AI for SE

In the era of ChatGPT

Michele Tufano
Sr. Research Scientist

Data+AI

About me

What I work on

- AI Models Assessment for Software Engineering Tasks
- Driving AI for Code Model Improvements

Team Collaborations

GitHub Copilot

Data+AI

4:20pm 100 %

Gabriele Bavota

Hey, can you make a 90min presentation for the Summer School?

4:20pm

100 %

Gabriele Bavota

Hey, can you make a 90min presentation for the Summer School?

Sure!

Data+AI

4:20pm

100 %

Gabriele Bavota

Hey, can you make a 90min presentation for the Summer School?

Sure!

Discovers lack of experience in 90min presentations

Data+AI

4:20pm

100 %

Gabriele Bavota

Hey, can you make a 90min presentation for the Summer School?

Sure!

Discovers lack of experience in 90min presentations

Recognizes potential to delegate work to students

Data+AI

4:20pm 100 %

Gabriele Bavota

Hey, can you make a 90min presentation for the Summer School?

Sure!

Discovers lack of experience in 90min presentations

Recognizes potential to delegate work to students

It will be great!

iMessage

Data+AI

AI for SE
~
(mostly LLMs for Code)

AI for SE
~
(mostly LLMs for Code)

Agenda

Light Finetuning Strategies

Reinforcement Learning from Human Feedback

Prompting Strategies

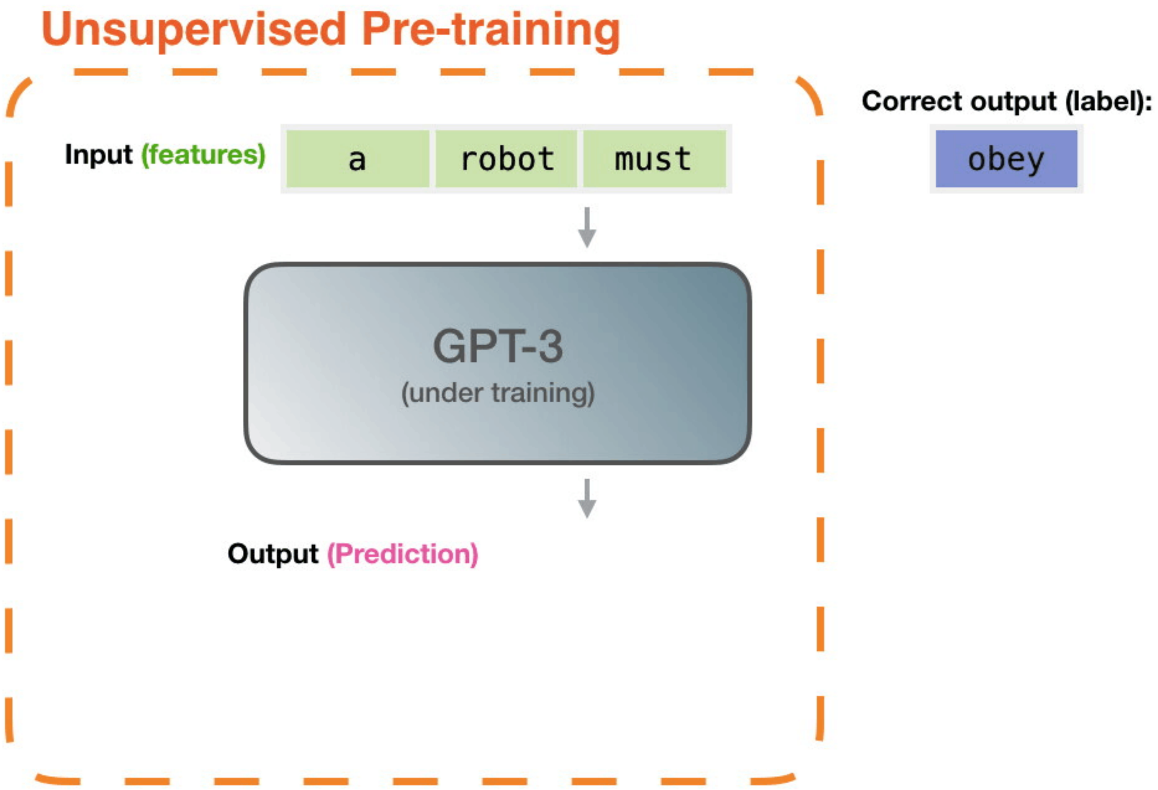
Evaluation of LLMs for Code

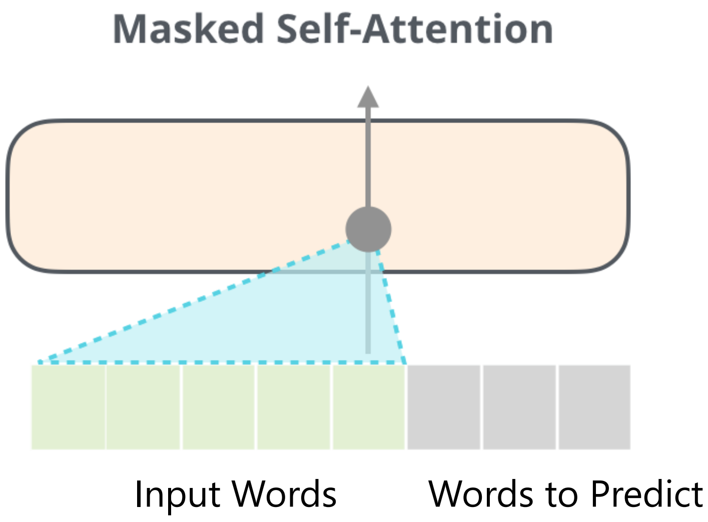
Your inputs and ideas

Data+AI

Large Language Models

OpenAI GPT-3 style LLMs
Decoder-only Transformer Models trained to predict the next word
Unsupervised Pre-Training on large amount of text using Masked Self-Attention





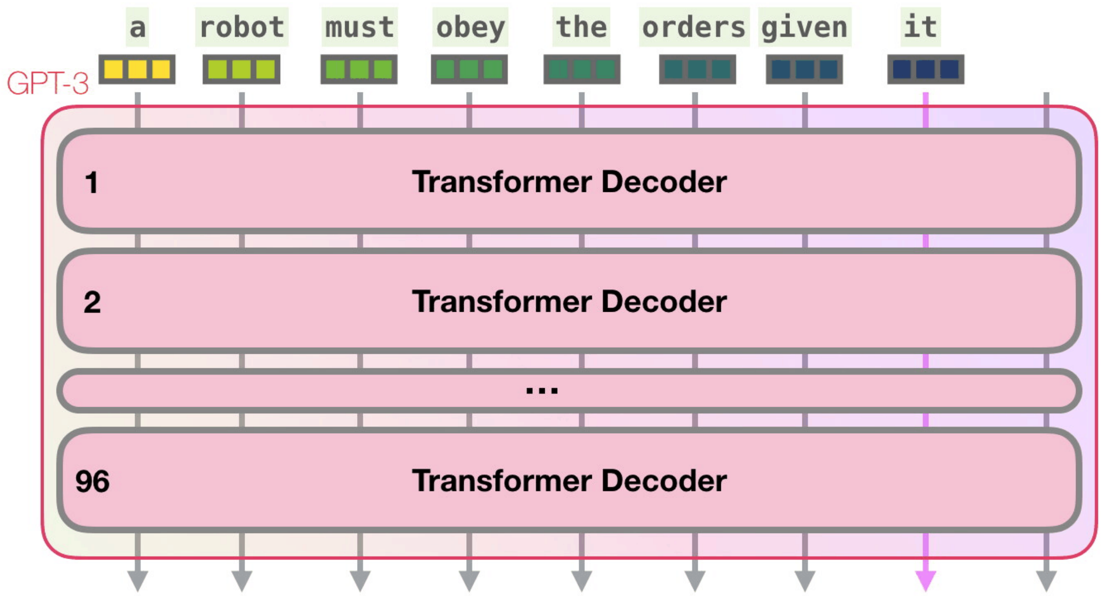
Images from: https://jalammar.github.io/

Large Language Models

OpenAI GPT-3 style LLMs

Codex (code-cushman-001)
- 12 billion parameters
- Pretrained specifically on code
- Efficient to finetune on specific tasks

Davinci (text-davinci-003)
- 175 billion parameters
- Similar to ChatGPT
- Expensive to finetune



Data+AI

Images from: https://jalammar.github.io/

Efficient finetuning for LLMs

LoRA: Low-Rank Adaption of LLMs

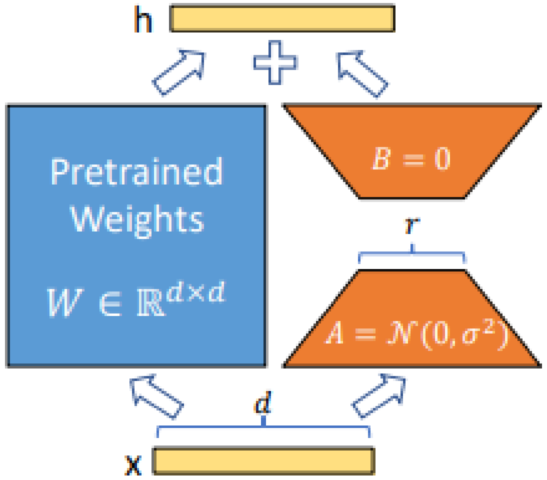


Figure 1: Our reparametrization. We only train A and B.

LoRA aims to learn the change factor ΔW.

Assuming the pre-training matrix is denoted as $W_0 \in R^{d*k}$,
the update to the pre-trained matrix can be represented as follows :

$$W_0 + \Delta W = W_0 + BA, B \in R^{d*r}, A \in R^{r*k}$$

The rank $r \ll min(d, k)$

Training
Both W_0 and ΔW are multiplied by the same input x, resulting in the following:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

Inference
Only necessary to add the change factor back into the original model:

$$W = W_0 + BA$$

How can we train many personalized models?

Exploring and Evaluating Personalized Models for Code Generation

Andrei Zlotchevski
McGill University
Montreal, Quebec, Canada
andrei.zlotchevski@mail.mcgill.ca

Dawn Drain
Anthropic
San Francisco, CA, USA
dawn@anthropic.com

Alexey Svyatkovskiy
Microsoft
Redmond, WA, USA
alsvyatk@microsoft.com

Colin B. Clement
Microsoft
Redmond, WA, USA
coclemen@microsoft.com

Neel Sundaresan
Microsoft
Redmond, WA, USA
neels@microsoft.com

Michele Tufano
Microsoft
Redmond, WA, USA
mitufano@microsoft.com

ABSTRACT

Large Transformer models achieved the state-of-the-art status for Natural Language Understanding tasks and are increasingly becoming the baseline model architecture for modeling source code. Transformers are usually pre-trained on large unsupervised corpora, learning token representations and transformations relevant to modeling generally available text, and are then fine-tuned on a particular downstream task of interest. While fine-tuning is a tried-and-true method for adapting a model to a new domain – for example, question-answering on a given topic – generalization remains an on-going challenge. In this paper, we explore and evaluate transformer model fine-tuning for personalization. In the context of generating unit tests for Java methods, we evaluate learning to personalize to a specific software project using several personalization techniques. We consider three key approaches: (i) *custom* fine-tuning, which allows all the model parameters to be tuned; (ii) *lightweight* fine-tuning, which freezes most of the model's parameters, allowing tuning of the token embeddings and softmax layer only or the final layer alone; (iii) *prefix* tuning, which keeps model parameters frozen, but optimizes a small project-specific prefix vector. Each of these techniques offers a trade-off in total compute cost and predictive performance, which we evaluate by code and task-specific metrics, training time, and total computational operations. We compare these fine-tuning strategies for code generation and discuss the potential generalization and cost benefits of each in various deployment scenarios.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **Information systems** → *Recommender systems.*

KEYWORDS

Personalized Models, Code Generation

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558959

ACM Reference Format:
Andrei Zlotchevski, Dawn Drain, Alexey Svyatkovskiy, Colin B. Clement, Neel Sundaresan, and Michele Tufano. 2022. Exploring and Evaluating Personalized Models for Code Generation. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3540250.3558959

1 INTRODUCTION

It is well-known that even the best models can fail to generalize properly to new domains, and even to new users of said models. For example, a model trained to answer questions in general may not answer StackOverflow questions as well as the questions in the training domain, or a software developer in an Enterprise environment with private code may have libraries and attribute name which differ from public source code used to train a code synthesis model.

The current dominant paradigm in Natural Language Processing (NLP) modeling is to pre-train a large transformer model [30] on a large corpus and then fine-tune it on a particular task of interest. For example, a question-answering (Q&A) model is generally first pre-trained on a large corpus of textual data for the specific language (*e.g.*, Wikipedia, and news articles in English), then fine-tuned on a task-specific dataset of paired questions and corresponding answers. The pre-training process aims at learning semantic vector representation of the language and words, while the fine-tuning process specializes the model for a specific domain.

Transformer models are also increasingly the baseline architecture used for code generation tasks, such as writing methods from natural language description [2, 5, 7], or generating test cases from the focal method under test [29]. Similarly for NLP tasks these models are pre-trained on a large corpus of natural text and publicly available source code and then fine-tuned on a specific code-related task. Further, these models also may not generalize to new domains of interest, and can benefit from task or even user-specific fine-tuning, here called customization or personalization. Customization is particularly relevant for code generation models since it provides several benefits:

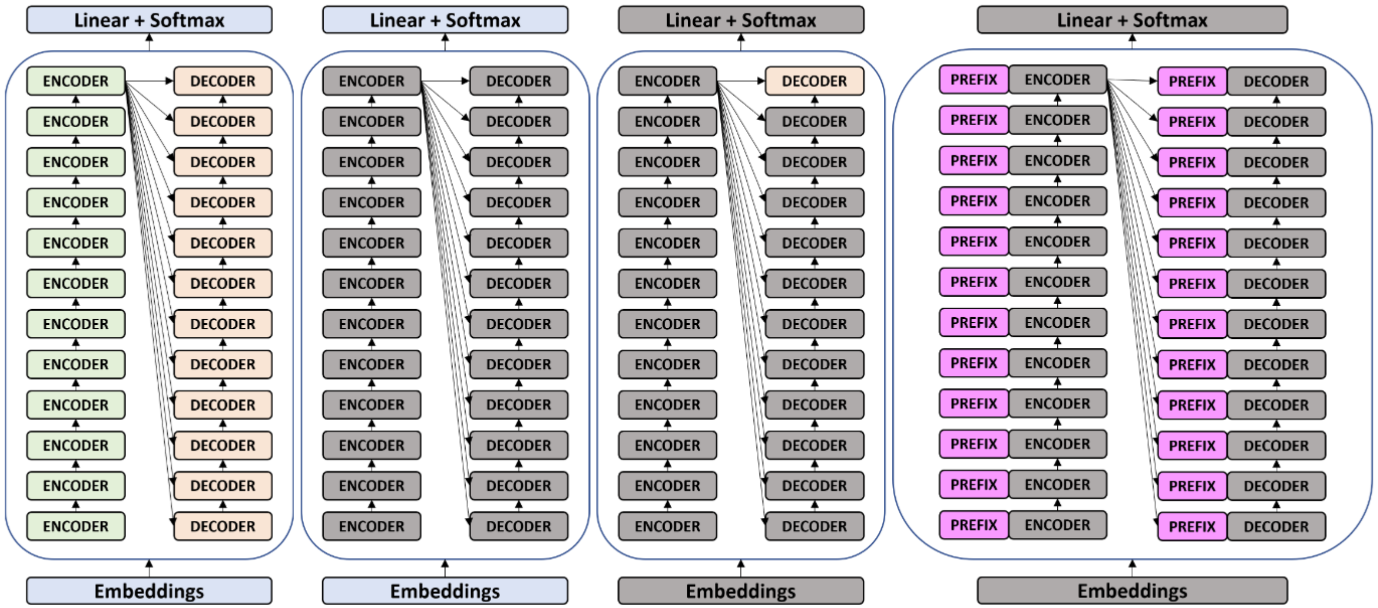
- allows fine-tuning on source code data that may not be available when training a base model (*e.g.*, private repositories or internal codebases), enabling improved overall performances on codebases with proprietary dependencies and code styles;

Data+AI

Freeze part of the model 😩

Let other parts change

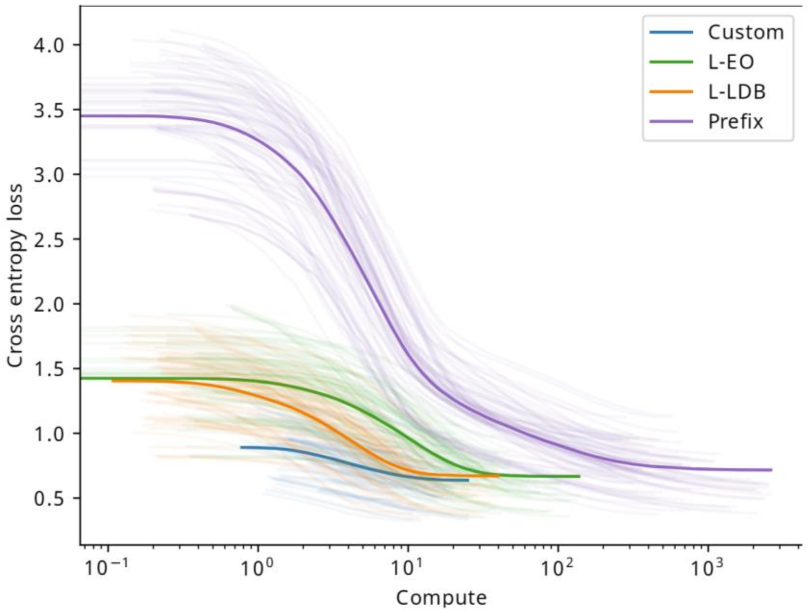
Finetune a prefix



(a) Custom (b) L-EO (c) L-LDB (d) Prefix

Trade-offs

- Compute
- Finetuning Time
- Space and memory



Reinforcement Learning from Human Feedback

Data+AI

arXiv:2009.01325v3 [cs.CL] 15 Feb 2022

Learning to summarize from human feedback

Nisan Stiennon* Long Ouyang* Jeff Wu* Daniel M. Ziegler* Ryan Lowe*

Chelsea Voss* Alec Radford Dario Amodei Paul Christiano*

OpenAI

Abstract

As language models become more powerful, training and evaluation are increasingly bottlenecked by the data and metrics used for a particular task. For example, summarization models are often trained to predict human reference summaries and evaluated using ROUGE, but both of these metrics are rough proxies for what we really care about—summary quality. In this work, we show that it is possible to significantly improve summary quality by training a model to optimize for human preferences. We collect a large, high-quality dataset of human comparisons between summaries, train a model to predict the human-preferred summary, and use that model as a reward function to fine-tune a summarization policy using reinforcement learning. We apply our method to a version of the TL;DR dataset of Reddit posts [63] and find that our models significantly outperform both human reference summaries and much larger models fine-tuned with supervised learning alone. Our models also transfer to CNN/DM news articles [22], producing summaries nearly as good as the human reference without any news-specific fine-tuning.[2] We conduct extensive analyses to understand our human feedback dataset and fine-tuned models.[3] We establish that our reward model generalizes to new datasets, and that optimizing our reward model results in better summaries than optimizing ROUGE according to humans. We hope the evidence from our paper motivates machine learning researchers to pay closer attention to how their training loss affects the model behavior they actually want.

1 Introduction

Large-scale language model pretraining has become increasingly prevalent for achieving high performance on a variety of natural language processing (NLP) tasks. When applying these models to a specific task, they are usually fine-tuned using supervised learning, often to maximize the log probability of a set of human demonstrations.

While this strategy has led to markedly improved performance, there is still a misalignment between this fine-tuning objective—maximizing the likelihood of human-written text—and what we care about—generating high-quality outputs as determined by humans. This misalignment has several causes: the maximum likelihood objective has no distinction between important errors (e.g. making up facts [41]) and unimportant errors (e.g. selecting the precise word from a set of synonyms); models

*This was a joint project of the OpenAI Reflection team. Author order was randomized amongst {LO, JW, DZ, NS}; CV and RL were full-time contributors for most of the duration. PC is the team lead.

[2]Samples from all of our models can be viewed on our website.

[3]We provide inference code for our 1.3B models and baselines, as well as a model card and our human feedback dataset with over 64k summary comparisons, here.

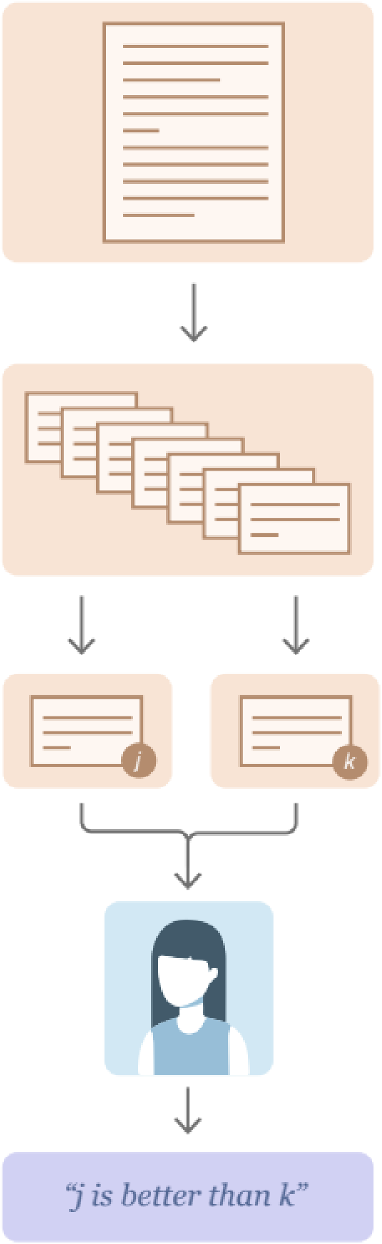
❶ Collect human feedback

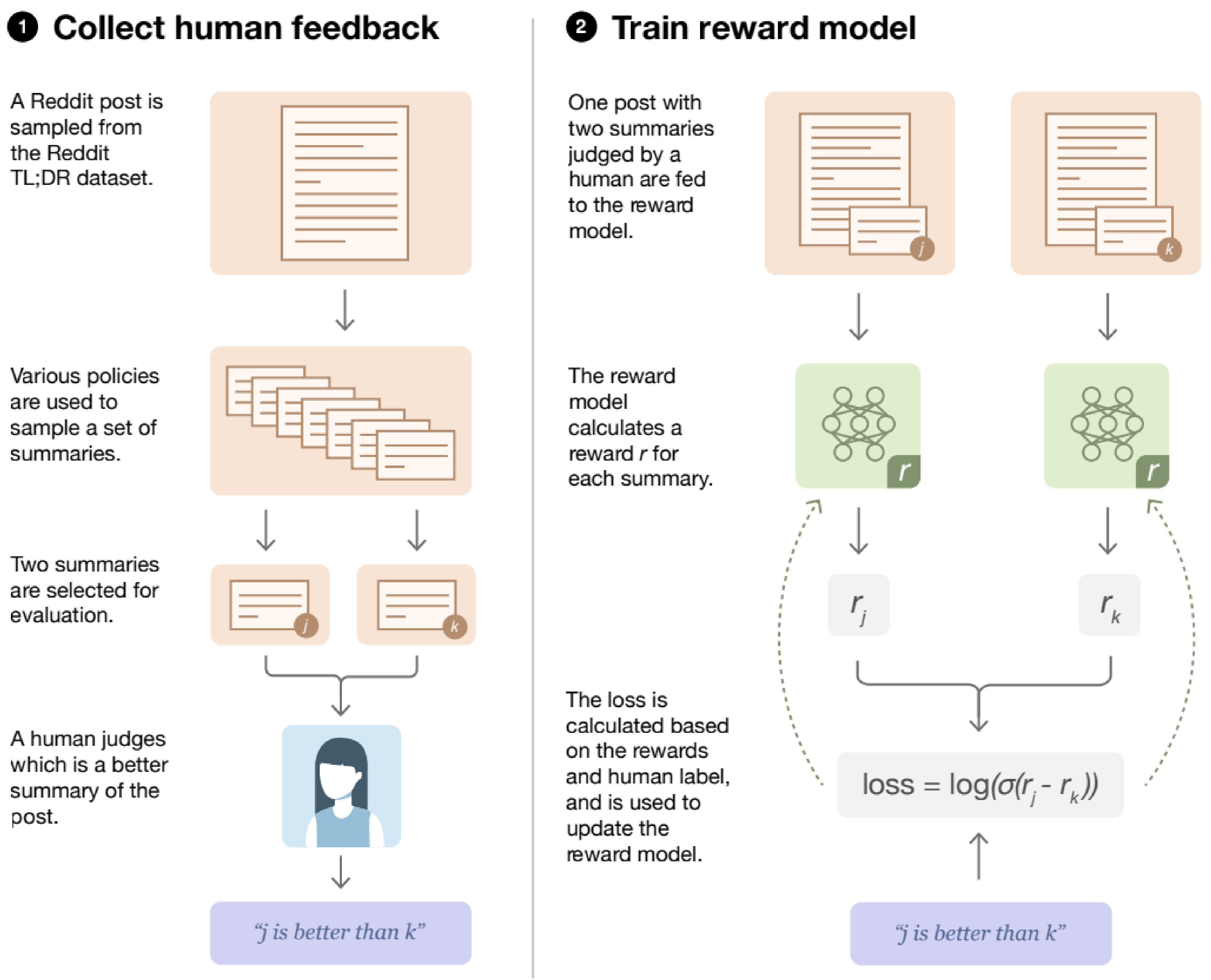
A Reddit post is sampled from the Reddit TL;DR dataset.

Various policies are used to sample a set of summaries.

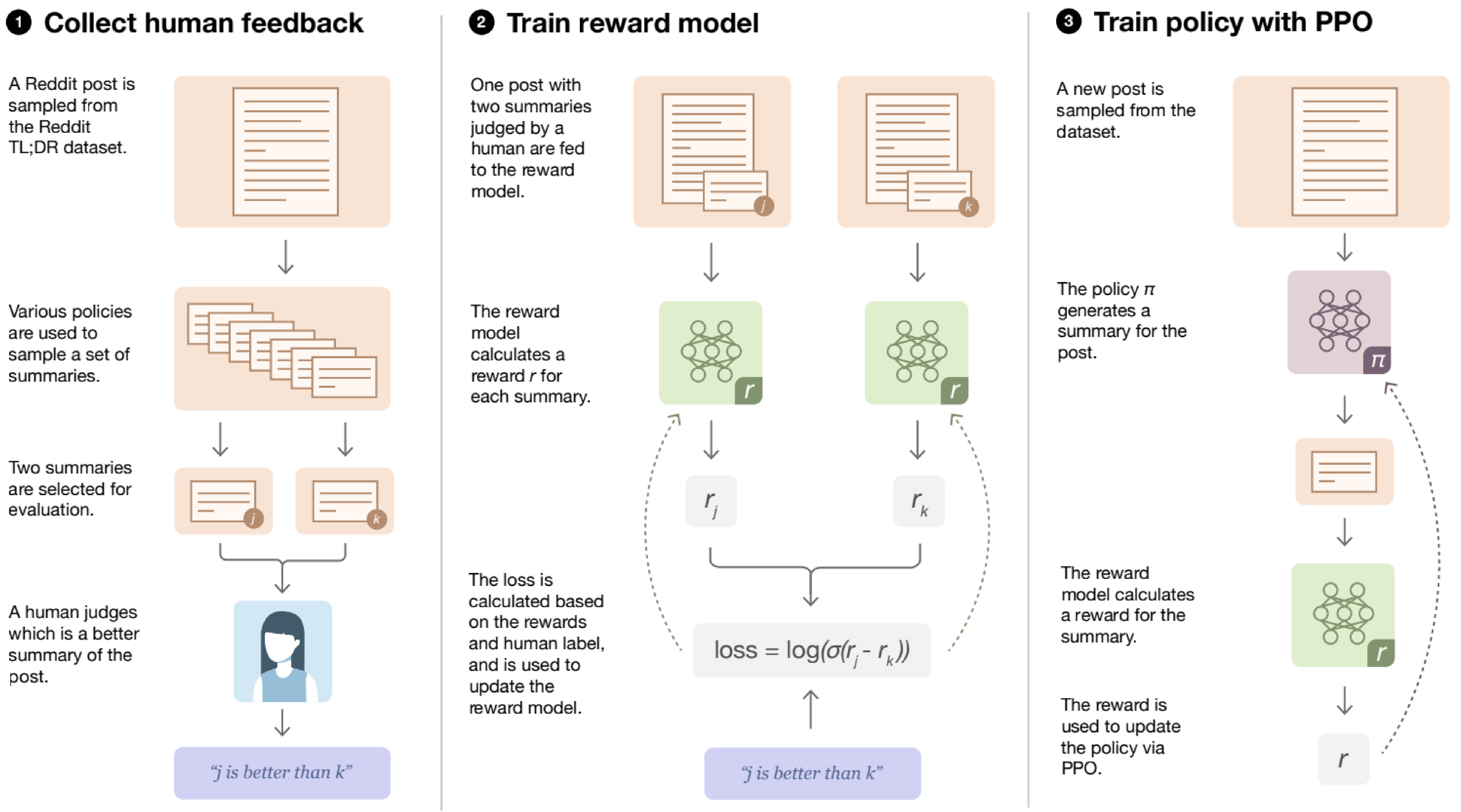
Two summaries are selected for evaluation.

A human judges which is a better summary of the post.



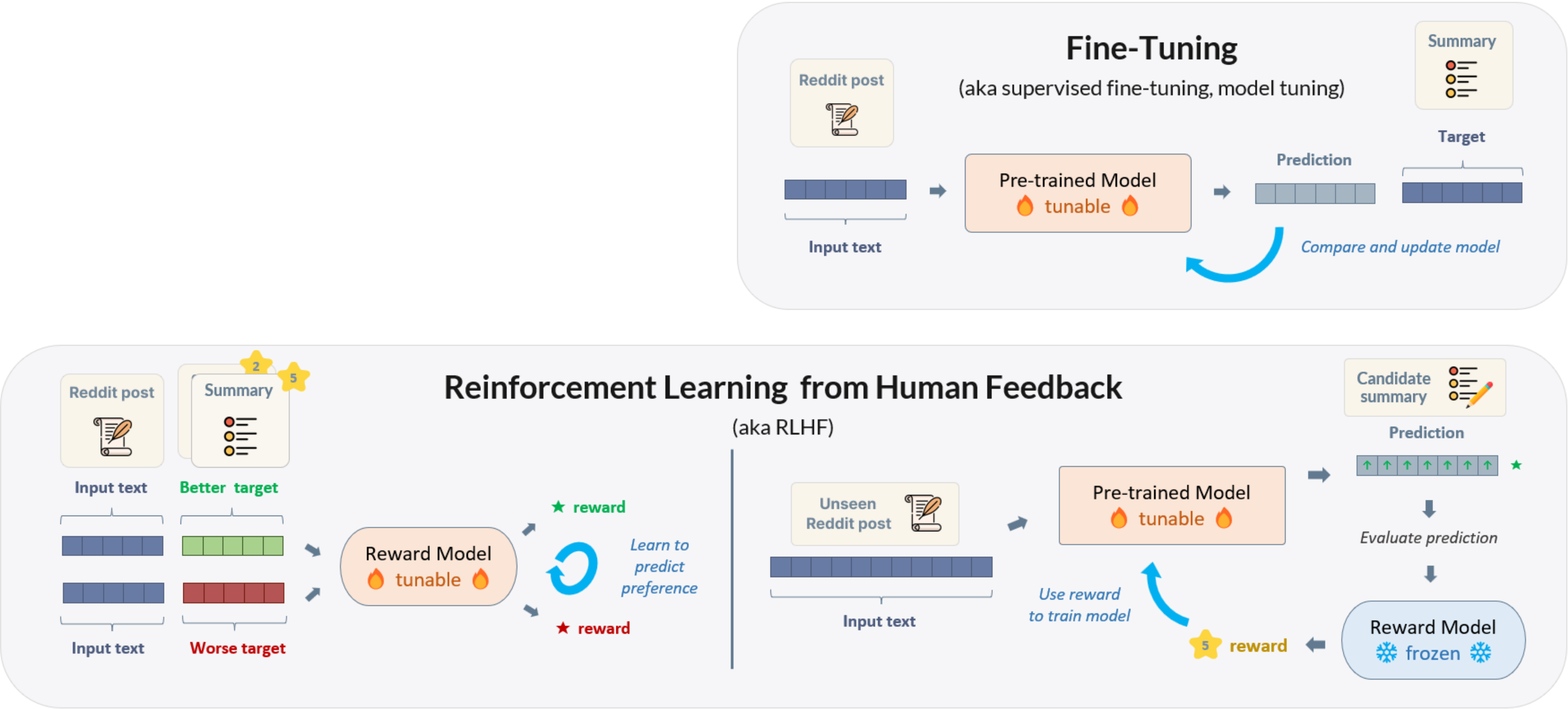


1 Collect human feedback
A Reddit post is sampled from the Reddit TL;DR dataset.
Various policies are used to sample a set of summaries.
Two summaries are selected for evaluation.
j
k
A human judges which is a better summary of the post.
"j is better than k"
2 Train reward model
One post with two summaries judged by a human are fed to the reward model.
j
k
The reward model calculates a reward r for each summary.
r
r
r_j
r_k
The loss is calculated based on the rewards and human label, and is used to update the reward model.
loss = $\log(\sigma(r_j - r_k))$
"j is better than k"



1 Collect human feedback
A Reddit post is sampled from the Reddit TL;DR dataset.
Various policies are used to sample a set of summaries.
Two summaries are selected for evaluation.
j
k
A human judges which is a better summary of the post.
"j is better than k"
2 Train reward model
One post with two summaries judged by a human are fed to the reward model.
j
k
The reward model calculates a reward r for each summary.
r
r
r_j
r_k
The loss is calculated based on the rewards and human label, and is used to update the reward model.
loss = $\log(\sigma(r_j - r_k))$
"j is better than k"
3 Train policy with PPO
A new post is sampled from the dataset.
The policy π generates a summary for the post.
π
The reward model calculates a reward for the summary.
r
The reward is used to update the policy via PPO.
r

How can I use RLHF?



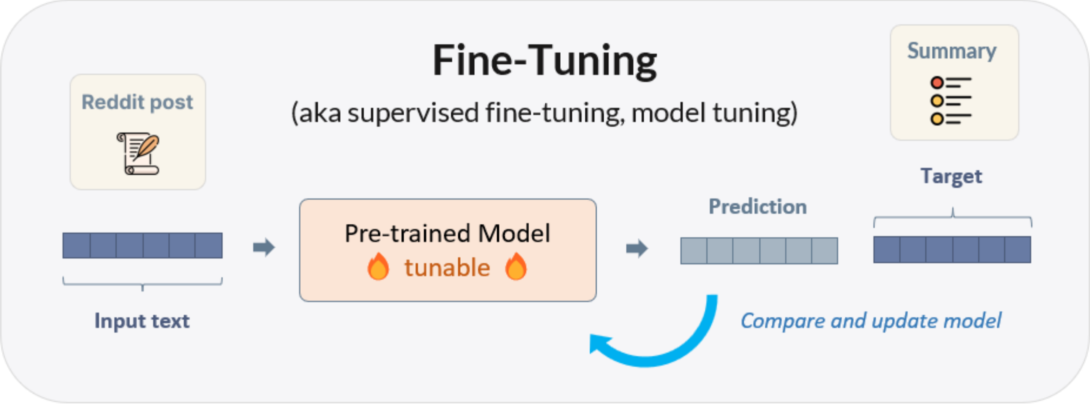
Comparison of customization methods for Reddit summarization use case.

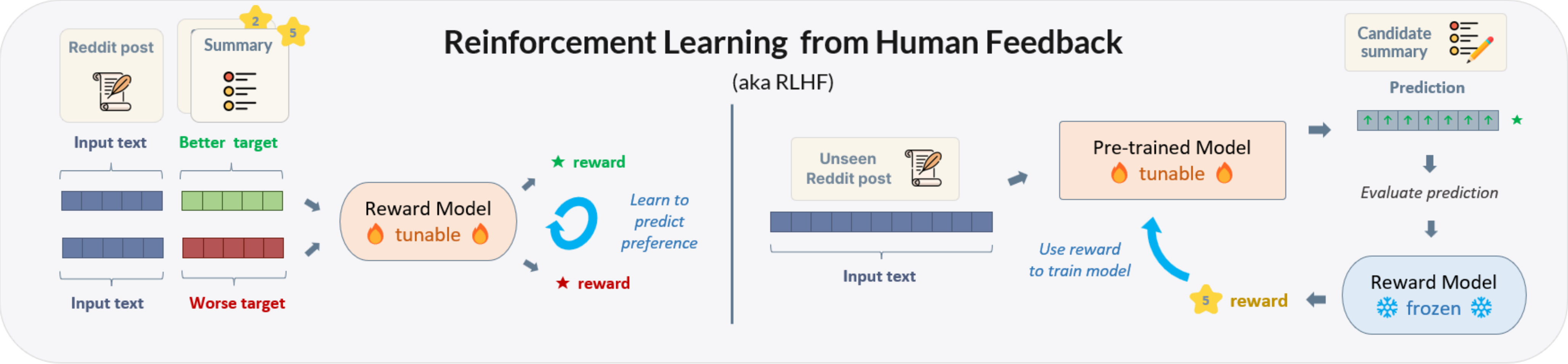
Data+AI

[RLHF | Babel (aml-babel.com)](), [[2009.01325] Learning to summarize from human feedback (arxiv.org)]()

How can I use RLHF?

Stage 1. (optional) **SFT model** learns to fit your dataset.
Or start from a pre-trained model





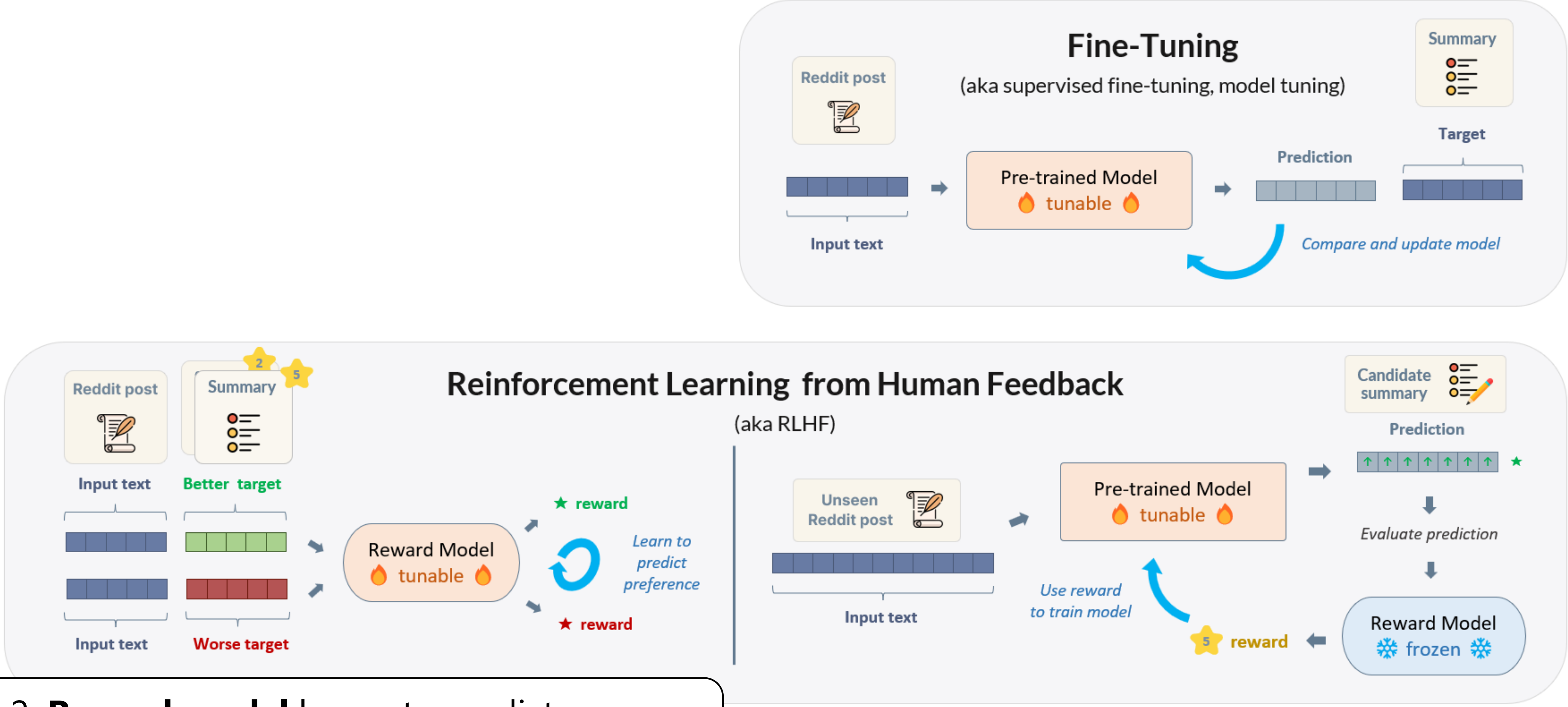
Comparison of customization methods for Reddit summarization use case.

Data+AI

[RLHF | Babel (aml-babel.com)](), [[2009.01325] Learning to summarize from human feedback (arxiv.org)]()

How can I use RLHF?

Stage 1. (optional) **SFT model** learns to fit your dataset.
Or start from a pre-trained model



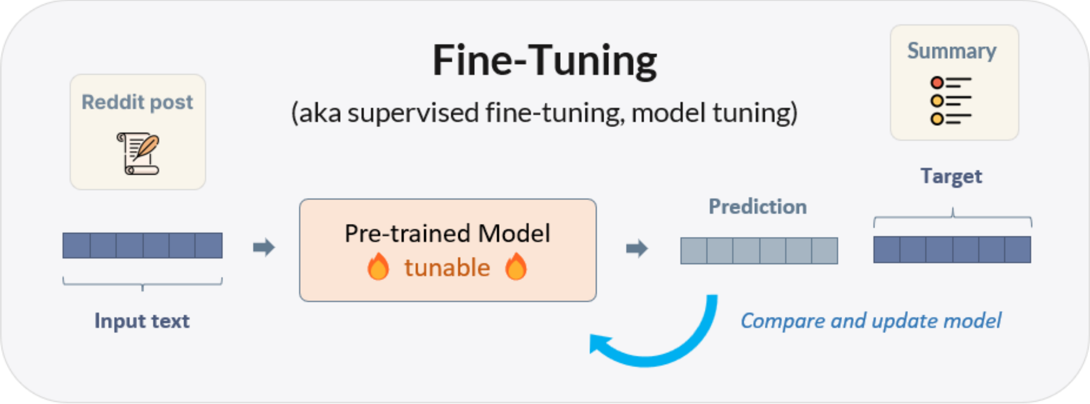
Stage 2. **Reward model** learns to predict preferences (human- or automatically-generated)

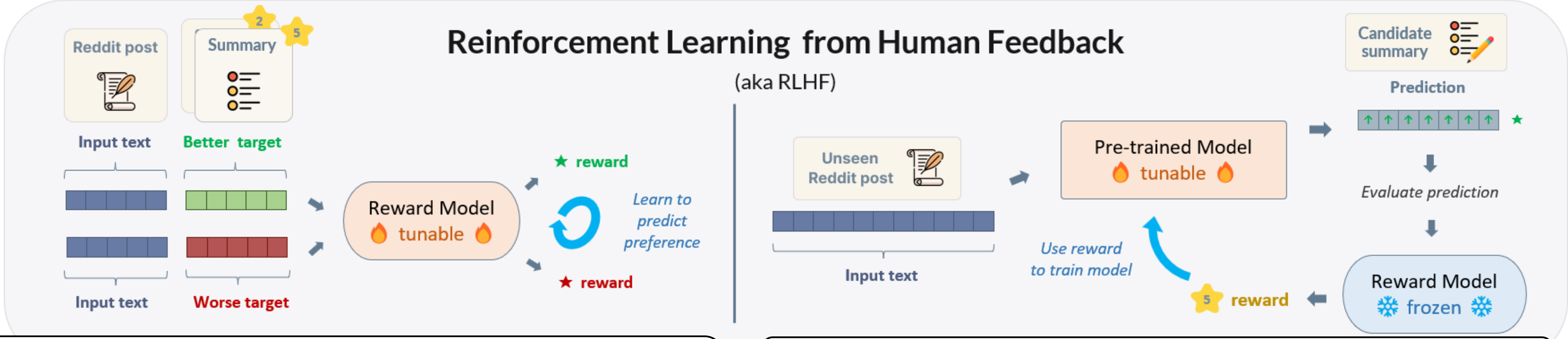
Comparison of customization methods for Reddit summarization use case.

[RLHF | Babel (aml-babel.com)](), [[2009.01325] Learning to summarize from human feedback (arxiv.org)]()

How can I use RLHF?

Stage 1. (optional) **SFT model** learns to fit your dataset.
Or start from a pre-trained model





Stage 2. **Reward model** learns to predict preferences (human- or automatically-generated)

Stage 3. **PPO model** learns to maximize reward

[RLHF | Babel (aml-babel.com)](), [[2009.01325] Learning to summarize from human feedback (arxiv.org)]()

Reinforcement Learning (RL) terminology

- **Agent**: actor in an environment, learner
- **Environment**: everything the agent can interact with, static
- **Policy**: the agent's strategy for selecting the next action
- **Action space**: set of actions that the agent can take
- **Observation space**: set of input states on which agent can train its policy
- **Reward**: scalar signal received by agent after taking an action

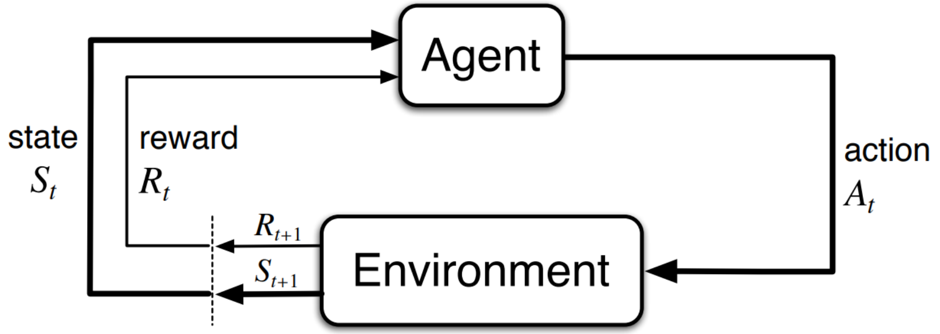
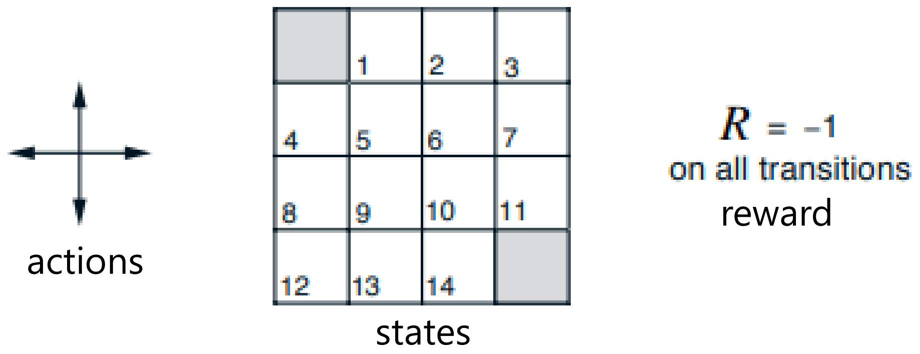


Figure 3.1: The agent–environment interaction in reinforcement learning.

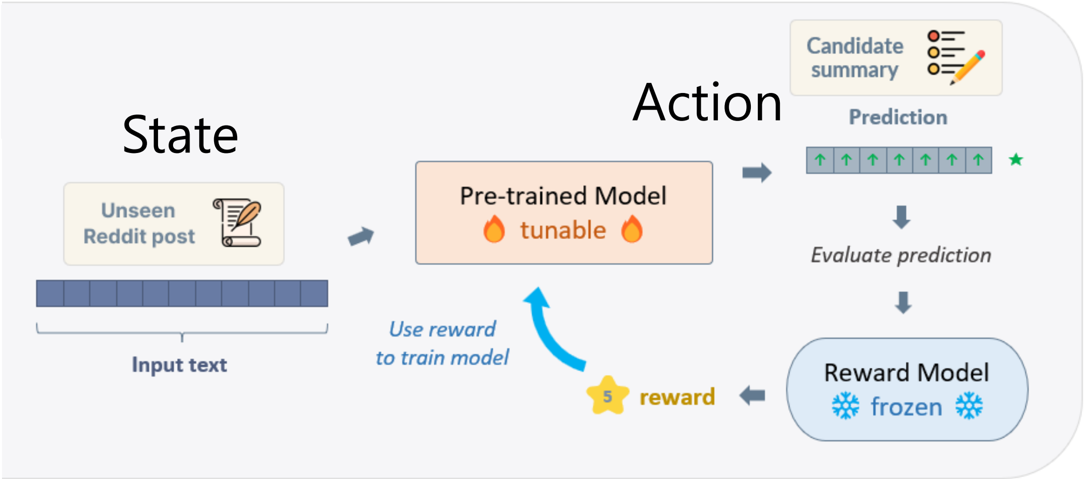
Example 4.1 Consider the 4×4 gridworld shown below.



Sutton & Barto

Reinforcement Learning (RL) part of RLHF

- **Agent/Policy**: language model
- **Environment**: language modeling task + reward model
- **Action space**: all tokens in the vocabulary
- **Observation space**: all possible input token sequences
  - Large! Dimension of $|vocabulary|^{|sequence|}$
- **Reward**: score returned by reward model



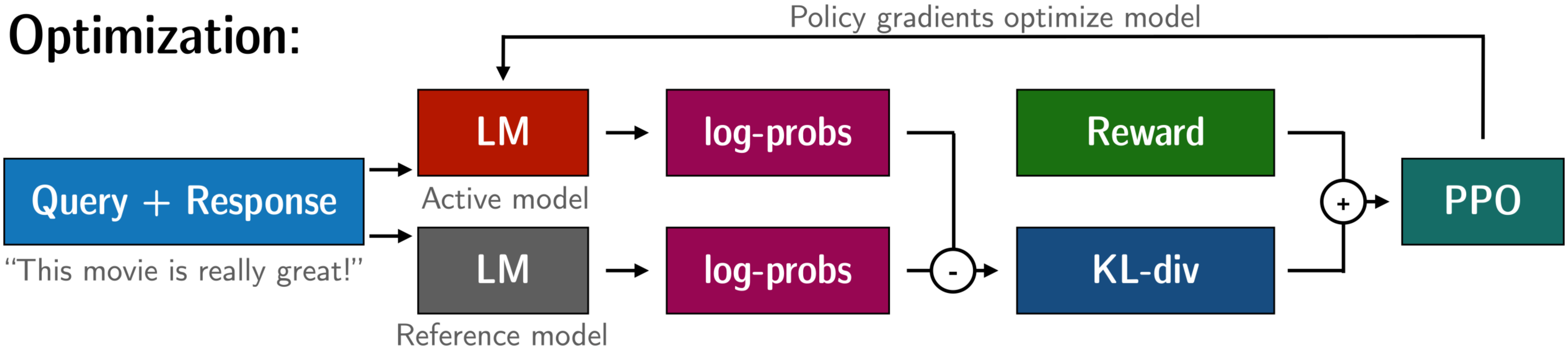
Sources: [Sutton & Barto](), [Illustrating Reinforcement Learning from Human Feedback (RLHF) (huggingface.co)]()

[lvwerra/trl (github.com)](https://github.com/lvwerra/trl)

Rollout:

Evaluation:

Optimization:



Reward model training

Format	Input	Label	Loss	Example
Pairwise comparison	prompt, completion_0, completion_1	binary choice: which completion is preferred	BCE	Hello, - world! ☑ - Mars! ✗ Choice: completion 0
Scalar reward	prompt, completion	floating-point reward value	MSE	Hello, - world! Reward: 8.0 - Dolly! Reward: 6.0 - Mars! Reward: 1.0
Binary reward	prompt, completion	binary reward value: 0 = bad, 1 = good	BCE	Hello, - world! Reward: 1 - Mars! Reward: 0

Data+AI

Open-Source tools for RLHF

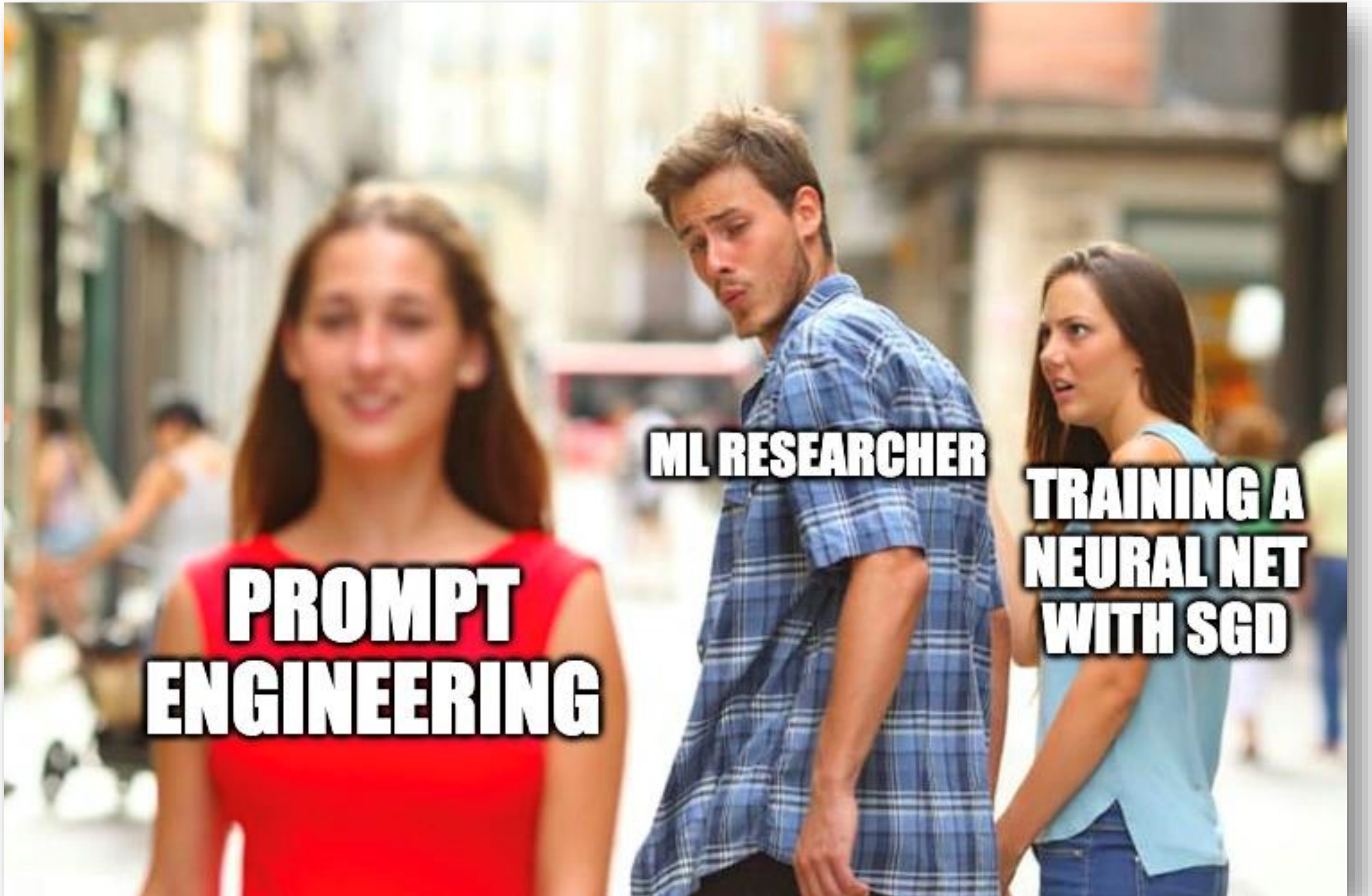
Data+AI

November 30 2022

ChatGPT is released

Data+AI

Boom, ChatGPT is out!
And just like that, scientists turned into Prompt Engineers.



Alright, for real though, prompting is cool.

Sam Altman @sama

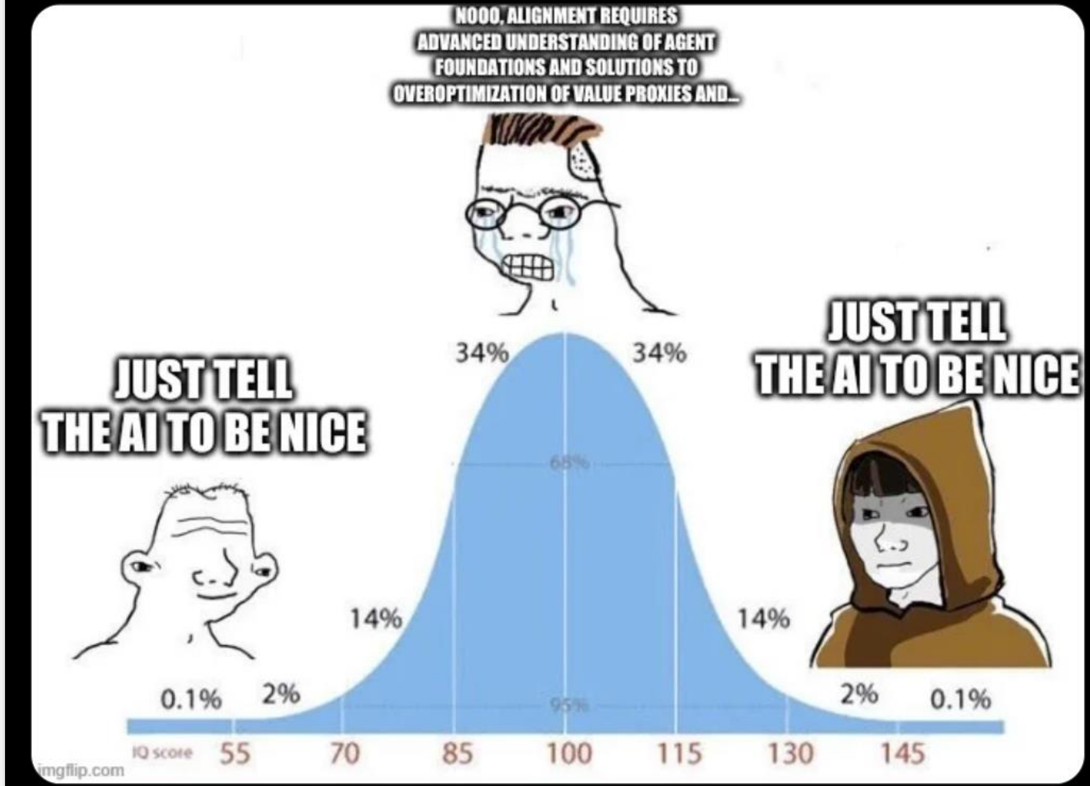
writing a really great prompt for a chatbot persona is an amazingly high-leverage skill and an early example of programming in a little bit of natural language

1:23 am · 21 Feb 2023 · **782.8K** Views

476 Retweets **111** Quote Tweets **4,917** Likes

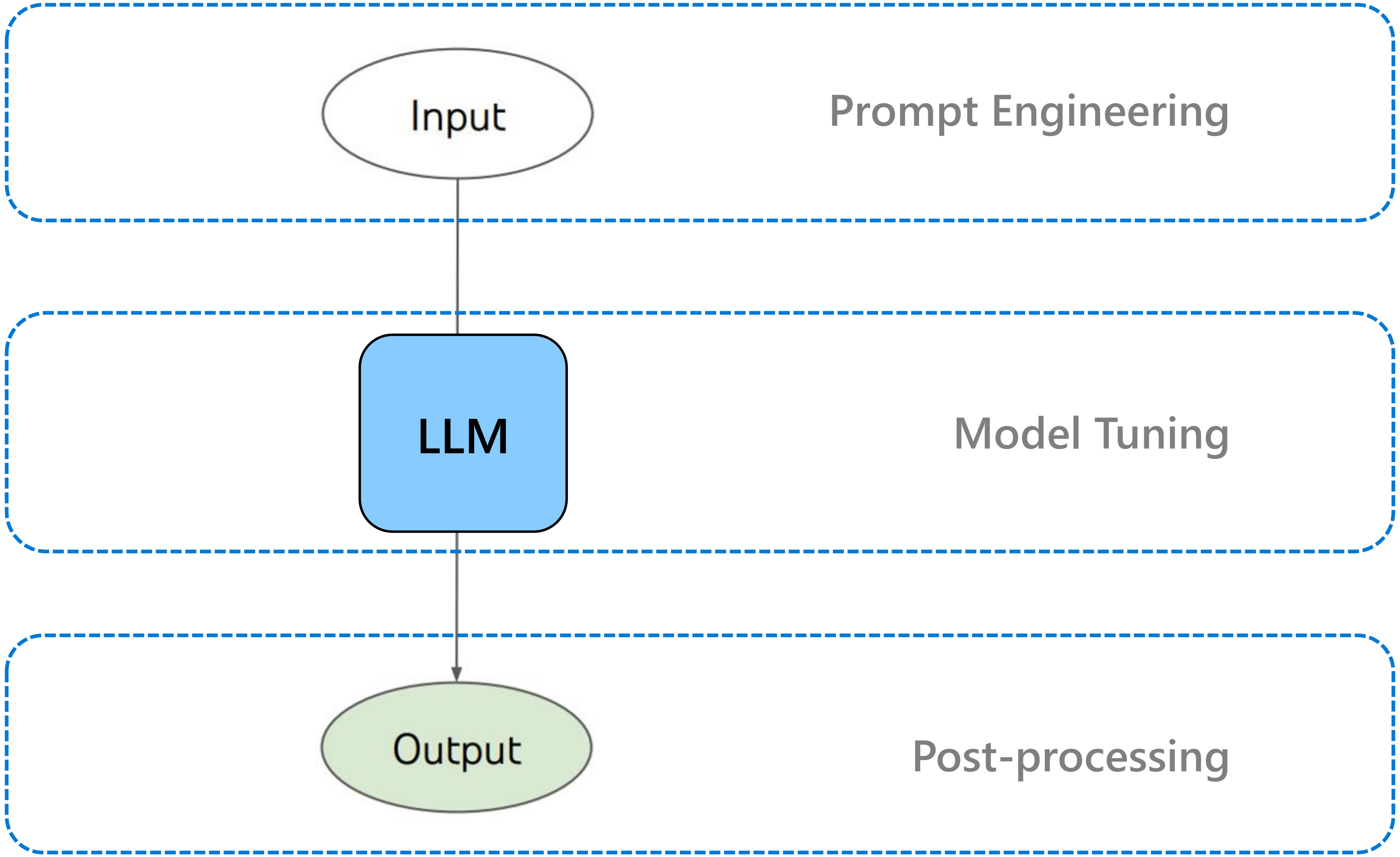
Andrej Karpathy @karpathy

The first time I was personally shook by this philosophy was when I saw the "Just tell the AI to be nice" meme on my Twitter, which is the same idea - GPT can be seen as a super multi-task policy (trained via supervised learning), and prompt engineering is the goal conditioning.



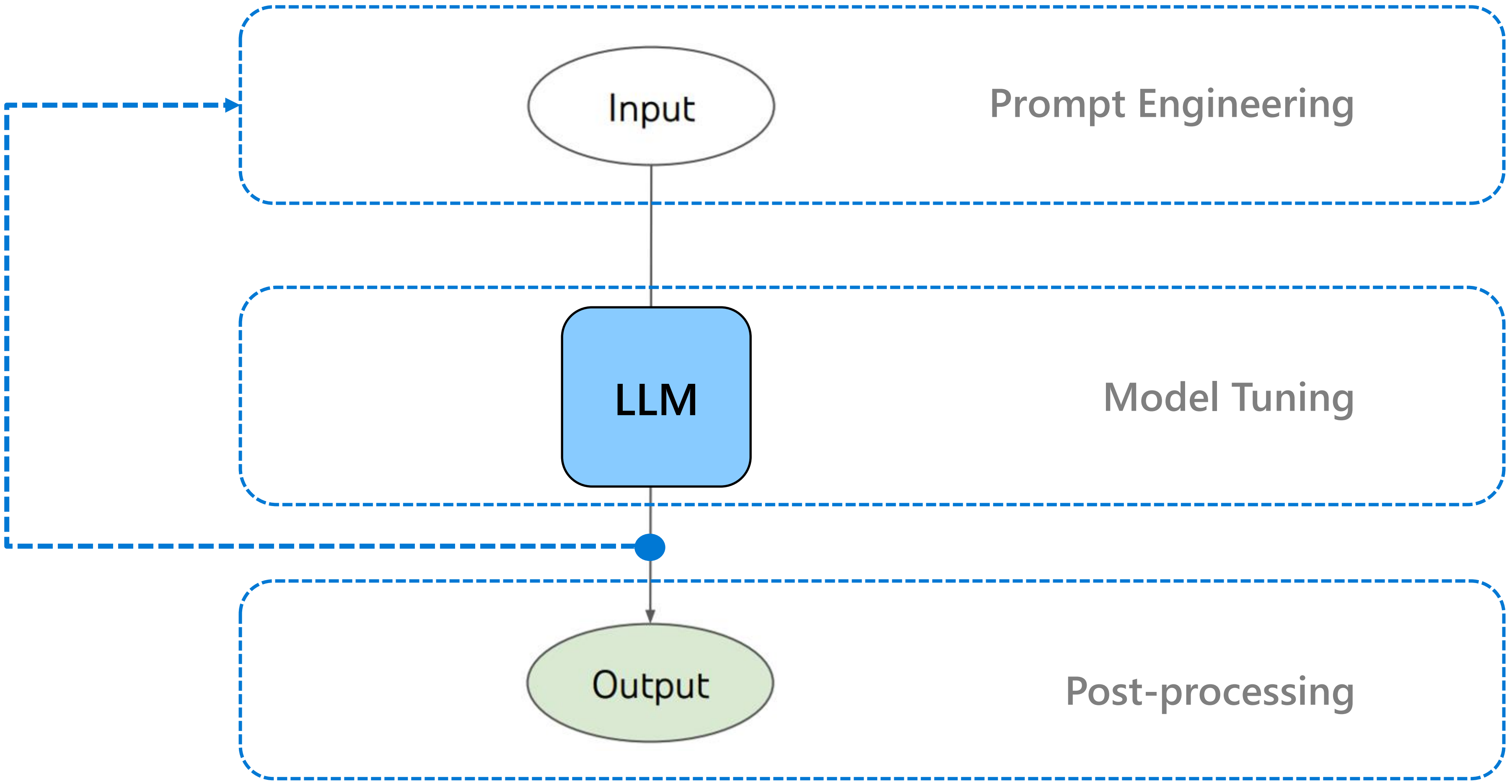
Prompting

Prompt engineering involves crafting the input to the LLM in order to guide the model towards the best and most accurate response.



Prompting

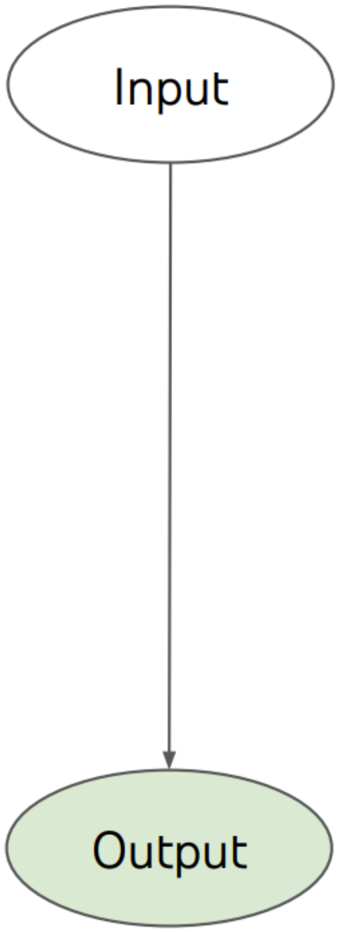
Prompt engineering involves crafting the input to the LLM in order to guide the model towards the best and most accurate response.



Input-Output Prompting

Zero-shot

Review: These wireless earbuds are amazing! The sound quality is superb, and they fit comfortably in my ears.
Sentiment:



(a) Input-Output Prompting (IO)

Data+AI

Input-Output Prompting

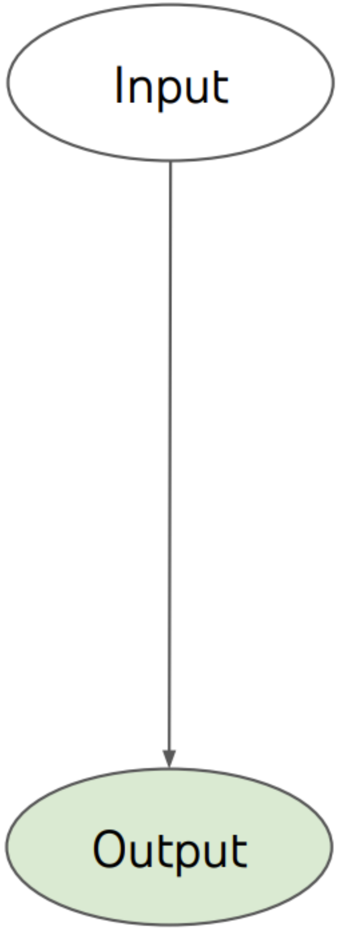
Few-shot

Review: The fitness tracker exceeded my expectations. It accurately tracks my steps and heart rate, and the app is easy to use.
Sentiment: positive

Review: I regret buying this fitness tracker. It constantly gave inaccurate readings, and the battery life is bad.
Sentiment: negative

Review: This blender is a game-changer in my kitchen.
Sentiment: positive

Review: These wireless earbuds are amazing! The sound quality is superb, and they fit comfortably in my ears.
Sentiment:



(a) Input-Output Prompting (IO)

Data+AI

How to select examples?

Retrieval

- K-NN Clustering
- Contrastive Learning

Choose examples that are semantically similar to the test example using k-NN clustering in the embedding space

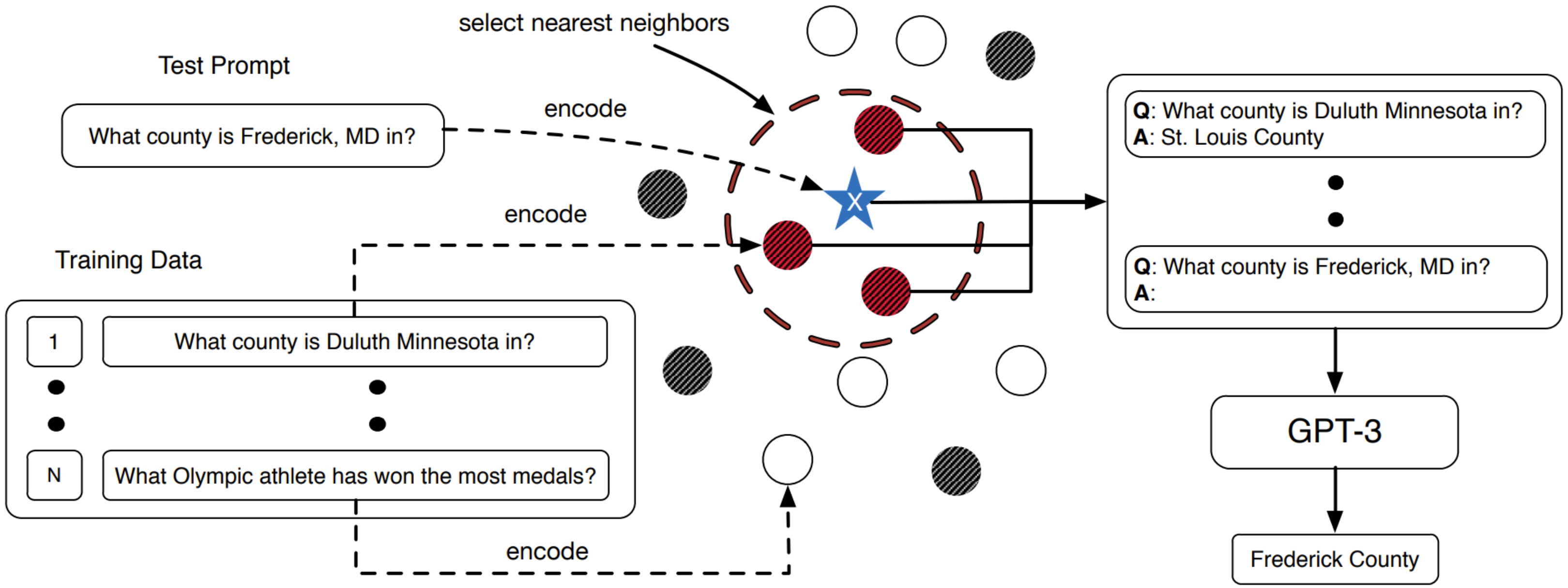


Figure 2: In-context example selection for GPT-3. White dots: unused training samples; grey dots: randomly sampled training samples; red dots: training samples selected by the k-nearest neighbors algorithm in the embedding space of a sentence encoder.

What Makes Good In-Context Examples for GPT-3? [Liu et. al]

Input-Output Prompting

Few-shot

General Suggestions

- Diverse selection of examples
- Relevant to the test sample
- In random order to avoid majority label bias and recency bias.

Data+AI

Input-Output Prompting

Instruction Prompting

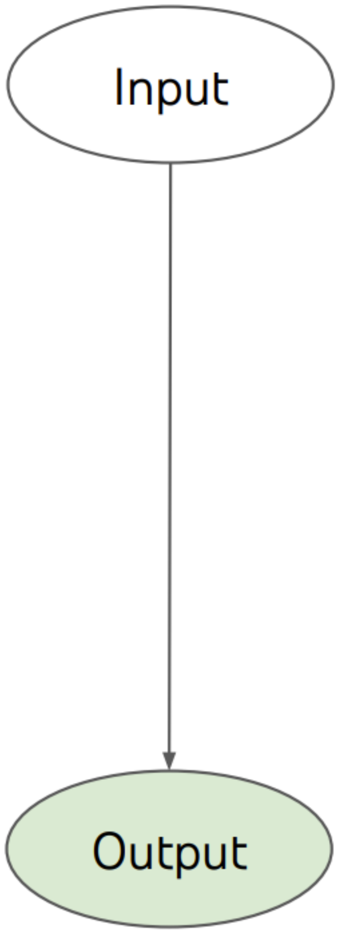
Few-shot learning might incur high token costs, which constrain the input/output budget.

Why not just give the instruction directly to the LLM?

Instruction: You are provided with a review for a product. Analyze the review and extract the sentiment. The sentiment label should be "positive" or "negative”.

Review: These wireless earbuds are amazing! The sound quality is superb, and they fit comfortably in my ears.

Sentiment:



(a) Input-Output Prompting (IO)

Data+AI

Chain of Thought Prompting (CoT)

Idea

Generate a series of concise sentences that outline reasoning steps, referred to as reasoning chains or rationales, culminating in the ultimate solution.

Effectiveness

- Effective for complex tasks
- Marginal improvements for simple task

Model Input

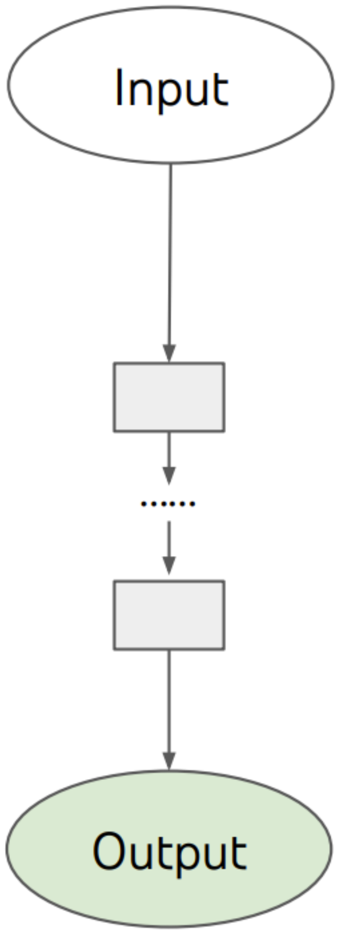
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓



(c) Chain of Thought Prompting (CoT)

Data+AI

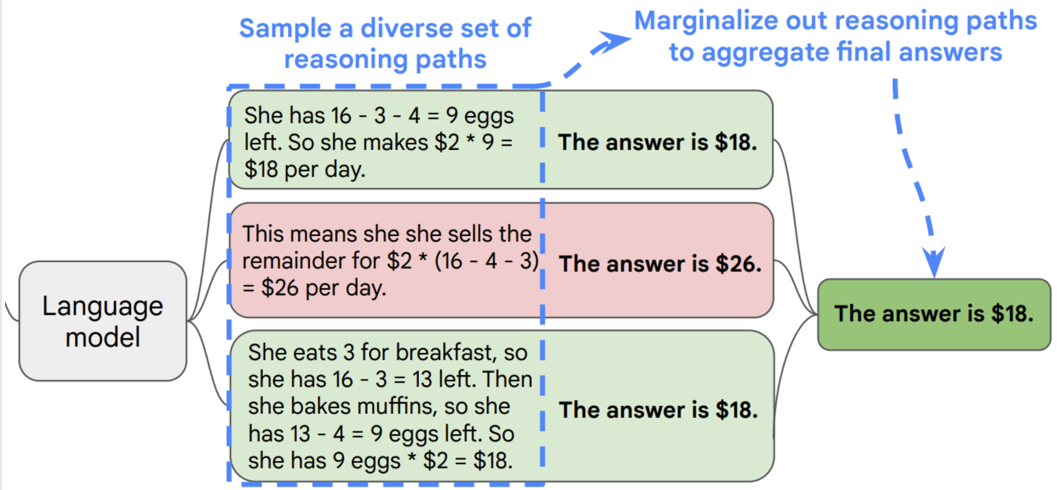
Chain-of-Thought Prompting Elicits Reasoning in Large Language Models [Wei et. al]

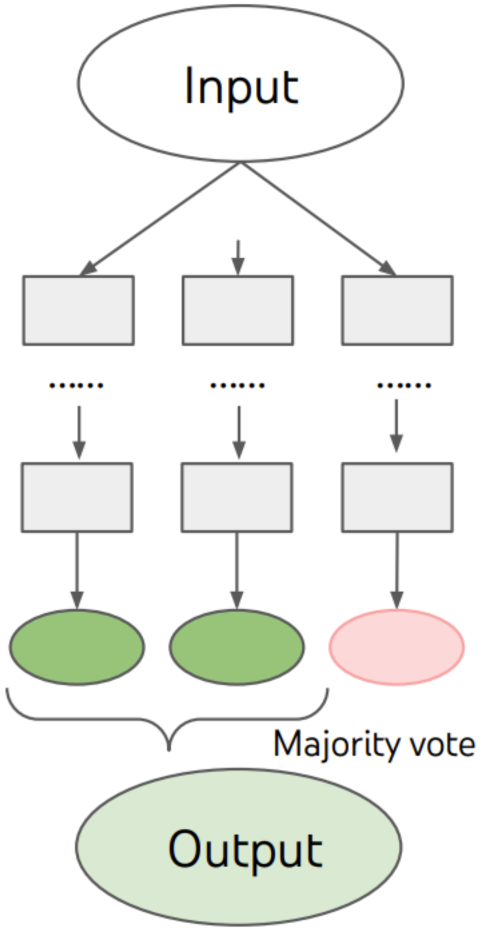
Chain of Thought Prompting (CoT)

with Self Consistency

Idea

1. Sample a diverse set of reasoning paths
2. Take a majority vote
   - The model itself
   - External validator





(c) Self Consistency with CoT (CoT-SC)

Data+AI

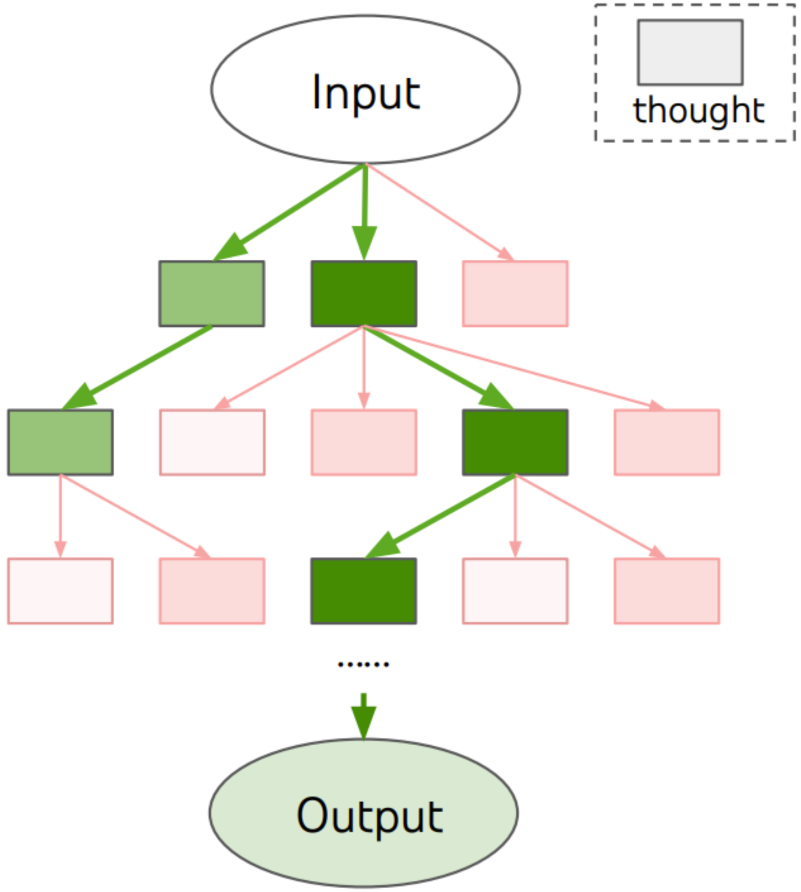
Self-Consistency Improves Chain of Thought Reasoning in Language Models [Wang et. al]

Tree of Thoughts Prompting (ToT)

Idea

1. Decomposes the problem into multiple thought steps
2. Generates multiple thoughts per step, essentially creating a tree structure.
3. Explore the tree with BFS or DFS
4. Validate each step (voting)

Data+AI



(d) Tree of Thoughts (ToT)

Tree of Thoughts: Deliberate Problem Solving with Large Language Models [Yao et. al]

Ok, but what about the SE part?

Data + AI

Ok, but what about the SE part?

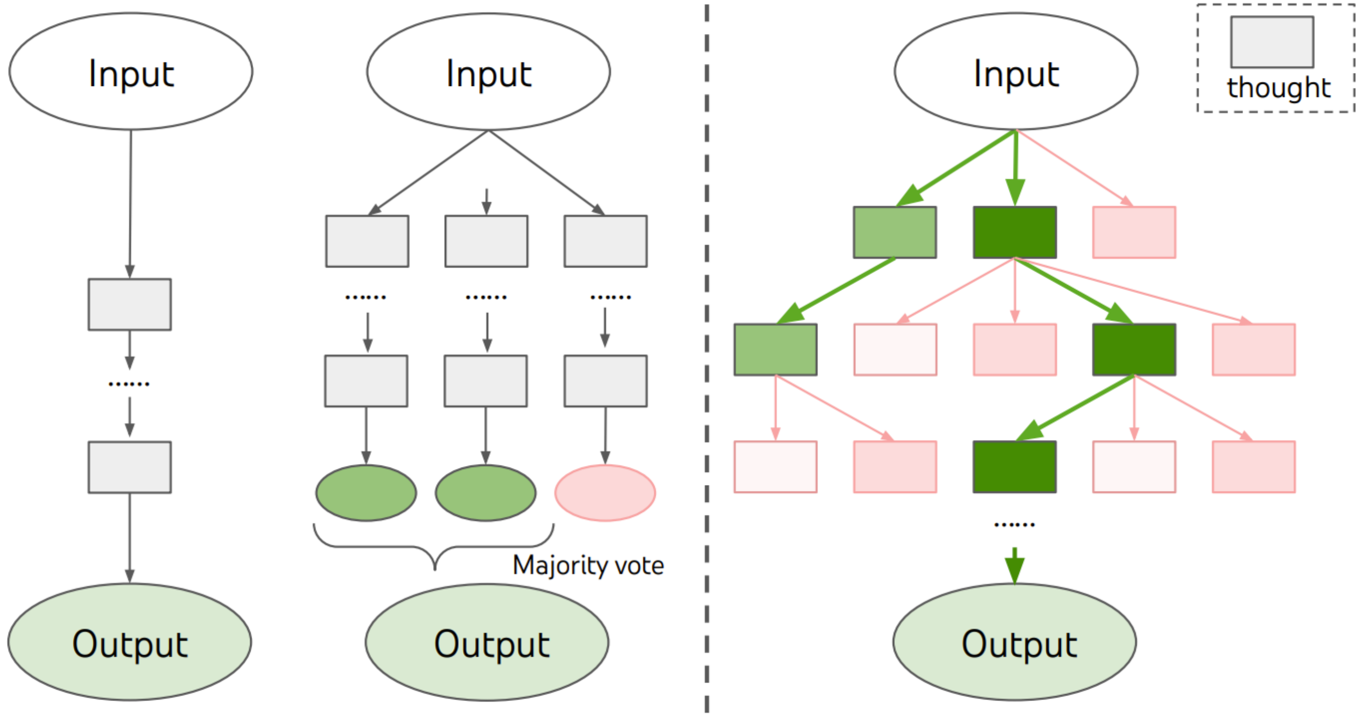
✨ That’s your time to shine! ✨

Data+AI

Design a 3-steps CoT or ToT for Performance Bug Resolution

Steps

1. Work individually or in team
2. Take up to 5 mins
3. Submit your design (be concise)
4. Vote the best design (not your own)



(c) Chain of Thought Prompting (CoT)

(c) Self Consistency with CoT (CoT-SC)

(d) Tree of Thoughts (ToT)

Data+AI

Join at menti.com use code 2738 6756

Mentimeter

Design a 3-steps CoT or ToT for Performance Bug Resolution

Voting in progress

Again

GO TO
menti.com
ENTER THE CODE
2738 6756

1

Microsoft

InferFix: End-to-End Program Repair
with LLMs over Retrieval-Augmented Prompts

Speaker: **Michele Tufano**
Co-authors: Matthew Jin, Syed Shahriar, Xin Shi, Shuai Lu, Neel Sundaresan, Alexey Svyatkovskiy

Data+AI

InferFix: End-to-End Program Repair

Problem

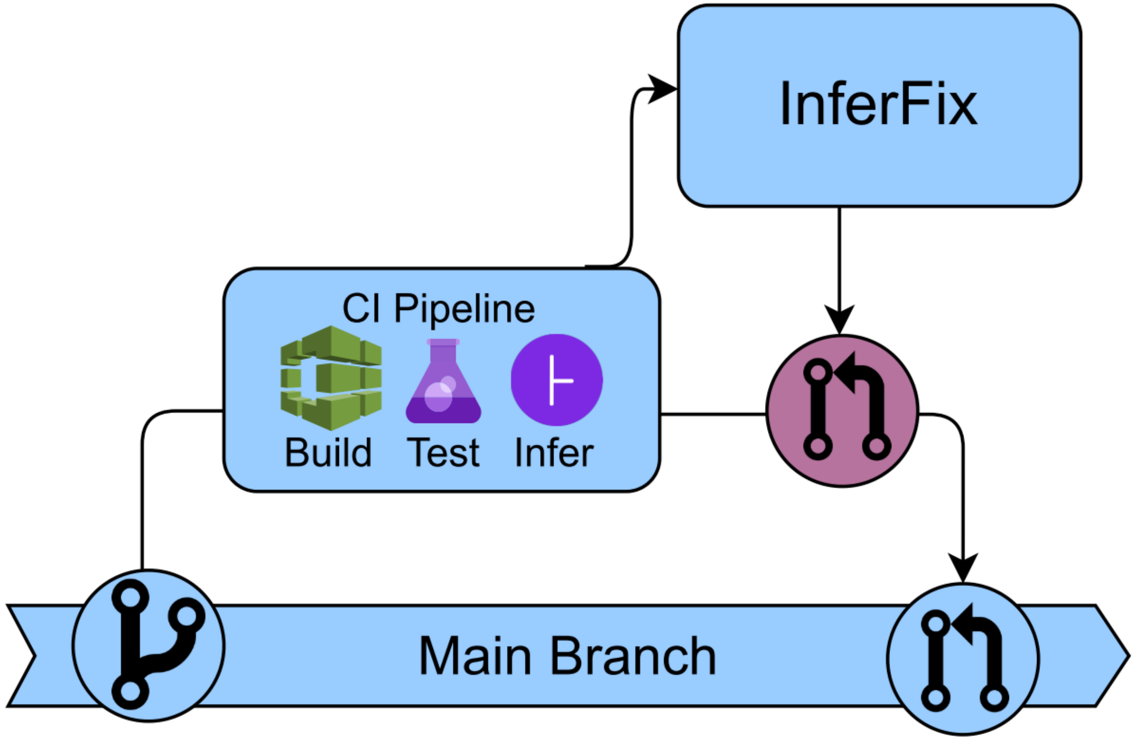
Detect and fix critical bugs for security, reliability, and performance issues.
Automate these steps for developers in the Continuous Integration (CI) pipeline

End-to-End Solution

Bug Detection -> Classification -> Localization -> Resolution
Leverage Large Language Models (LLMs)
Integrated in the CI Pipeline

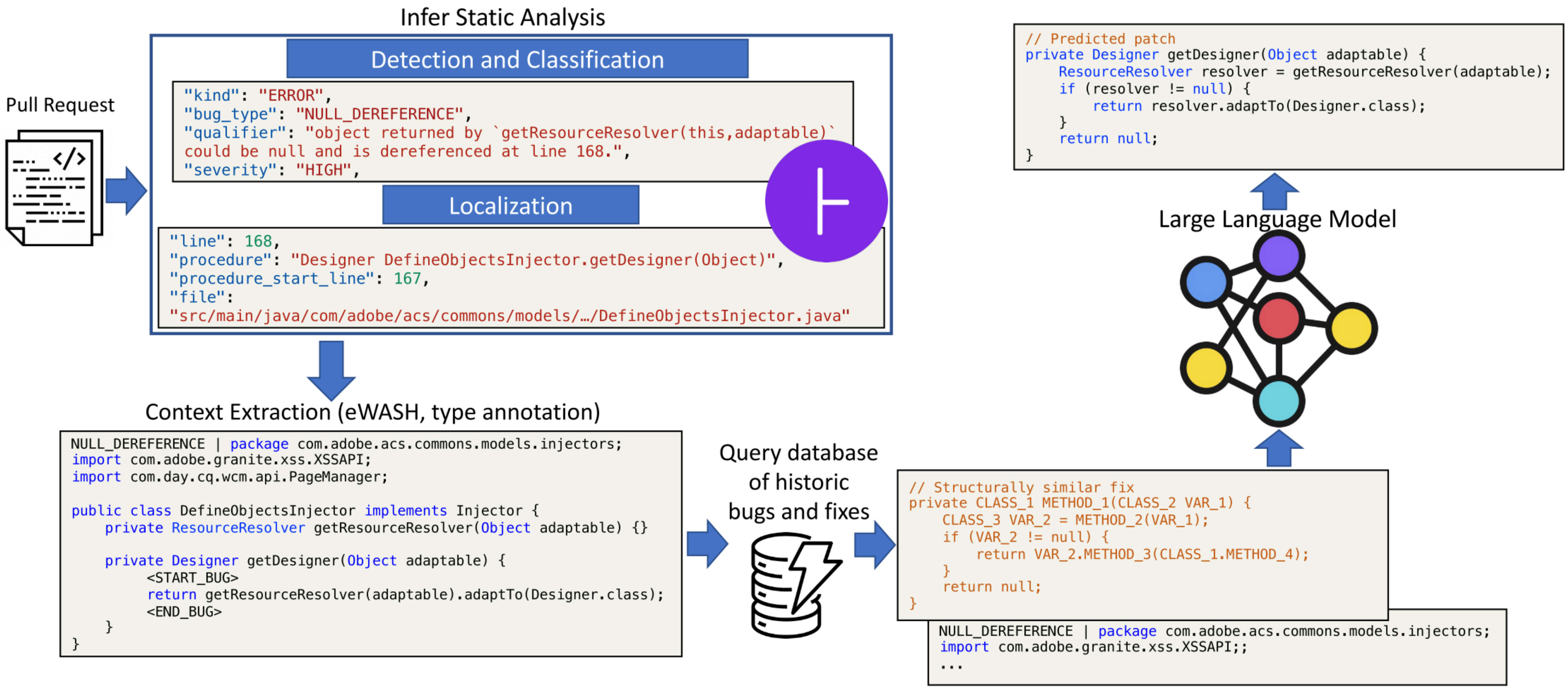
Benefits

Identify and fix bugs early during the development process
Developers can focus on faster delivery of new features



Data+AI

Overview - InferFix: End-to-End Program Repair



Infer

Infer is a static analyzer that relies on formal verification to detect software errors statically.

Null Pointer Dereference

Program attempts to access or manipulate data using a null pointer

```
p = foo(); // foo() might return null
stuff();
p.goo();   // dereferencing p, potential NPE
```

Resource Leak

A resource (file, database, etc.) is not properly released or closed after it is no longer needed, potentially leading to unexpected behavior.

```
// Standard idiom
Allocate resource
try {
  do some stuff
} finally {
  close resource
}
```

Thread Safety Violation

Concurrent access or modification of shared data by multiple threads leads to unexpected and incorrect results due to race conditions and lack of synchronization.

```
public class CounterClass {
    private int count;

    public void increment() {
        count++;
    }
}
```

Data+AI

InferredBugs Dataset

Repositories
- 2.9k Java
- 3.3k C#
- 1 million commits

Infer Analysis
- Analyze change history of a repo
- Detect Bug Introduction
- Detect Bug-Fix

Bug Data
- Bug type
- Bug Location
- Introduction/Fix in the change history

Bug-Fixes
- Java: 8,650
- C#: 2,945
- Total: 11,595

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
Num. bug patches	2686	1116	2382	1789	3582	40
Mean lines per patch	12.2	8.8	10.9	7.2	14.1	17.1
Mean char per patch	457.1	310.2	404.1	275.8	482.7	455.3

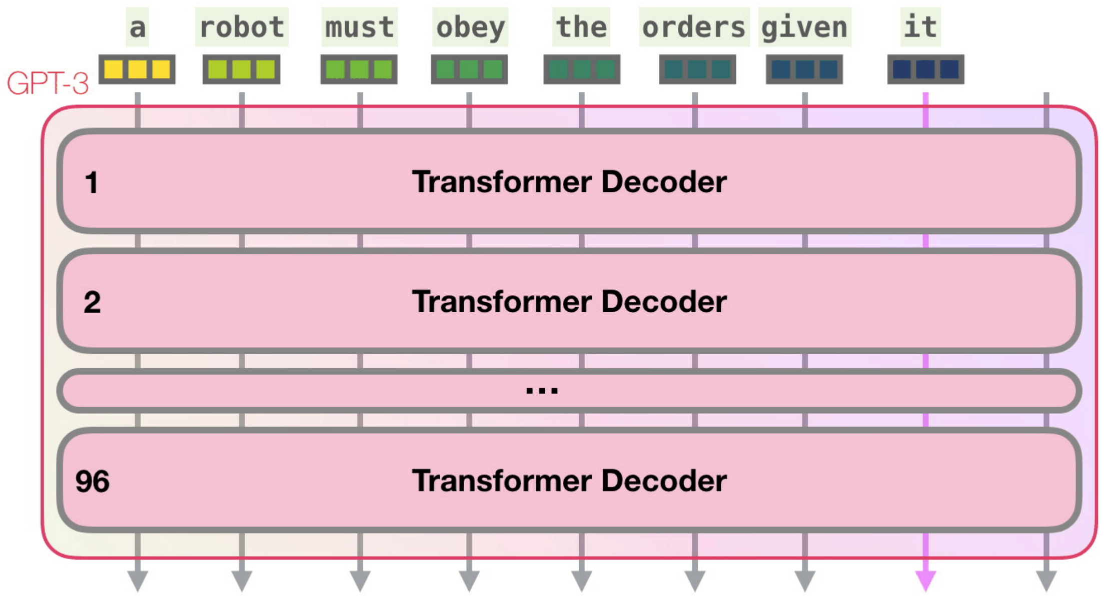
Data+AI

Large Language Models

OpenAI GPT-3 style LLMs

Codex (code-cushman-001)
- 12 billion parameters
- Pretrained specifically on code
- Efficient to finetune on specific tasks

Davinci (text-davinci-003)
- 175 billion parameters
- Similar to ChatGPT
- Expensive to finetune



Data+AI

Images from: https://jalammar.github.io/

LLM Prompting Strategies

Completion

// Buggy code

Buggy Code

// Fixed code

Demonstration

// Buggy code

Buggy Code

// Fixed code

Fixed Code

// Buggy code

Buggy Code

// Fixed code

Instruction

You are an expert developer.

When given a buggy code ...

// Buggy code

Buggy Code

Output fixed code:

Fixed Code

Data+AI

InferFix Building Stages

1. Prompting Strategies vs Finetuning Model
2. Adding Bug Type information
3. Adding Bug Localization information
4. Extended File-level Context
5. Enriching context with Bug-Fix Hints

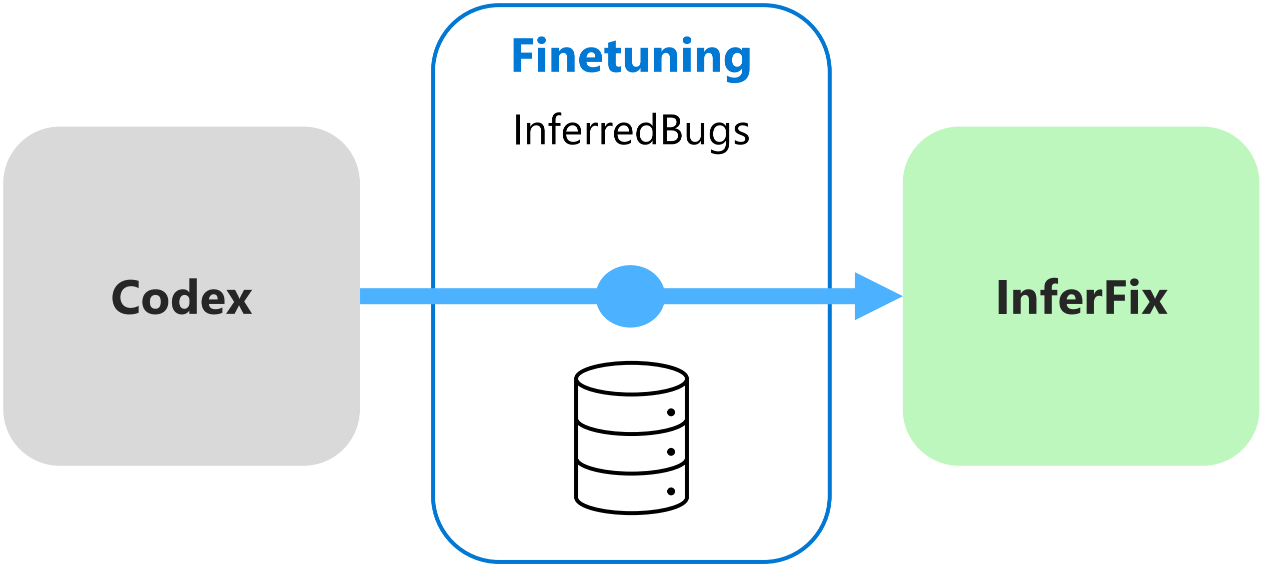
Prompting Strategies vs Finetuning

Metric: Perfect Match
Identical predictions to the dev's fix

Basic Prompt
Just the buggy Code

6.2% - 36.7% Improvement

InferFix over second best Instruction (Davinci)



Approach	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
Demonstration (Codex)	20.3	30.1	25.3	29.1	19.0	16.7
Completion (Codex)	6.7	6.1	7.8	5.7	3.9	0.0
Instruction (Davinci)	40.5	22.2	53.8	19.7	41.3	33.3
InferFix (basic prompt)	49.7	58.1	60.0	51.9	64.4	70.0

Data+AI

Bug Type

1.4% - 3.5% Improvement

Adding bug info
to the prompt

Data+AI

```
NULL_DEREFERENCE
```

```java
private Designer getDesigner(Object adaptable) {
    return getResourceResolver(adaptable).adaptTo(Designer.class);
}
```

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (basic prompt)	49.7	58.1	60.0	51.9	64.4	70.0
InferFix (+ bug type)	52.3	60.4	63.1	53.3	67.9	72.5

Bug Localization

```
NULL_DEREFERENCE
```

```
private Designer getDesigner(Object adaptable) {
    <START_BUG>
    return getResourceResolver(adaptable).adaptTo(Designer.class);
    <END_BUG>
}
```

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (bug type)	52.3	60.4	63.1	53.3	67.9	72.5
InferFix (+ localization)	53.5	61.4	64.4	53.9	69.6	75.0

0.6% - 2.5% Improvement

Adding bug localization to the prompt

Data+AI

Extended File-Level Context with eWASH

Problem

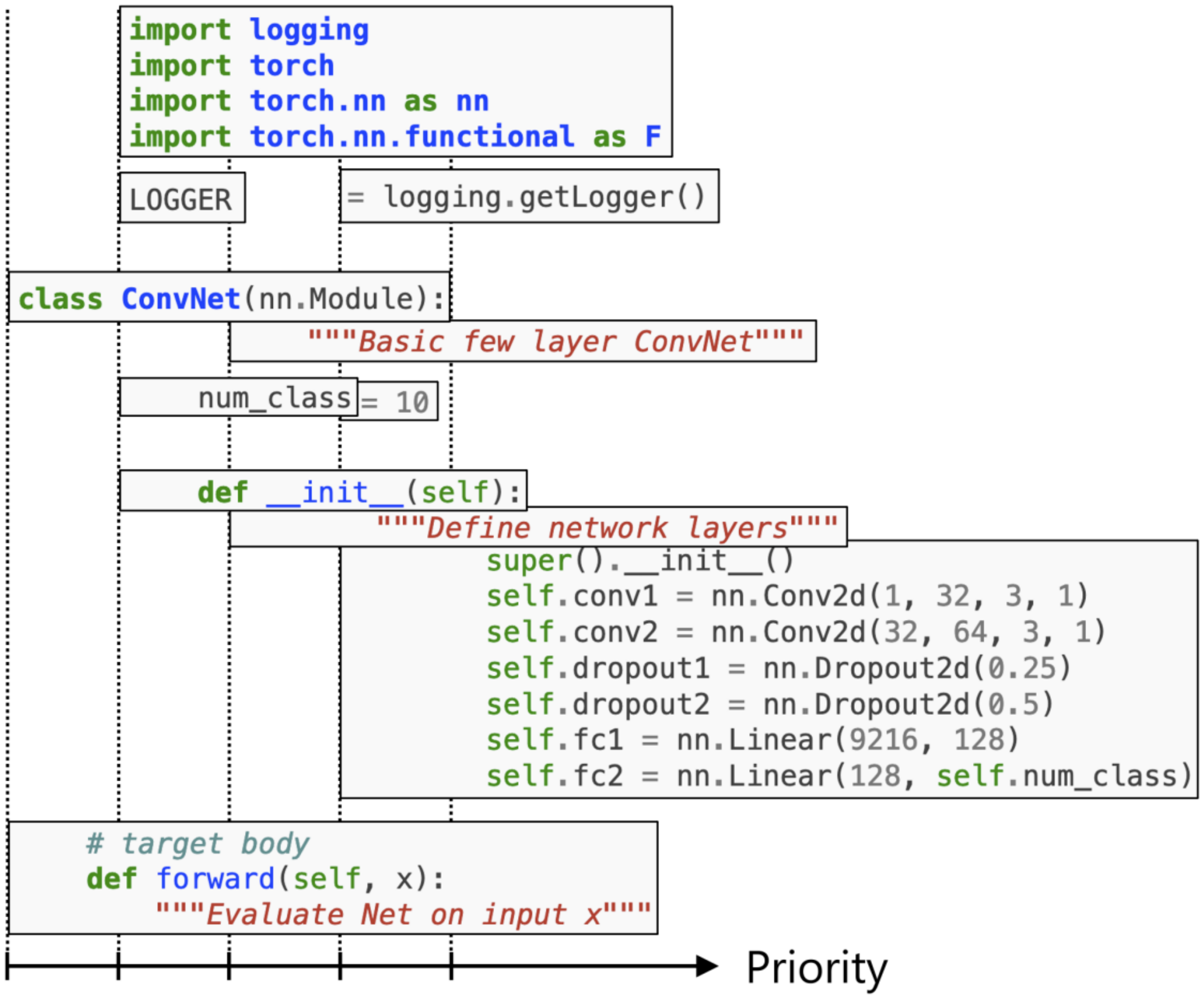
Provide as much code context as possible from the buggy file
Model input is limited in tokens, and file may be truncated

eWASH Approach

Allows to fill the model input with as much context as possible

Defines a syntax-based priority system to dynamically choose context based on available token budget

1. Buggy Method and Class Name
2. Imports, variables, and method signatures
3. Method docstrings
4. Method bodies



Data+AI

Extended Context with eWASH

Bug type annotation

```
NULL_DEREFERENCE
```

eWASH extended context

```
package com.adobe.acs.commons.models.injectors;
import com.adobe.granite.xss.XSSAPI;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageManager;
...
public class DefineObjectsInjector implements Injector {

    private static Designer getDesigner(Object adaptable) {}
```

Focal methods

```
    private ResourceResolver getResourceResolver(Object adaptable) {
        if (adaptable instanceof SlingHttpServletRequest) {
            return ((SlingHttpServletRequest)adaptable).getResourceResolver();
        }
        if (adaptable instanceof Resource) {
            return ((Resource)adaptable).getResourceResolver();
        }
        return null;
    }
```

Buggy method with location markers

```
    private Designer getDesigner(Object adaptable) {
        <START_BUG>
        return getResourceResolver(adaptable).adaptTo(Designer.class);
        <END_BUG>
    }
```

3.7% - 5.4% Improvement

Adding extended context to the prompt

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (localization)	53.5	61.4	64.4	53.9	69.6	75.0
InferFix (+ eWASH)	57.6	65.1	69.1	56.1	75.0	80.0

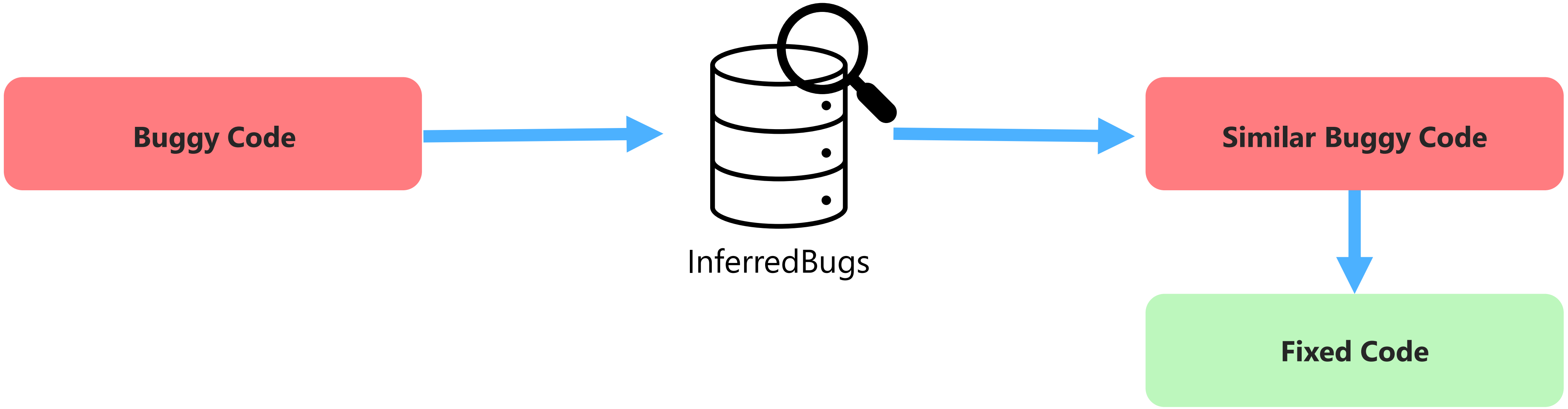
Enriching context with Bug-Fix Hints

Idea

Find examples on how to fix a similar bug, and provide it to the model

Steps

- Search for similar buggy code in a historical database of bug-fixes
- Select the fixed version of the bug
- Provide the example of the bug-fix to the model



Data+AI

Enriching context with Bug-Fix Hints

Retriever Model

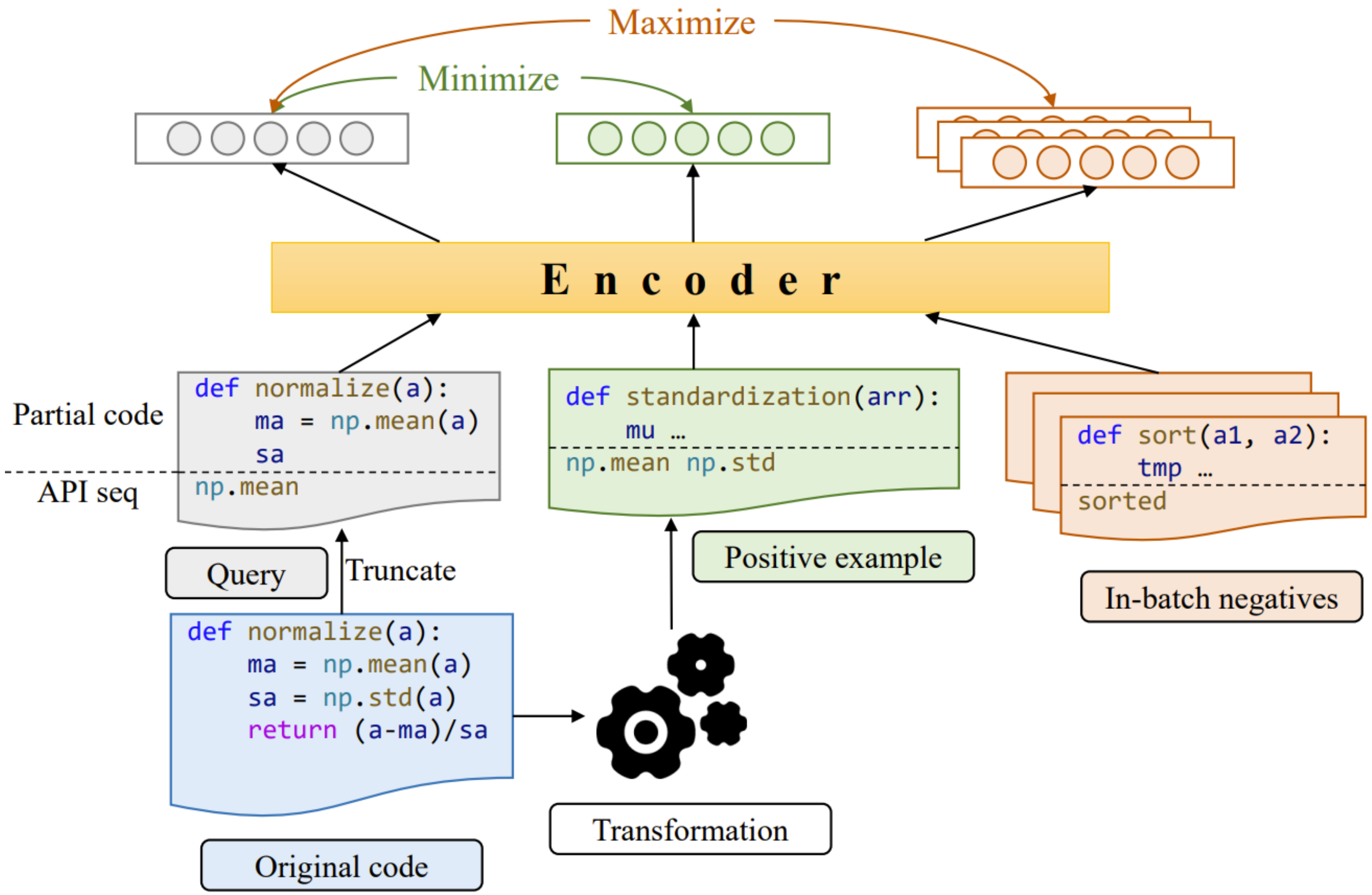
Bidirectional Transformer Encoder Model that maps a code snippet to an embedding
Trained using **contrastive learning** objective:

- Minimized distance from positive examples
- Maximize distance from negative examples

Positive Examples -> Bugs of the same type
Negative Examples -> Bugs of different type

Retrieving Steps

1. Generate embedding for given buggy code
2. Compute cosine similarity with the bugs in the db
3. Select the associated fixed code (key-value pair)



Data+AI

Enriching context with Bug-Fix Hints

Abstraction

To extract structurally similar fixes and reduce the dependency on identifier naming we obfuscate code snippets

Process

We parse and analyze the code identifier types and mask the names of classes, methods, and identifiers with placeholder symbols: CLASS_NN, METHOD_NN, and VAR_NN, where NN is a unique number

```
private Designer getDesigner(Object adaptable) {
    ResourceResolver resolver = getResourceResolver(adaptable);
    if (resolver != null) {
        return resolver.adaptTo(Designer.class);
    }
    return null;
}
```

```
private CLASS_1 METHOD_1(CLASS_2 VAR_1) {
    CLASS_3 VAR_2 = METHOD_2(VAR_1);
    if (VAR_2 != null) {
        return VAR_2.METHOD_3(CLASS_1.METHOD_4);
    }
    return null;
}
```

Enriching context with Bug-Fix Hints

0.9%- 2.5% Improvement

Adding bug-fix hints to the prompt

Data+AI

Retrieved similar fix

```
// Structurally similar fix
private CLASS_1 METHOD_1(CLASS_2 VAR_1) {
    CLASS_3 VAR_2 = METHOD_2(VAR_1);
    if (VAR_2 != null) {
        return VAR_2.METHOD_3(CLASS_1.METHOD_4);
    }
    return null;
}
```

Bug type annotation

```
NULL_DEREFERENCE
```

eWASH extended context

Focal methods

```
package com.adobe.acs.commons.models.injectors;
import com.adobe.granite.xss.XSSAPI;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageManager;
...
public class DefineObjectsInjector implements Injector {

    private static Designer getDesigner(Object adaptable) {}

    private ResourceResolver getResourceResolver(Object adaptable) {
        if (adaptable instanceof SlingHttpServletRequest) {
            return ((SlingHttpServletRequest)adaptable).getResourceResolver();
        }
        if (adaptable instanceof Resource) {
            return ((Resource)adaptable).getResourceResolver();
        }
        return null;
    }
```

Buggy method with location markers

```
    private Designer getDesigner(Object adaptable) {
        <START_BUG>
        return getResourceResolver(adaptable).adaptTo(Designer.class);
        <END_BUG>
    }
```

	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
InferFix (eWASH)	57.6	65.1	69.1	56.1	75.0	80.0
InferFix (+ retrieved hints)	59.5	66.7	71.2	57.0	77.4	82.5

Overall Results

Finetuning Boost
Finetuning on bug dataset improves performances

Prompt Augmentation
Augmenting the prompt is beneficial:
- Bug type & Location
- Context
- Bug-Fix hints

5.1% - 13% Improvement
on top of finetuning

Data+AI

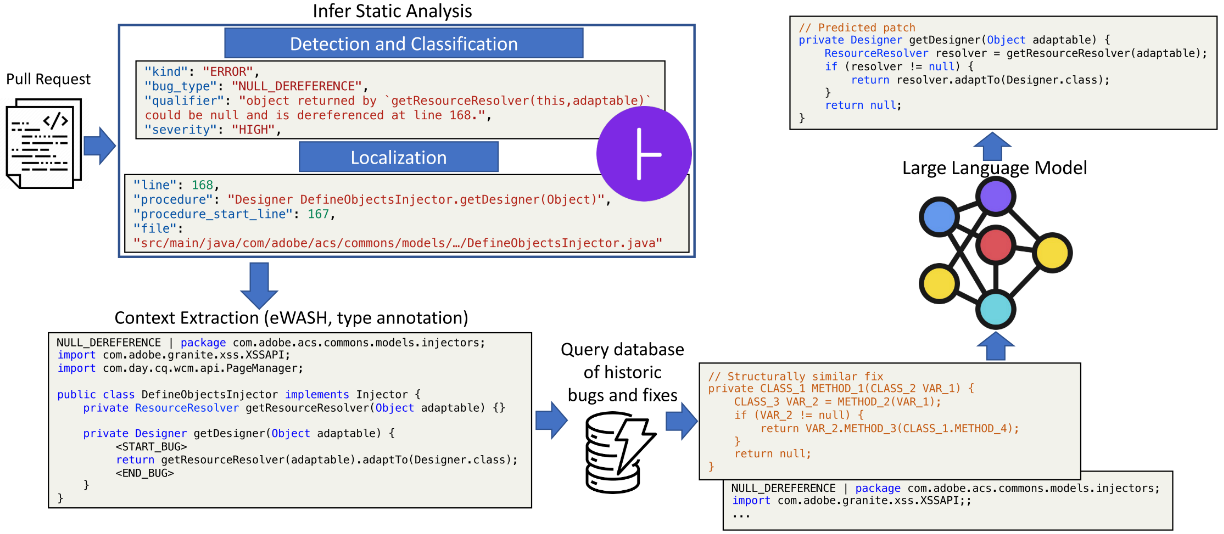
Approach	NPD		RL		TSV	
	Java	*C#*	*Java*	*C#*	*Java*	*C#*
Demonstration (Codex)	20.3	30.1	25.3	29.1	19.0	16.7
Completion (Codex)	6.7	6.1	7.8	5.7	3.9	0.0
Instruction (Davinci)	40.5	22.2	53.8	19.7	41.3	33.3
Finetuning (Codex)	49.7	58.1	60.0	51.9	64.4	70.0
InferFix	**59.5**	**66.7**	**71.2**	**57.0**	**77.4**	**82.5**

Resource Leak

```java
private static String readResource(final String name)
↪ {
    final StringBuilder ret = new StringBuilder();
    InputStream is = null;
    try {
        is = UrlRegularExpressions.class.getClassLoad⌋
        ↪ er().getResourceAsStream(name);
        final InputStreamReader reader = new
        ↪ InputStreamReader(is, ASCII);
        int read = 0;
        final char[] buf = new char[1024];
        do {
            read = reader.read(buf, 0, buf.length);
            if (read > 0) {
                ret.append(buf, 0, read);
            }
        } while (read >= 0);
    } catch (final IOException ex) {
        throw new RuntimeException(ex);
    } finally {
        closeQuietly(is);
    }
    return ret.toString();
}
```

```java
private static String readResource(final String name)
↪ {
    final StringBuilder ret = new StringBuilder();
    InputStream is = null;
    InputStreamReader reader = null;
    try {
        is = UrlRegularExpressions.class.getClassLoad⌋
        ↪ er().getResourceAsStream(name);
        reader = new InputStreamReader(is, ASCII);
        int read = 0;
        final char[] buf = new char[1024];
        do {
            read = reader.read(buf, 0, buf.length);
            if (read > 0) {
                ret.append(buf, 0, read);
            }
        } while (read >= 0);
    } catch (final IOException ex) {
        throw new RuntimeException(ex);
    } finally {
        closeQuietly(is);
        closeQuietly(reader);
    }
    return ret.toString();
}
```

InferFix: End-to-End Program Repair with LLMs



Augmented Prompt

Retrieved similar fix

```
// Structurally similar fix
private CLASS_1 METHOD_1(CLASS_2 VAR_1) {
    CLASS_3 VAR_2 = METHOD_2(VAR_1);
    if (VAR_2 != null) {
        return VAR_2.METHOD_3(CLASS_1.METHOD_4);
    }
    return null;
}
```

Bug type annotation

```
NULL_DEREFERENCE
```

eWASH extended context

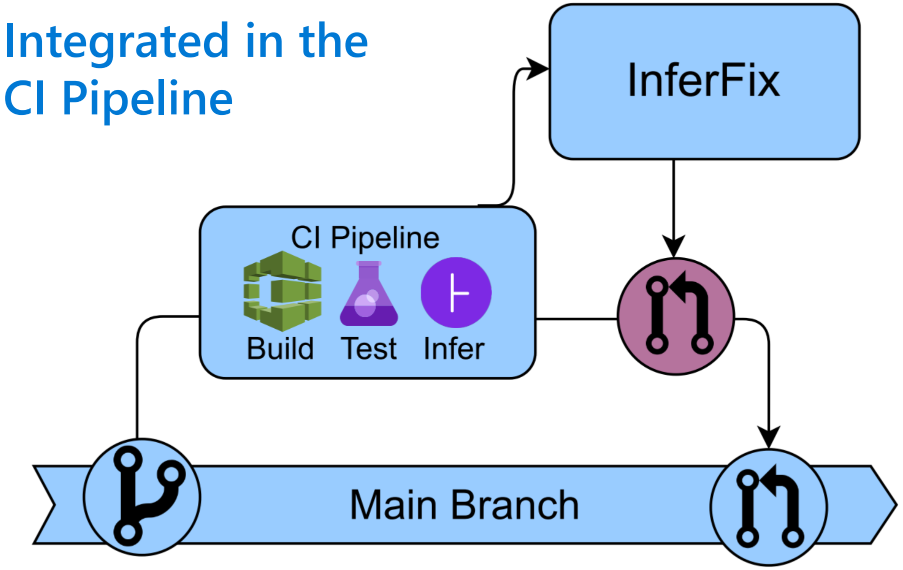
Focal methods

```
package com.adobe.acs.commons.models.injectors;
import com.adobe.granite.xss.XSSAPI;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageManager;
...
public class DefineObjectsInjector implements Injector {

    private static Designer getDesigner(Object adaptable) {}

    private ResourceResolver getResourceResolver(Object adaptable) {
        if (adaptable instanceof SlingHttpServletRequest) {
            return ((SlingHttpServletRequest)adaptable).getResourceResolver();
        }
        if (adaptable instanceof Resource) {
            return ((Resource)adaptable).getResourceResolver();
        }
        return null;
    }
```

Buggy method with location markers

```
    private Designer getDesigner(Object adaptable) {
        <START_BUG>
        return getResourceResolver(adaptable).adaptTo(Designer.class);
        <END_BUG>
    }
```

Integrated in the CI Pipeline



Paper on ArXiv

Contacts

Michele Tufano

Sr. Research Scientist

Michele.Tufano@microsoft.com

How to evaluate LLM capabilities for Code?

Data + AI

Human-Eval

Evaluation Harness for Code Generation

Task
Synthesizing programs from docstrings

Evaluation
Check Functional Correctness
by computing tests passing rate

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]

arXiv:2107.03374v2 [cs.LG] 14 Jul 2021

Abstract

We introduce Codex, a GPT language model fine-tuned on publicly available code from GitHub, and study its Python code-writing capabilities. A distinct production version of Codex powers GitHub Copilot. On HumanEval, a new evaluation set we release to measure functional correctness for synthesizing programs from docstrings, our model solves 28.8% of the problems, while GPT-3 solves 0% and GPT-J solves 11.4%. Furthermore, we find that repeated sampling from the model is a surprisingly effective strategy for producing working solutions to difficult prompts. Using this method, we solve 70.2% of our problems with 100 samples per problem. Careful investigation of our model reveals its limitations, including difficulty with docstrings describing long chains of operations and with binding operations to variables. Finally, we discuss the potential broader impacts of deploying powerful code generation technologies, covering safety, security, and economics.

*Equal contribution
[1]OpenAI, San Francisco, California, USA.
[2]Anthropic AI, San Francisco, California, USA. Work performed while at OpenAI.
[3]Zipline, South San Francisco, California, USA. Work performed while at OpenAI.
Correspondence to: Mark Chen <mark@openai.com>, Jerry Tworek <jt@openai.com>, Heewoo Jun <heewoo@openai.com>, Qiming Yuan <qiming@openai.com>.

1. Introduction

Scalable sequence prediction models (Graves, 2014; Vaswani et al., 2017; Child et al., 2019) have become a general-purpose method for generation and representation learning in many domains, including natural language processing (Mikolov et al., 2013; Sutskever et al., 2014; Dai & Le, 2015; Peters et al., 2018; Radford et al., 2018; Devlin et al., 2018), computer vision (Van Oord et al., 2016; Menick & Kalchbrenner, 2018; Chen et al., 2020; Bao et al., 2021), audio and speech processing (Oord et al., 2016; 2018; Dhariwal et al., 2020; Baevski et al., 2020), biology (Alley et al., 2019; Rives et al., 2021), and even across multiple modalities (Das et al., 2017; Lu et al., 2019; Ramesh et al., 2021; Zellers et al., 2021). More recently, language models have also fueled progress towards the longstanding challenge of program synthesis (Simon, 1963; Manna & Waldinger, 1971), spurred by the presence of code in large datasets (Husain et al., 2019; Gao et al., 2020) and the resulting programming capabilities of language models trained on these datasets (Wang & Komatsuzaki, 2021). Popular language modeling objectives like masked language modeling (Devlin et al., 2018) and span prediction (Raffel et al., 2020) have also been adapted to train their programming counterparts CodeBERT (Feng et al., 2020) and PyMT5 (Clement et al., 2020).

Similarly, our early investigation of GPT-3 (Brown et al., 2020) revealed that it could generate simple programs from Python docstrings. While rudimentary, this capability was exciting because GPT-3 was not explicitly trained for code generation. Given the considerable success of large language models in other modalities and the abundance of publicly available code, we hypothesized that a specialized GPT model, called Codex, could excel at a variety of coding tasks. This paper describes several early Codex models, whose descendants power GitHub Copilot and the Codex models in the OpenAI API.

Data+AI

Coverage-Eval

Evaluation Harness for Coverage Prediction

Task

Predicting code coverage for a given:

- Method
- Test Case

Goal

Evaluate LLM capabilities to understand code execution in terms of coverage

Data+AI

Predicting Code Coverage without Execution

Michele Tufano, Shubham Chandel, Anisha Agarwal, Neel Sundaresan, Colin Clement
Microsoft
Redmond, WA, USA
{mitufano, schandel, anisagarwal, neels, coclement}@microsoft.com

Abstract

Code coverage is a widely used metric for quantifying the extent to which program elements, such as statements or branches, are executed during testing. Calculating code coverage is resource-intensive, requiring code building and execution with additional overhead for the instrumentation. Furthermore, computing coverage of any snippet of code requires the whole program context. Using Machine Learning to amortize this expensive process could lower the cost of code coverage by requiring only the source code context, and the task of code coverage prediction can be a novel benchmark for judging the ability of models to understand code. We propose a novel benchmark task called Code Coverage Prediction for Large Language Models (LLMs). We formalize this task to evaluate the capability of LLMs in understanding code execution by determining which lines of a method are executed by a given test case and inputs. We curate and release a dataset we call COVERAGEEVAL by executing tests and code from the HumanEval dataset and collecting code coverage information. We report the performance of four state-of-the-art LLMs used for code-related tasks, including OpenAI's GPT-4 and GPT-3.5-Turbo, Google's BARD, and Anthropic's Claude, on the Code Coverage Prediction task. Finally, we argue that code coverage as a metric and pre-training data source are valuable for overall LLM performance on software engineering tasks.

1 Introduction

Software testing is an essential part of the software life-cycle which aims at detecting bugs in a program prior to shipping new versions. Code coverage is a widely used metric which estimates the quality of testing, providing some confidence that the system will operate conforming to the specified

Focal Method $\{m\}$

```
public String foo(int x){
    if(x == 0){
        return "zero";
    } else if(x > 0){
        return "positive";
    } else {
        return "negative";
    }
    return "impossible";}
```

Test Case $\{t\}$

```
public void testFoo() {
    String res = foo(2);
    Assert.isEqual("positive", res);}
```

Coverage-Annotated Method $\{\mathrm{cov}(m, t)\}$

```
> public String foo(int x){
>     if(x == 0){
!         return "zero";
>     } else if(x > 0){
>         return "positive";
!     } else {
!         return "negative";
!     }
-     return "impossible";}
```

Figure 1: Given a focal method m, that is a method under test, and a test case t covering that method, the code coverage obtained by t on m can be represented as the coverage-annotated method $\mathrm{cov}(m, t)$, where > represents executed statements, ! represents statements not executed, and - represents unreachable code.

For example, coverage is one of the metrics considered by the Federal Aviation Administration (FAA) for safety certification of avionic equipment, as documented in DO-178B (Johnson, 1998) and DO-178C (Rierson, 2017). Test coverage is also a requirement in the automotive safety standard ISO 26262 Road Vehicles - Functional Safety (Palin et al., 2011).

Given a focal method m, which is executed *directly* by the test case t, code coverage measures the number of statements that have been executed

Coverage-Eval

Evaluation Harness for Coverage Prediction

What is Coverage?

Important metric for quantifying the number of statements and branches executed during testing

How it works?

Requires instrumenting the code, building and monitoring its execution

Potential Benefits

- LLMs that performs well on this task may generate better code.
- LLMs could replace/improve the process of code coverage computation

Data+AI

Focal Method $\{m\}$

```java
public String foo(int x){
    if(x == 0){
        return "zero";
    } else if(x > 0){
        return "positive";
    } else {
        return "negative";
    }
    return "impossible";}
```

Test Case $\{t\}$

```java
public void testFoo() {
    String res = foo(2);
    Assert.isEqual("positive", res);}
```

Coverage-Annotated Method $\{\text{cov}(m, t)\}$

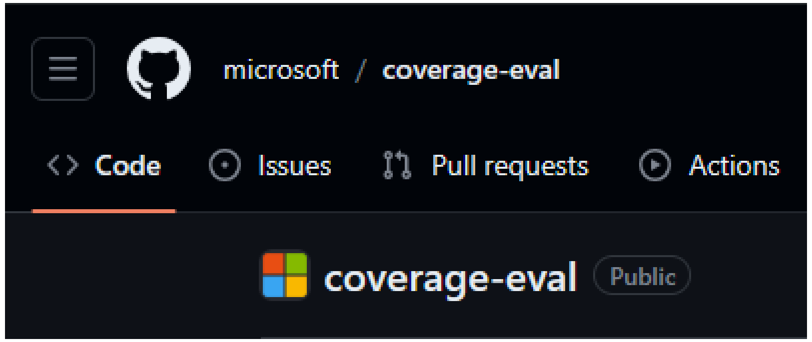
```
> public String foo(int x){
>     if(x == 0){
!         return "zero";
>     } else if(x > 0){
>         return "positive";
!     } else {
!         return "negative";
!     }
-     return "impossible";}
```

Coverage-Eval

Evaluation Harness for Coverage Prediction

Dataset Creation Steps

1. Start with Human-Eval dataset with problems, code solutions, and tests
2. Spit each test case in a single-assert test case (each test now covers less statements/branches)
3. Collect coverage information by running coverage.py
4. Parse and organize the dataset



Problems	Solutions	Tests	Coverage Symbols		
			Executed (>)	Missed (!)	Unreachable (-)
158	164	1160	20037	1734	0

Table 1: COVERAGEEVAL statistics.

Coverage-Eval

Evaluation Harness for Coverage Prediction

Prompting

- Start with a System NL prompt explaining the task
- Mimic a terminal environment
  - Cat code to show it
  - Run coverage computation
- Show zero, one, multiple examples
- Show current focal method and test

Data+AI

System NL Prompt

```
You are a terminal. Instruction:
When user runs:
coverage run -m  pytest code.py

then you'll cat the file code.py,
with each line starting with either of the two symbols below:

> if the line is executed
! is the line is not executed

Example output:
> line1
! line2
> line3
...
> linen

You job is to figure out which line will be executed
given different test cases.
```

Examples

```
(anaconda3-2020.11) | cat code.py
def split_words(txt):
    ...

(anaconda3-2020.11) | cat test.py
def test():
    assert split_words("Hello,world!") == ["Hello","world!"]
    assert True

(anaconda3-2020.11) | coverage run -m  pytest test.py
> def split_words(txt):
>     if " " in txt:
!         return txt.split()
>     elif "," in txt:
>         return txt.replace(',',' ').split()
!     else:
    ...
```

Coverage-Eval

Evaluation Harness for Coverage Prediction

Leaderboard

- **GPT-4** obtains the best results
- All models struggle on branches
- Challenging task for LLMs

Future Work

- Open models like StarCoder or Llama2
- Pretrain model on this task
- Investigate benefits on code generation

Model	zero-shot			one-shot			multi-shot		
	Match	Stmt	Branch	Match	Stmt	Branch	Match	Stmt	Branch
OpenAI GPT-4 (gpt-4)	**25.75**	**84.47**	**20.16**	**22.85**	**90.71**	**22.65**	**30.04**	**90.5**	**22.5**
OpenAI GPT-3.5 (gpt-3.5-turbo)	0	39.87	8.33	8.17	76.53	17.17	11.03	82.29	17.9
Google BARD (text-bison-001)	0	81.27	17.21	1.87	86.93	19.63	21.56	85.66	20.52
Anthropic Claude (claude-1.3)	3.9	84.47	20.07	4.83	83.21	19.16	6.88	55.7	12.23

Table 2: LLMs performances on the Code Coverage Prediction Task. The table reports the percentages of predicted coverage sequences that match the ground truth (Match), the percentage of correct coverage symbols for statements (Stmt), and specifically for branches (Branch). Evaluation performed for zero-shot, one-shot, and multi-shot.

Data+AI

How to evaluate LLM capabilities for Code?

Data + AI

How to evaluate LLM capabilities for Code?

What are other capabilities to evaluate?

Data+AI

Join at menti.com use code 2738 6756

Mentimeter

LLM capabilities for Code

Start Menti

Waiting for responses ...

GO TO
menti.com
ENTER THE CODE
2738 6756

Microsoft

Questions?

Michele Tufano
Sr. Research Scientist
Michele.Tufano@microsoft.com

Data+AI