38.5	64.4	36.7 7.0	44.7	59.3 9.0
MBPP									
code-cushman-001	45.9	66.9	79.9	51.5	59.0	73.3	55.4 9.5	61.7	72.7 5.8
code-davinci-001	51.8	72.8	84.1	56.2	64.7	78.8	61.9 10.1	69.1	79.3 6.5
code-davinci-002	58.1	76.7	84.5	62.0	70.7	79.9	67.7 9.6	74.6	81.5 4.8
InCoder-6B	21.3 19.4	46.5	66.2	26.7	35.3	56.2	34.4 13.1	43.9	58.2 11.7
CodeGen-Mono-16B	42.4	65.8	79.1	41.0	55.9	73.6	49.5 7.1	56.6	68.5 2.7

Finding: CodeT consistently improves the performance of all models, including the strongest baselines (which can generate better tests)

Do we have to generate Tests?

No, many projects already have developer written tests in them!

Recall: Reinforcement Learning from Human Feedback

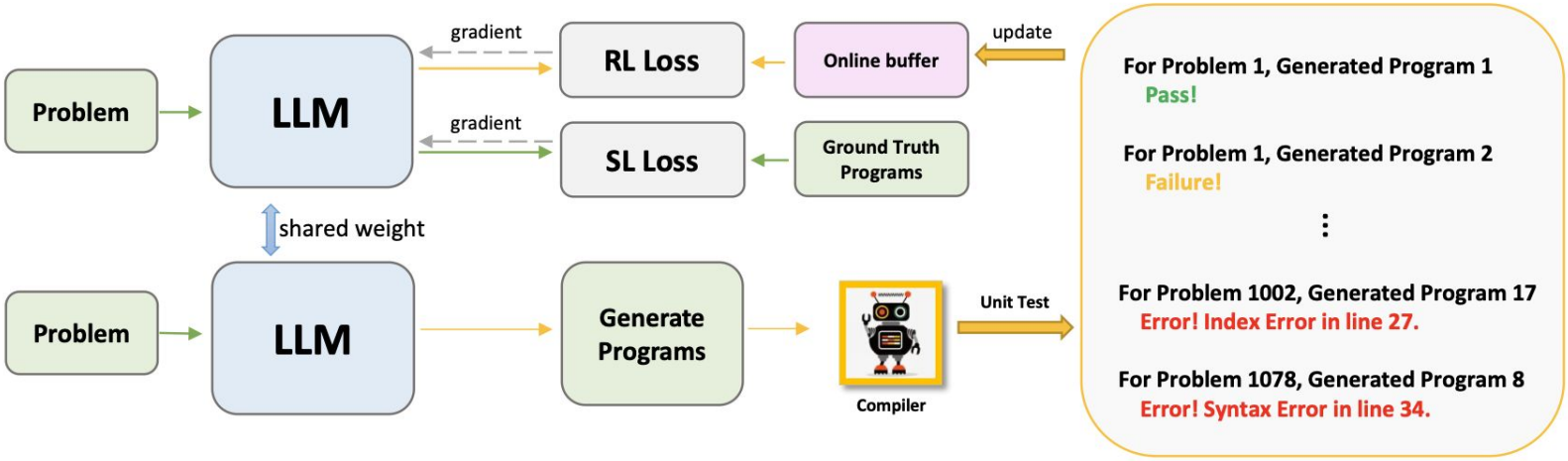


InstructGPT (OpenAI, 2022), https://arxiv.org/pdf/2203.02155.pdf

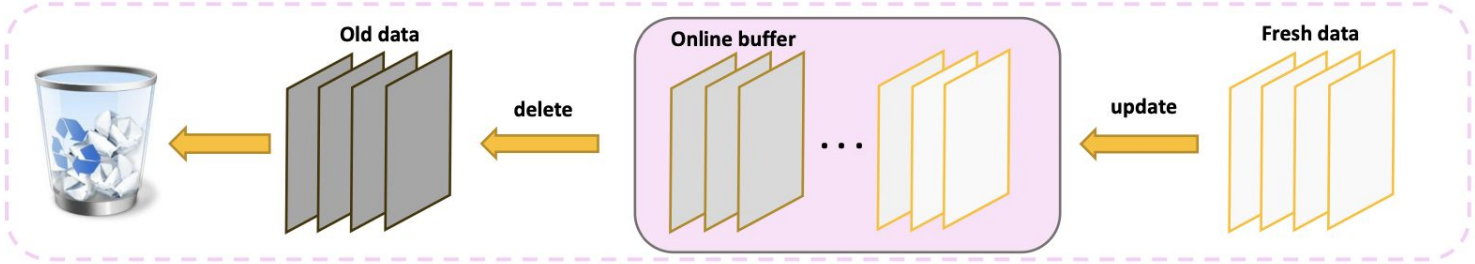
Can the same idea be used to improve Code LLMs, without human feedback and using Tests instead?

Yes, with Reinforcement Learning from Unit Test Feedback (RLTF)

RLTF: Overview



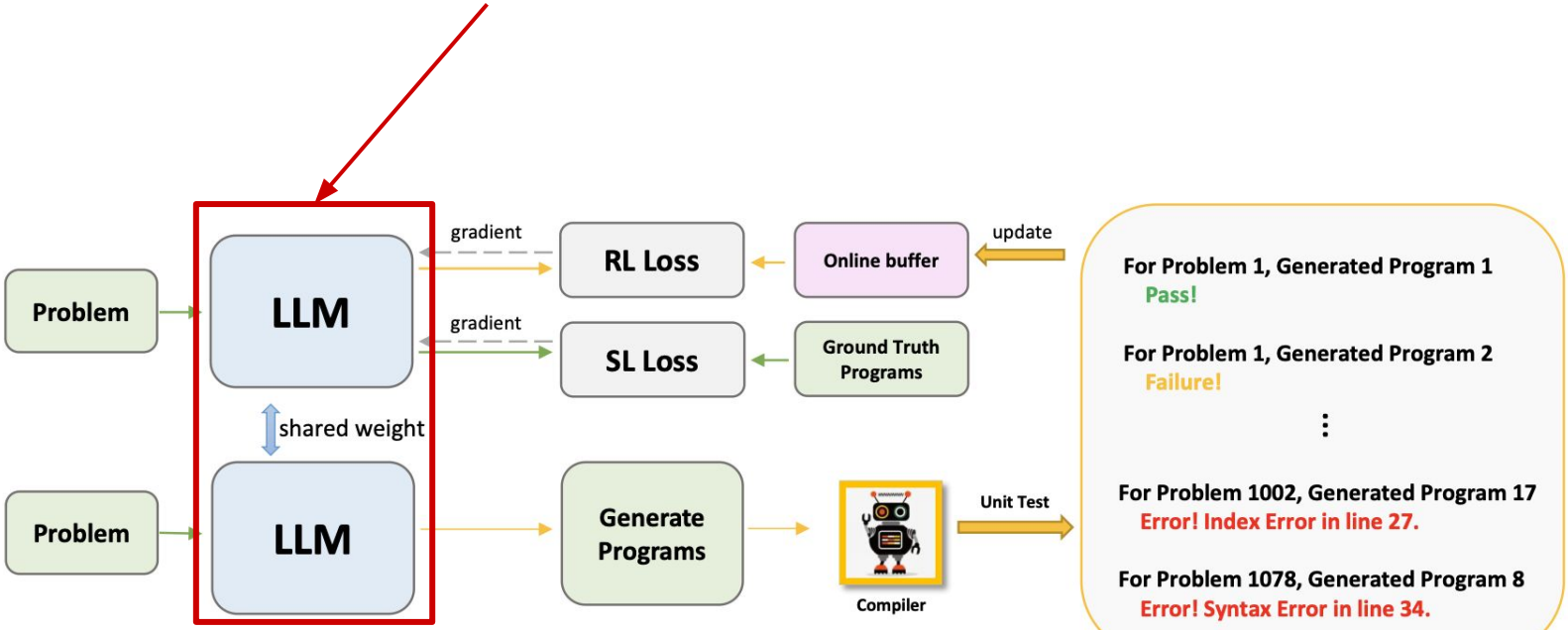
(a) Online reinforcement learning framework.



(b) Online buffer workflow.

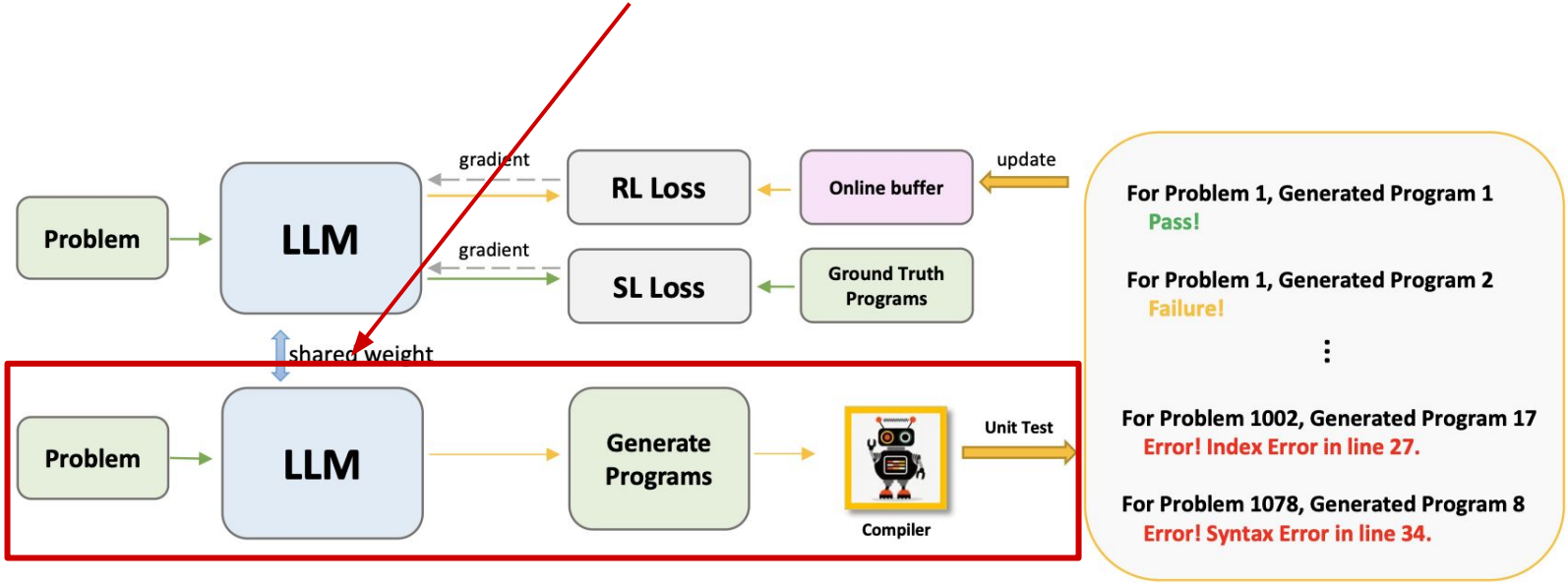
RLTF,

RLTF: Two LLMs with shared weights



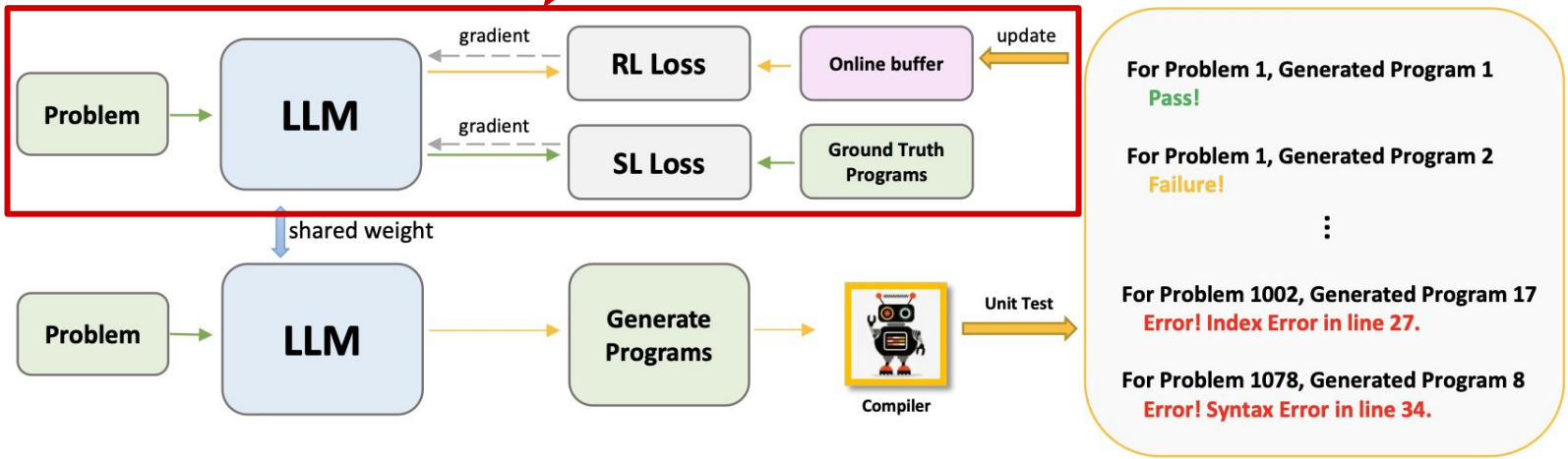
(a) Online reinforcement learning framework.

RLTF: First LLM generates the target program that is passed to the compiler to produce a training data pair (stored in online buffer)



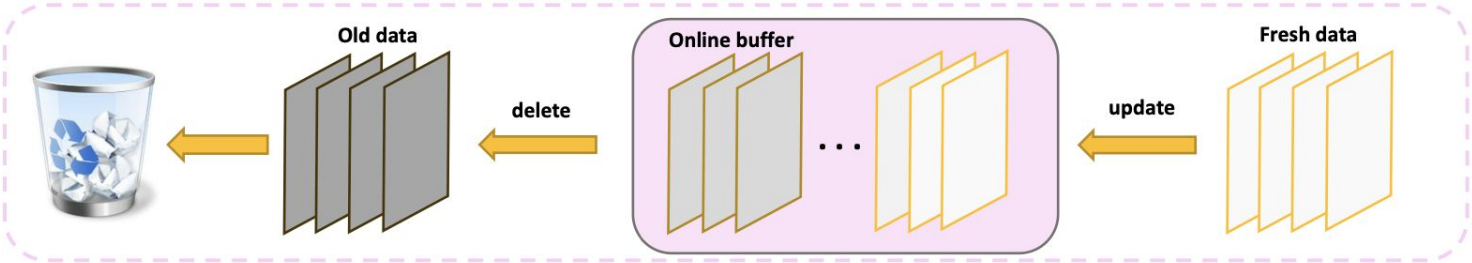
(a) Online reinforcement learning framework.

RLTF: Second LLM uses the ground truth data (supervised learning) and the online buffer data (RL) to compute loss and update model weights through gradient feedback.



(a) Online reinforcement learning framework.

RLTF: Online buffers are utilized to retain the newly generated data for RL training. As fresh data is received, old data is deleted.



(b) Online buffer workflow.

RLTF: Some notes

- Online framework that uses feedback from unit tests in real time
- Uses fine-grained feedback for programs with errors, which categorizes errors and penalizes the specific erroneous parts of the code
- Uses adaptive feedback for programs that do not pass all test cases, which assigns varying rewards based on the ratio of passed test cases

Data Benchmarks

APPS (Automated Programming Progress Standard) - Program Synthesis

- 10K coding problems with varying difficulty levels
  - Introductory (total = 3639, train = 2639, test = 1000)
  - Interview (total = 5000, train = 2000, test = 3000)
  - Competition (total = 1361, train = 361, test = 1000)
- On average, each problem has
  - 23.2 accurate Python programs
  - 21.2 unit tests

MBBP (Mostly basic programming problems)

- 974 (374/90/500) problems with 1 correct solution and 3 unit tests

RLTF: Quantitative Evaluation on APPS

Method	Size	CS	pass@1				pass@5				pass@1000			
			Intro	Inter	Comp	all	Intro	Inter	Comp	all	Intro	Inter	Comp	all
Codex	12B	w/o	4.14	0.14	0.02	0.92	9.65	0.51	0.09	2.25	25.02	3.70	3.23	7.87
AlphaCode	1B	w/o	-	-	-	-	-	-	-	-	17.67	5.24	7.06	8.09
GPT3	175B	w/o	0.20	0.03	0.00	0.06	-	-	-	-	-	-	-	-
GPT2	0.1B	w/o	1.00	0.33	0.00	0.40	2.70	0.73	0.00	1.02	-	-	-	-
GPT2	1.5B	w/o	1.30	0.70	0.00	0.68	3.60	1.03	0.00	1.58	27.90	9.83	11.40	13.76
GPT-Neo	2.7B	w/o	3.90	0.57	0.00	1.12	5.50	0.80	0.00	1.58	27.90	9.83	11.40	13.76
CodeRL	770M	w/o	4.00	0.78	0.15	1.30	9.83	2.03	0.69	3.32	35.30	13.33	13.60	17.78
PPOCoder	770M	w/o	4.06	0.79	0.15	1.32	9.97	2.06	0.70	3.37	35.42	13.37	13.65	17.84
RLTF	770M	w/o	**4.16**	**0.97**	**0.20**	**1.45**	**10.12**	**2.65**	**0.82**	**3.78**	**38.30**	**15.13**	**15.90**	**19.92**
CodeRL	770M	w	6.77	1.80	0.69	2.57	15.27	4.48	2.36	6.21	38.10	14.33	15.70	19.36
RLTF	770M	w	**8.40**	**2.28**	**1.10**	**3.27**	**18.60**	**5.57**	**3.70**	**7.80**	**39.70**	**15.03**	**16.80**	**20.32**

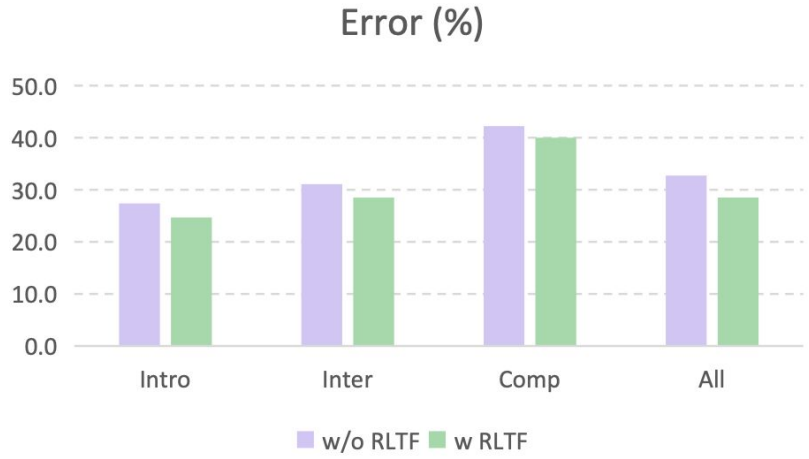
Finding: CodeT5-RLTF outperforms other base models, and in addition to other CodeT5 based approaches like CodeRL and PPOCoder.

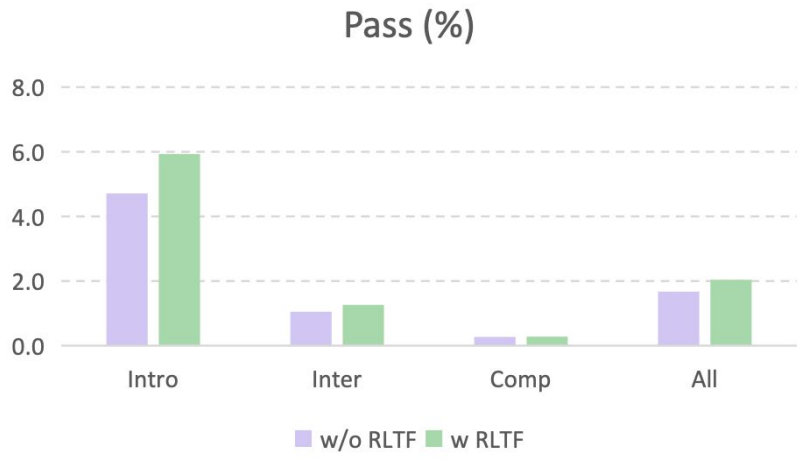
RLTF: Impact of Language Model

Language Model	RLTF	pass@1	pass@5	pass@10	pass@100	pass@1000
CodeT5 770M	w/o	1.30	3.39	4.68	10.51	17.80
CodeT5 770M	w	**1.45**	**3.78**	**5.21**	**11.23**	**19.92**
CodeGen2.7B	w/o	1.64	4.23	5.79	12.48	21.44
CodeGen2.7B	w	**2.04**	**5.04**	**6.80**	**14.17**	**23.96**

Finding: Larger the base model, greater the performance boost provided by RLTF

Qualitative Analysis by Unit Test Outcomes on CodeGen





Finding: RLTF reduces the proportion of programs resulting in errors and increases the proportion of programs that pass.

RLTF: Evaluation on MBBP (zero-shot)

Method	Size	state	pass@80
GPT	224M	fine-tuned	7.2
GPT	442M	fine-tuned	12.6
GPT	1B	fine-tuned	22.4
GPT	4B	fine-tuned	33.0
GPT	8B	fine-tuned	40.6
GPT	68B	fine-tuned	53.6
GPT	137B	fine-tuned	61.4
CodeT5 + CodeRL	770M	zero-shot	68.1
CodeT5 + PPOCoder	770M	zero-shot	68.2
CodeT5 + RLTF	770M	zero-shot	**71.3**

Prompt: problem description + “Your code should satisfy these tests:” + 3 assert statements.

Finding: CodeT5 + RLTF(trained on APPS) outperformed all sizes of GPT models

GPT-4 vs  LM

Compute and data resources

Open Source

Proprietary data

Proof of concept

Nikitha Rao | nikitharao@cmu.edu

Examples of first test generated

GPT-4

```java
@Before
public void setUp() {
    bank = new Bank();
}

@Test
public void testAddCustomerAndCustomerSummary()
    {
    Customer customer1 = new Customer("John
        Doe");
    bank.addCustomer(customer1);

    String expectedSummary = "Customer
        Summary\n - John Doe (0 accounts)";
    String actualSummary =
        bank.customerSummary();

    assertEquals(expectedSummary,
        actualSummary);
}
```

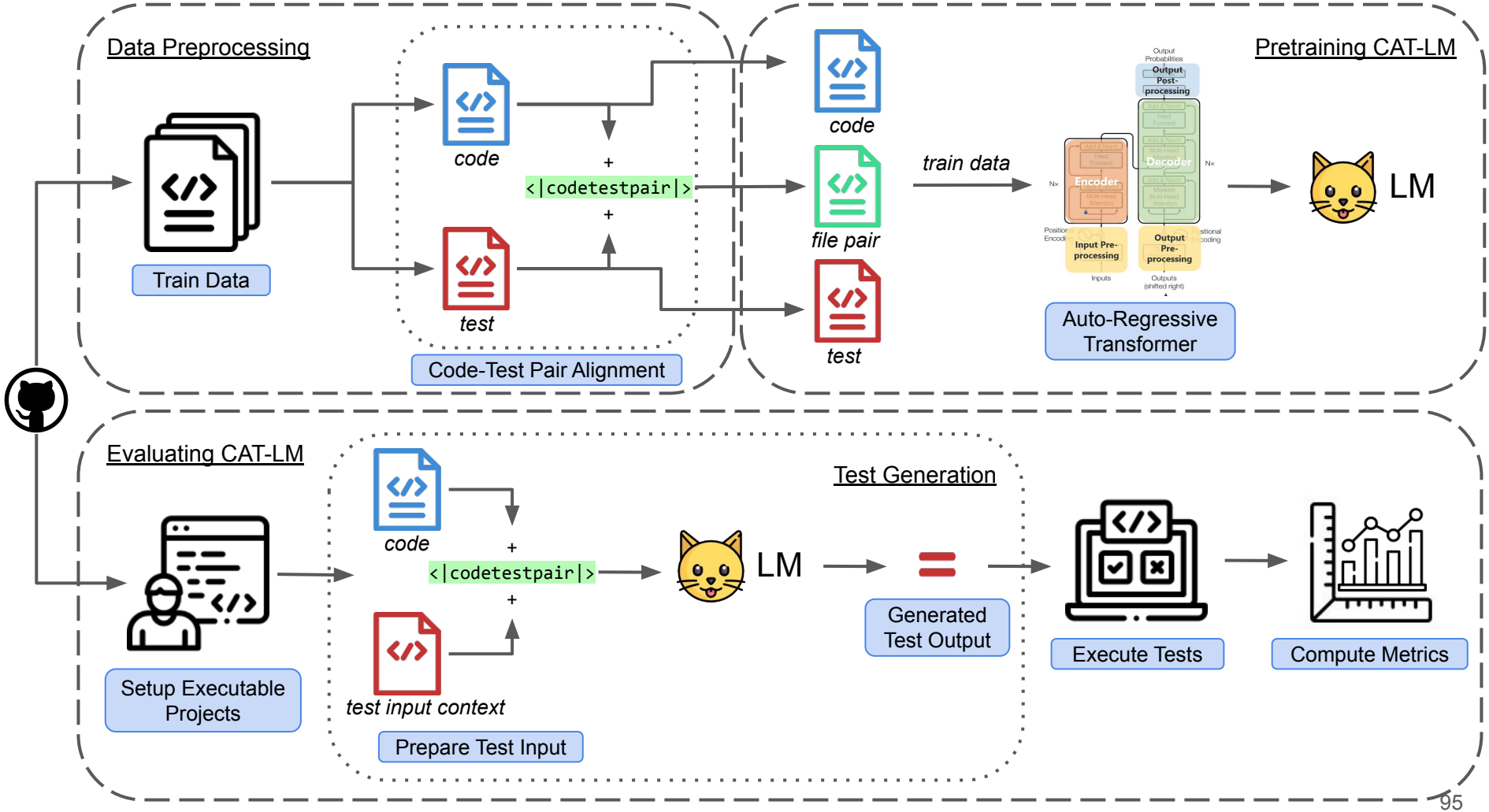
EvoSuite

```java
@Test(timeout = 4000)
public void test0() throws Throwable  {
    Bank bank0 = new Bank();
    Customer customer0 = new Customer("v\"PD");
    bank0.addCustomer(customer0);
    Account account0 = new Account(0);
    account0.deposit(148.3628547);
    customer0.openAccount(account0);
    double double0 = bank0.totalInterestPaid();
    assertEquals(0.14836285470000002, double0,
        0.01);
}
```

Takeaway

GPT-4: high quality tests with readable asserts

Evosuite: meaningless tests, poor naming conventions & spurious exception handling


Data Preprocessing
Train Data
code
<|codetestpair|>
test
Code-Test Pair Alignment
code
file pair
test
train data
Pretraining CAT-LM
Output Probabilities
Output Post-processing
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Encoder
Decoder
Nx
Input Pre-processing
Output Pre-processing
Inputs
Outputs (shifted right)
Auto-Regressive Transformer
LM
Evaluating CAT-LM
Setup Executable Projects
code
<|codetestpair|>
test input context
Prepare Test Input
Test Generation
LM
Generated Test Output
Execute Tests
Compute Metrics