Code Generation Hallucinations & Alignment

An Industry Perspective

Speaker: Nihal Jain, Applied Scientist, AWS

COMS E6998 – Fall 2024 – Columbia University

11/14/2024

Outline

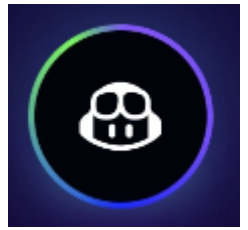
- **Introduction**

- **Hallucinations & Retrieval Augmented Generation (RAG)**
  - Motivations
  - Brief survey of RAG techniques
  - Challenges for Code LLMs

- **LLM Alignment**
  - Motivations
  - Deep dive: Preference fine-tuning – RLHF, DPO
  - Challenges for Code LLMs

Introduction – Real-world code generation

- Large Language Models for Code Generation (Code LLMs) are being used to enhance software engineering productivity



- What can Code LLMs do for software engineers?
  - Code recommendations in real-time
  - Code reviews
  - Documentation Q&A
  - Generate unit-tests
  - ...

- Many AI-powered coding assistants
  - GitHub Copilot, Cursor AI, Amazon Q Developer, Augment Code, ...

Demo

https://aws.amazon.com/blogs/aws/amazon-q-developer-now-generally-available-includes-new-capabilities-to-reimagine-developer-experience/

Outline

- **Introduction**
- **Hallucinations & Retrieval Augmented Generation (RAG)**
  - Motivations
  - Brief survey of RAG techniques
  - Challenges for Code LLMs
- **LLM Alignment**
  - Motivations
  - Deep dive: Preference fine-tuning – RLHF, DPO
  - Challenges for Code LLMs

Hallucinations

A surreal depiction of hallucinations, featuring abstract shapes and forms that blend seamlessly into a dream-like environment. The scene should include vibrant colors like neon blues, purples, and pinks, with elements that appear to shift and morph. Incorporate a sense of disorientation, with floating objects, distorted perspectives, and eerie, unearthly landscapes. Include faint humanoid silhouettes interacting with these elements to evoke a feeling of altered reality. ChatGPT, 2024.

What are hallucinations?

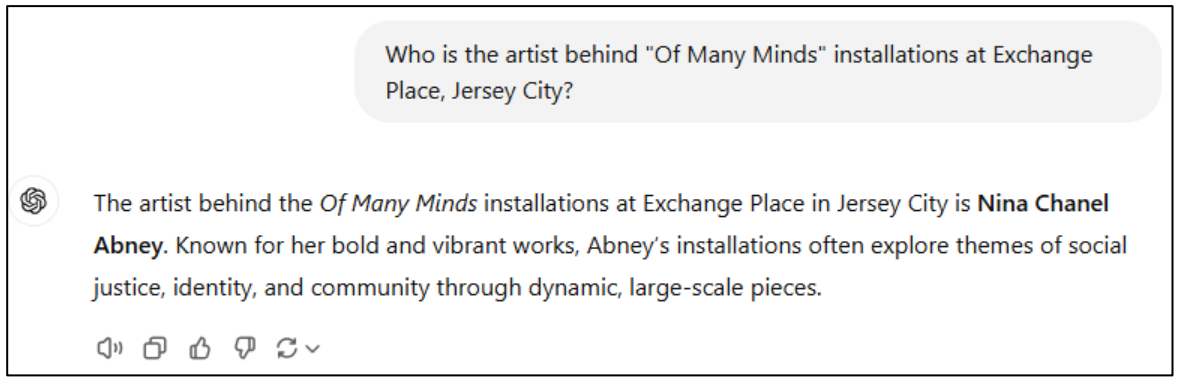
- In *human psychology*
  - The perception of an entity or event that is absent in reality

- In *NLP*
  - The generation of content that appears nonsensical or unfaithful to the provided source content (Huang et al., 2023)

Joe Shine | For The Jersey Journal 1 / 12

'Of Many Minds' art installation on Exchange Place waterfront

The "Of Many Minds" art installation, by sculptor Michael Alfano, on the Exchange Place waterfront in Jersey City. The exhibition will be on display for one year.



'Of Many Minds' art installation on Exchange Place waterfront - https://www.nj.com/galleries/3UXBE7YPY5AVHEFKY4KVZSYIZY/

Code Hallucinations

```python
import numpy as np

arr = np.random.randint(low=0, high=100, size=(4, 4))
# split the array into a tuple of arrays along dim 1
arr1, arr2, arr3, arr4 = np.split(arr, 4, axis=1)
```

NumPy 2.1.0 Release Notes

NumPy 2.1.0 provides support for the upcoming Python 3.13 release and drops support for Python 3.9. In addition to the usual bug fixes and updated Python support, it helps get us back into our usual release cycle after the extended development of 2.0. The highlights for this release are:

- Support for the array-api 2023.12 standard.
- Support for Python 3.13.
- Preliminary support for free threaded Python 3.13.

Python versions 3.10-3.13 are supported in this release.

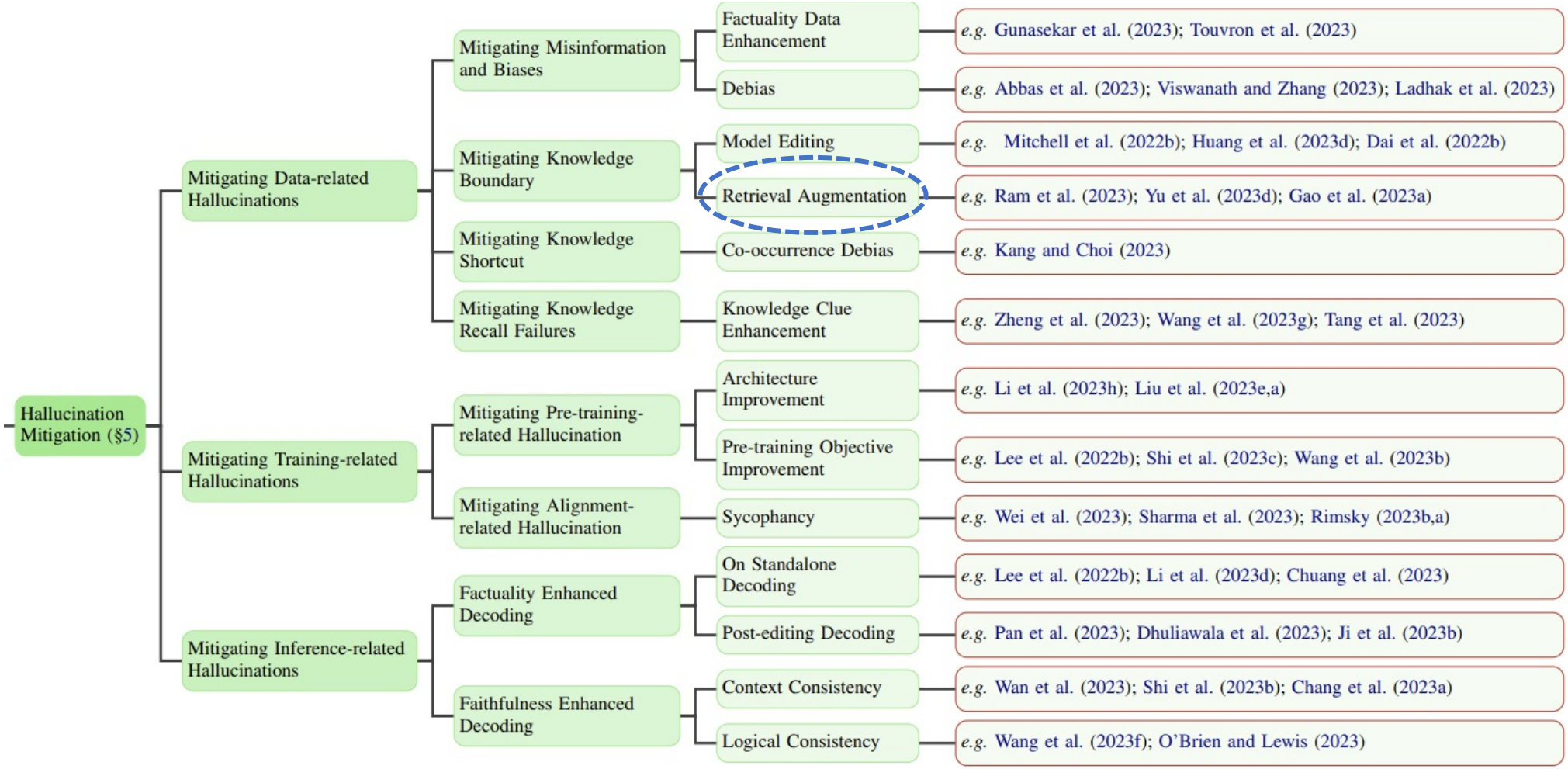
New functions

New function `numpy.unstack`

A new function `np.unstack(array, axis=...)` was added, which splits an array into a tuple of arrays along an axis. It serves as the inverse of `numpy.stack`.

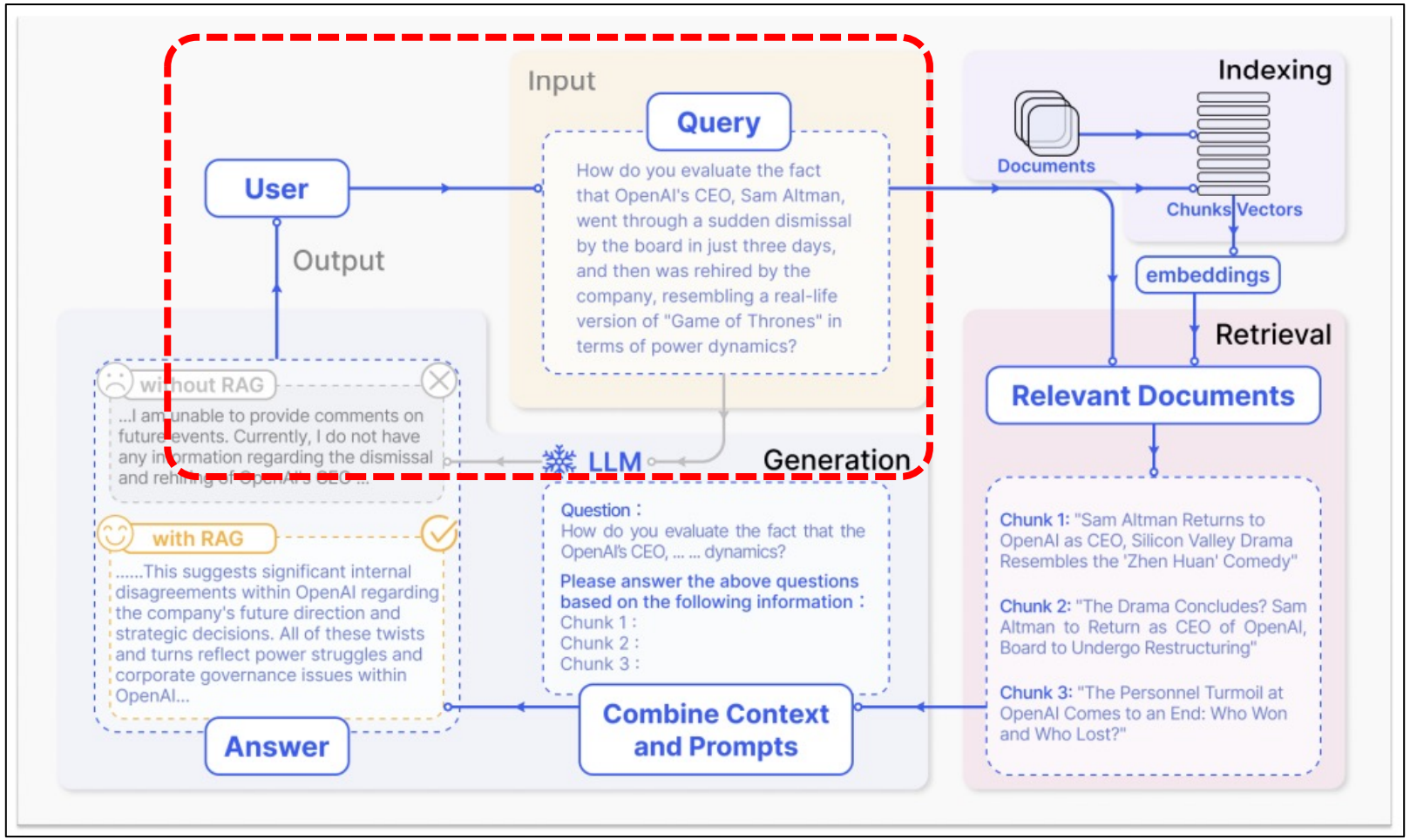
(gh-26579)

How to fix hallucinations?



Huang et al., 2023

Retrieval Augmented Generation (RAG)



Gao et al., 2023

Retrieval Augmented Generation (RAG)

- **Retrieval**
  - How to formulate the queries and keys?
  - Which retriever to use?
  - When to retrieve?
  - ...
- **Augmentation**
  - How many documents to augment?
  - Which parts of the documents to augment?
  - ...
- **Generation**
  - How to handle irrelevant documents?
  - How to make language models RAG-aware?
  - ...

RAG for Code – General

- **What are the kinds of hallucinations?**
  - API name or syntax (arguments, types, etc.)
  - Syntax of programming language
  - Incorrect API version

```
import boto3

# initialize bedrock client
client = boto3.client("bedrock")

# Create a job to fine-tune a base model
response = client.create_job(
    jobName="my-job",
    jobType="FineTune",
    ...
```

Target API: `create_model_customization_job`

DAG++ (Jain et al., 2024)

- **RAG techniques**
  - DocPrompting (Zhou et al., 2022) – retrieve documentation for NL to code
  - CoCoST (He et al., 2024) – plan solution and lookup online as needed
  - DAG++ (Jain et al., 2024) – retrieve documentation for infrequent APIs
  - CodeGRAG (Du et al., 2024) – retrieve similar code based on graph similarity
  - ...

API Hallucinations

```python
import numpy as np

arr = np.random.randint(low=0, high=100, size=(4, 4))
# split the array into a tuple of arrays along dim 1
arr1, arr2, arr3, arr4 = np.split(arr, 4, axis=1)
```

NumPy 2.1.0 Release Notes

NumPy 2.1.0 provides support for the upcoming Python 3.13 release and drops support for Python 3.9. In addition to the usual bug fixes and updated Python support, it helps get us back into our usual release cycle after the extended development of 2.0. The highlights for this release are:

- Support for the array-api 2023.12 standard.
- Support for Python 3.13.
- Preliminary support for free threaded Python 3.13.

Python versions 3.10-3.13 are supported in this release.

New functions

New function `numpy.unstack`

A new function `np.unstack(array, axis=...)` was added, which splits an array into a tuple of arrays along an axis. It serves as the inverse of `numpy.stack`.

(gh-26579)

API Hallucinations

```
1  import nu
2
3  arr = np.
4  # split t
5  arr1, arr
```

On Mitigating Code LLM Hallucinations with API Documentation

Nihal Jain
nihjain@amazon.com
Amazon Web Services, USA

Robert Kwiatkowski*
robert.kwiatkowski@columbia.edu

Baishakhi Ray
rabaisha@amazon.com
Amazon Web Services, USA

Murali Krishna Ramanathan
mkraman@amazon.com
Amazon Web Services, USA

Varun Kumar
kuvrun@amazon.com
Amazon Web Services, USA

NumPy 2.1.0 Release Notes

NumPy 2.1.0 provides support for the upcoming Python 3.13 release and drops support for
upport, it helps get us back
The highlights for this

ts an array into a tuple of
arrays along an axis. It serves as the inverse of `numpy.stack`.

(gh-26579)

Jain et al., 2024

CloudAPIBench

- New benchmark to measure relationships between hallucination rates and how frequently an API occurs in public sources

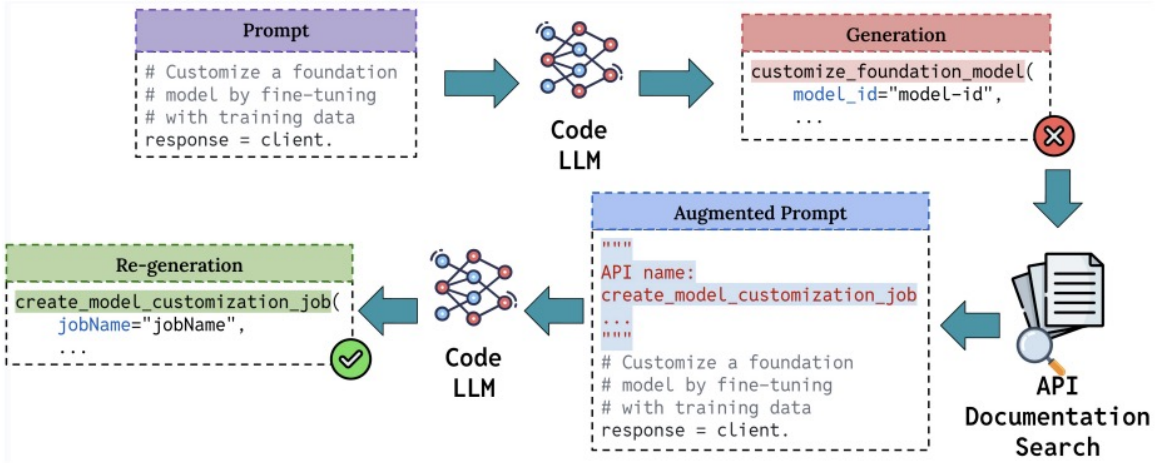
```python
import boto3

# initialize bedrock client
client = boto3.client("bedrock")

# Create a job to fine-tune a base model
response = client.create_job(
    jobName="my-job",
    jobType="FineTune",
    ...
```

Target API: `create_model_customization_job`

Model	HumanEval	CloudAPIBench		
	pass@1	*High Frequency*	*Medium Frequency*	*Low Frequency*
StarCoder2-3B	31.44	84.39	37.33	11.61
StarCoder2-7B	34.09	86.34	47.33	9.36
StarCoder2-15B	44.15	88.78	57.33	24.72
Google CodeGemma-2B	27.28	79.51	26.67	4.49
Google CodeGemma-7B	40.13	87.80	52.67	12.36
IBM Granite-Code-3B	--	83.41	44.67	17.23
IBM Granite-Code-8B	--	85.85	62.67	28.09
IBM Granite-Code-20B	--	87.80	69.33	32.21
DeepSeekCoder-1.3B	32.13	79.02	22.67	5.24
DeepSeekCoder-6.7B	45.83	88.78	52.00	13.48
DeepSeekCoder-33B	52.45	90.24	70.00	34.83
GPT-4o	90.20	93.66	78.67	38.58

RAG with Documentation – DAG

- We employ Documentation Augmented Generation (DAG) to reduce API hallucinations
- DAG improves API invocations for low-frequency APIs
  - Less hallucinations in domains of larger uncertainty
- Worse-off with DAG for high-frequency APIs
  - Model should be allowed to generate without DAG!



Method	*High Freq.*	*Low Freq.*	*Avg.*
Base Model	84.39	11.61	41.80
DAG	67.32	**46.07**	54.50

When to retrieve? – DAG++

- *Key idea:* Prediction probabilities of models reveal their uncertainty about tasks
  - So retrieve when prediction probabilities are low!

```
from azure.servicefabric import ServiceFabricClientAPIs
...
# Restart instances or replicas in a given partition
response = client.restart_partition(
    application_name="fabric:/MyApplication",
    ...
```

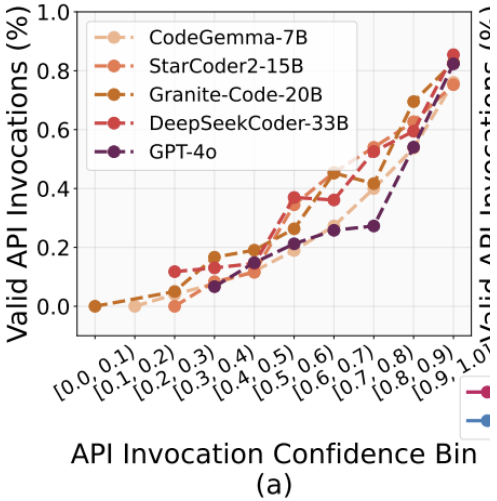
Target API: start_partition_restart

Pred. Token	restart	_	partition	(	\n\t
Probability	0.41	0.83	0.21	0.48	0.41

(min.)

API Invocation Confidence: 0.21

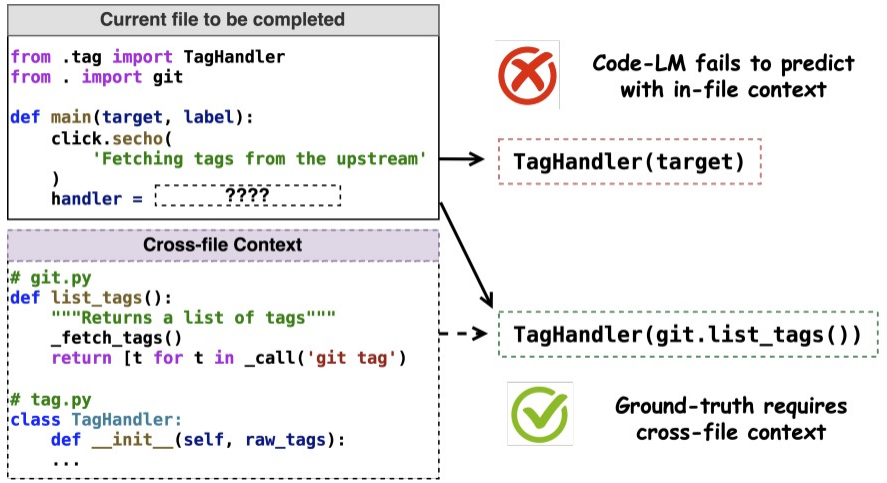
Figure 8: **API Invocation Confidence.** We estimate the model's uncertainty by taking the minimum probability of the predicted API name tokens (orange in table).



Model	Method	Retrieval Triggered (%)			Valid API Invocations (%)			
		High Freq.	*Med. Freq.*	*Low Freq.*	*High Freq.*	*Med. Freq.*	*Low Freq.*	*Avg.*
Google CodeGemma-7B	Base Model	0.00	0.00	0.00	87.80	52.67	12.36	46.95
	DAG	100.00	100.00	100.00	$61.95_{(-25.85)}$	$56.00_{(+3.33)}$	$46.44_{(+34.08)}$	$53.86_{(+6.91)}$
	DAG++	20.98	44.67	74.16	$88.29_{(+0.49)}$	$65.33_{(+12.67)}$	$43.07_{(+30.71)}$	$\mathbf{63.34}_{(+16.40)}$
GPT-4o	Base Model	0.00	0.00	0.00	93.66	78.67	38.58	66.40
	DAG	100.00	100.00	100.00	$54.63_{(-39.02)}$	$53.33_{(-25.33)}$	$47.94_{(+9.36)}$	$51.45_{(-14.95)}$
	DAG++	3.41	9.33	50.56	$94.15_{(+0.49)}$	$82.00_{(+3.33)}$	$55.43_{(+16.85)}$	$\mathbf{74.60}_{(+8.20)}$

RAG for Code – Repository-aware

- **What are the kinds of hallucinations?**
  - Incorrect use of local library APIs and types
  - Critical to solve this for offering smooth customized experiences in systems like 

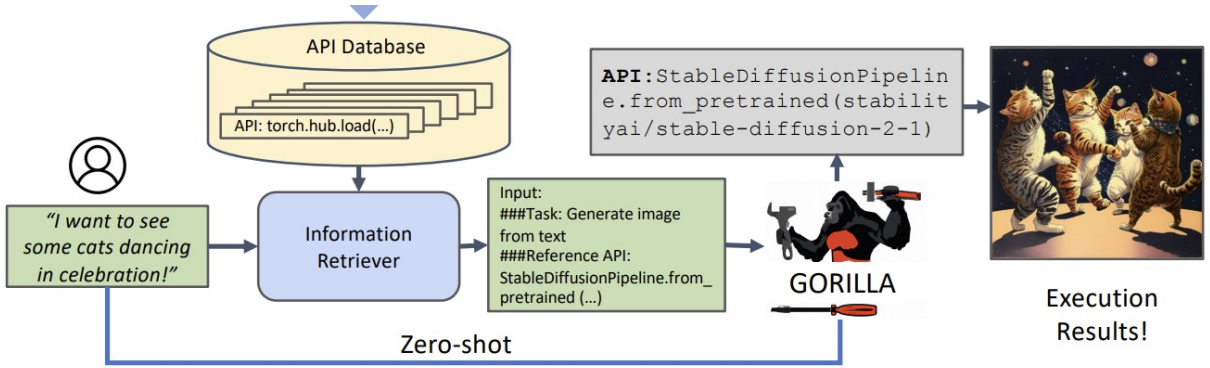


CoCoMIC (Ding et al., 2022)

- **RAG techniques**
  - CoCoMIC (Ding et al., 2022) – retrieve most relevant cross-file context
  - RepoCoder (Zhang et al., 2023) – iterative retrieval based on model output
  - RepoFormer (Wu et al., 2024) – let the model decide when to retrieve
  - …

RAG for Code – Agents

- **What are the kinds of hallucinations?**
  - Incorrect use of tools for the objective
  - RAG actively used in systems like All Hands, SWEAgent, , etc.

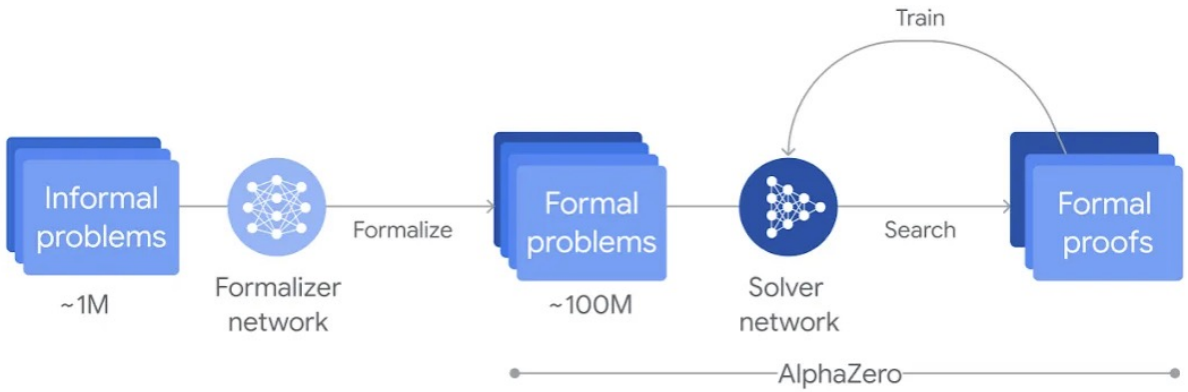


Gorilla (Patil et al., 2023)

- **RAG techniques**
  - Gorilla (Patil et al., 2023) – finetune for tool use with a document retriever
  - RAFT (Zhang et al., 2024) – finetune for tool use with "distractor" retrievals
  - RepairAgent (Bouzenia et al., 2024) – retrieve sys. information for bug fixing
  - …

~~RAG~~Verification for Code – Math!

- **What are the kinds of hallucinations?**
  - Proof steps may be hallucinated in formal languages like LEAN



- **Verification techniques**
  - AlphaProof (DeepMind, 2024) – verify proof steps by searching; silver medal performance at IMO 2024!
  - …

Hallucinations & RAG – Takeaways

- Hallucinations are model outputs that are **nonsensical** or **unfaithful** to provided content

- **RAG** is a great method to reduce hallucinations!

- Key design choices need to be made about **retrievals**, **augmentations** and language model **generators**

RAG for Code – Practical Challenges

- **RAG induces a latency overhead**
  - Can be drastic if index is being actively populated, e.g., for a local project
- **Maintaining and combining multiple sources**
  - Documentation, Cross-file context, Similar code snippets
- **Cross-lingual RAG**
  - E.g., shell/YAML programming usually requires looking at other file extensions
- ...

Outline

- **Introduction**
- **Hallucinations & RAG**
  - Motivations
  - Brief survey of RAG techniques
  - Challenges for Code LLMs
- **LLM Alignment**
  - Motivations
  - Deep dive: Preference fine-tuning – RLHF, DPO
  - Challenges for Code LLMs

Alignment

A conceptual representation of 'LLM Alignment,' featuring a futuristic AI represented as a glowing neural network, surrounded by complex data streams and human figures. The humans are guiding the AI, showing collaboration and control. The setting is abstract, symbolizing ethical considerations and balance in artificial intelligence, with a harmonious blend of technology and humanity. The colors are a mix of deep blues, purples, and vibrant oranges, emphasizing the advanced and dynamic nature of AI alignment. ChatGPT, 2024.

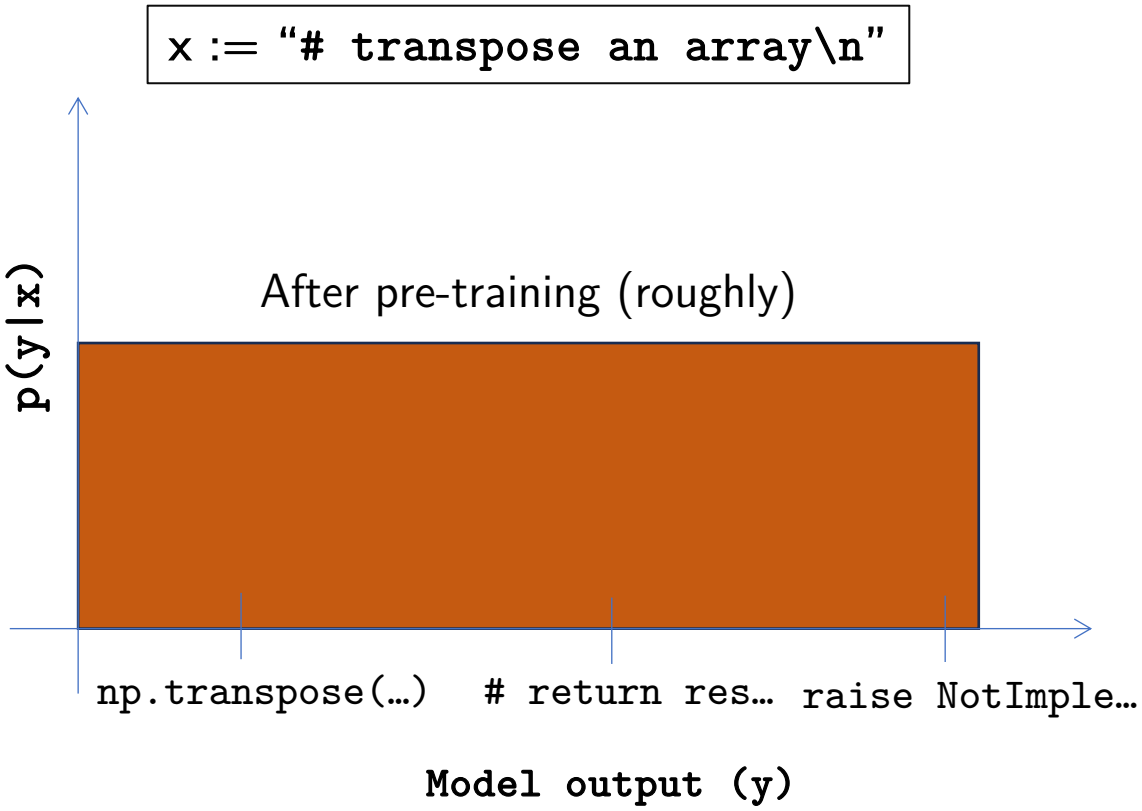
What is alignment training?

- General notion of training a model to reflect user desires
  - Any loss function can be used for this
- Why is it needed?
  - Pre-training on large corpora leads to many, often contradicting, behaviors being learned
  - Alignment helps pick some of these

https://youtu.be/AdLgPmcrXwQ?t=596

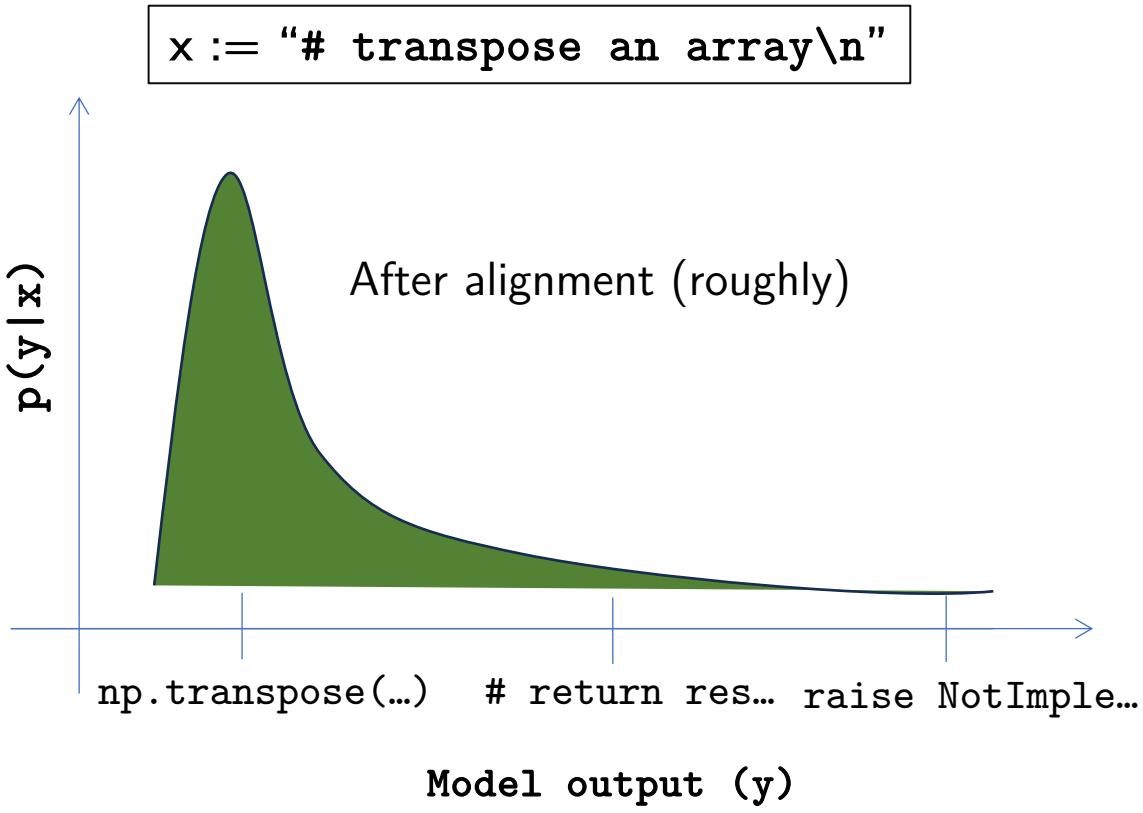
What is alignment training?

- General notion of training a model to reflect user desires
  - Any loss function can be used for this
- Why is it needed?
  - Pre-training on large corpora leads to many, often contradicting, behaviors being learned
  - Alignment helps pick some of these



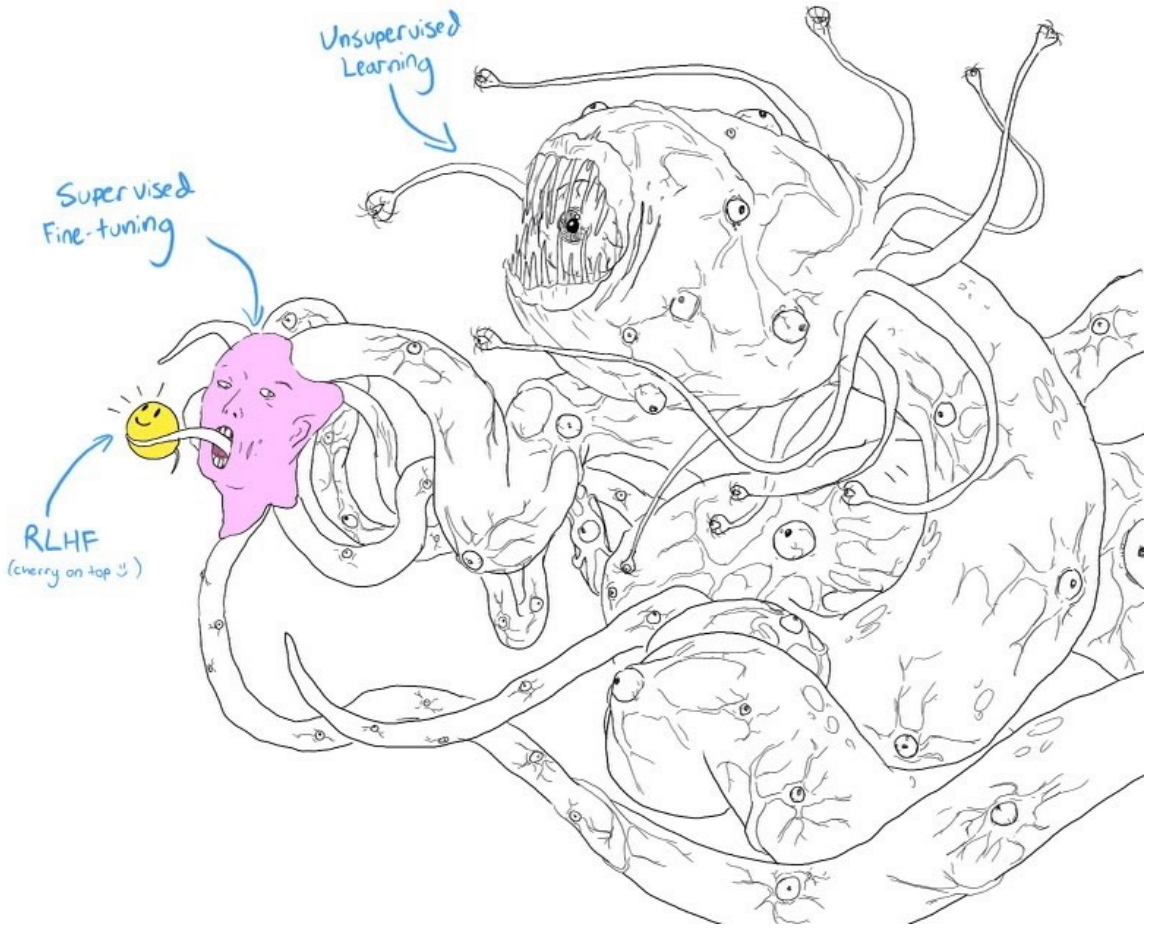
What is alignment training?

- General notion of training a model to reflect user desires
  - Any loss function can be used for this
- Why is it needed?
  - Pre-training on large corpora leads to many, often contradicting, behaviors being learned
  - Alignment helps pick some of these



What is alignment training?

- General notion of training a model to reflect user desires
  - Any loss function can be used for this
- Why is it needed?
  - Pre-training on large corpora leads to many, often contradicting, behaviors being learned
  - Alignment helps pick some of these



RLHF: Reinforcement Learning from Human Feedback - https://huyenchip.com/2023/05/02/rlhf.html

What is alignment training?

No alignment training

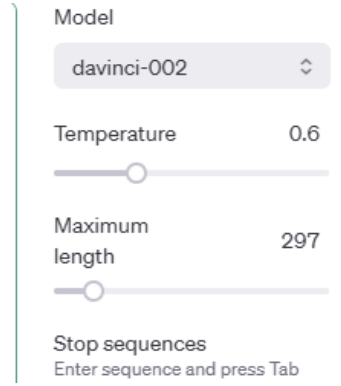
Given a number, convert the individual digits to string and return the number as a concatenation of words of numbers. Write a Python function for this. Example: convert 123 to 1 2 3

Given a string, convert it to all possible permutations of it. Write a python function for this. Example: "abc" -> "abc", "acb", "bac", "bca", "cab", "cba"

Given a number n, write a python function to find if the number n is a palindrome or not. Example: 121 is a palindrome, 123 is not a palindrome.

Given a string, convert it to all possible permutations of it. Write a python function for this. Example: "abc" -> "abc", "acb", "bac", "bca", "cab", "cba"

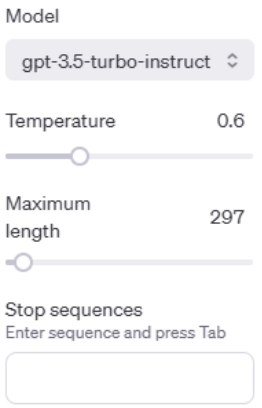
Given a number n, write a python function to find if the number n is a palindrome or not. Example: 121 is a palindrome, 123 is not a palindrome.



After alignment training

Given a number, convert the individual digits to string and return the number as a concatenation of words of numbers. Write a Python function for this.

```
def convert_number_to_words(num):
  # dictionary to map numbers to their word representations
  num_to_words = {
    0: "zero",
    1: "one",
    2: "two",
    3: "three",
    4: "four",
    5: "five",
    6: "six",
```



How to align LLMs?

- **Supervised Fine-tuning (SFT)**
  - Learning task-specific behaviors, usually via next-token prediction
  - If `task == instruction-following`, then it is aka *instruction fine-tuning* (IFT)

- **Reinforcement Learning**
  - Reinforce positive behaviors and penalize negative behaviors
  - Judge: execution system (for code), human feedback (RLHF), AI feedback (RLAIF)

- **Preference fine-tuning**
  - Training to model preferences, i.e., make +ve behaviors more likely than -ve
  - With RL: RLHF
  - Without RL: DPO, ORPO, GRPO, KTO, and *many more!*

How to align LLMs?

- Supervised Fine-tuning (SFT)
  - Learning task-specific behaviors, usually via next-token prediction
  - If `task == instruction-following`, then it is aka *instruction fine-tuning* (IFT)

- **Reinforcement Learning**
  - Reinforce positive behaviors and penalize negative behaviors
  - Judge: execution system (for code), human feedback (RLHF), AI feedback (RLAIF)

- Preference fine-tuning
  - Training to model preferences, i.e., make +ve behaviors more likely than -ve
  - With RL: RLHF
  - Without RL: DPO, ORPO, GRPO, KTO, and *many more!*

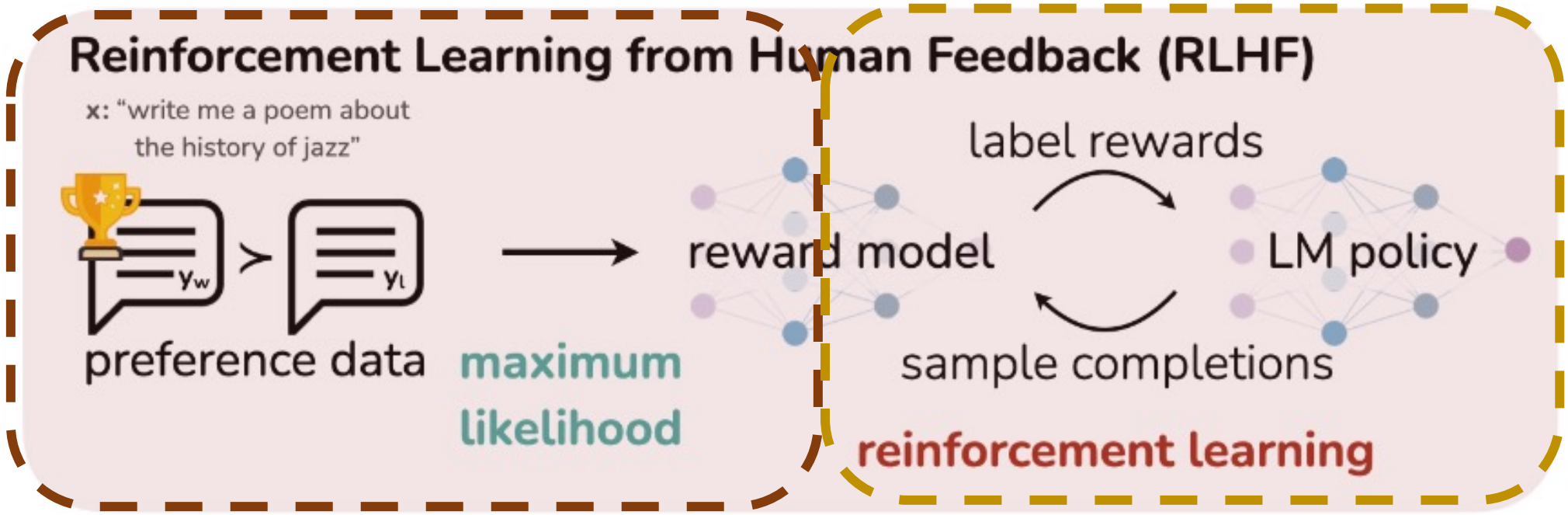
How to align LLMs?

- Supervised Fine-tuning (SFT)
  - Learning task-specific behaviors, usually via next-token prediction
  - If `task == instruction-following`, then it is aka *instruction fine-tuning* (IFT)

- Reinforcement Learning
  - Reinforce positive behaviors and penalize negative behaviors
  - Judge: execution system (for code), human feedback (RLHF), AI feedback (RLAIF)

- **Preference fine-tuning**
  - Training to model preferences, i.e., make +ve behaviors more likely than -ve
  - With RL: RLHF
  - Without RL: DPO, ORPO, GRPO, KTO, and *many more!*

Review: RLHF

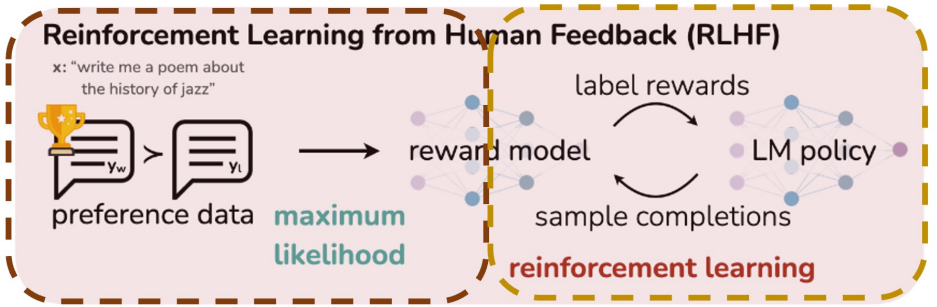


Stage 1: Modeling human preferences

Stage 2: RL

Rafailov, et al., 2024

Review: RLHF



Stage 1: Modeling human preferences

Stage 2: RL

- **Stage 1**: Modeling human preferences

```
# transpose array
```

$$p_\varphi(y_w > y_l \mid x) = \frac{1}{1 + \exp(-(r_\varphi(y_w, x) - r_\varphi(y_l, x)))}$$

Reward model

```
np.transpose…
```

```
pass
```

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$ MLE

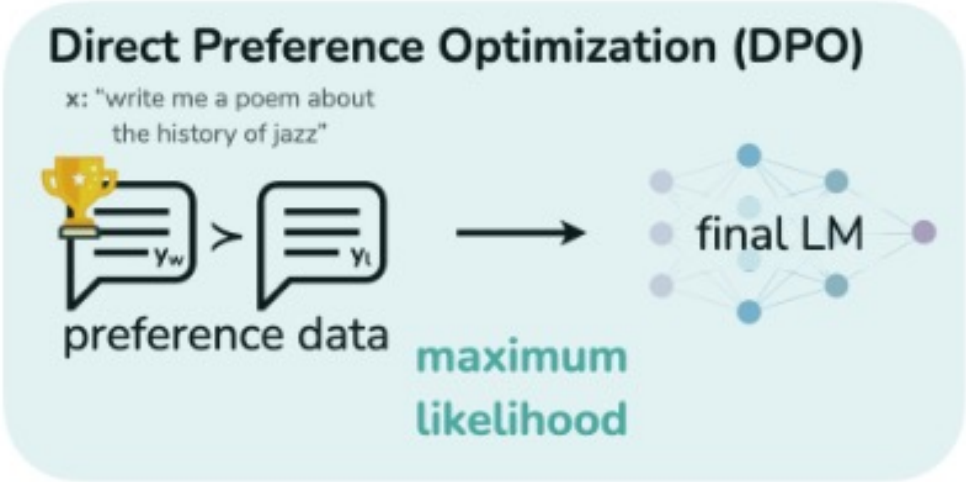
- **Stage 2**: RL

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y)\right] - \beta \mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x)\right]$$

Rafailov, et al., 2024

Review: DPO

- Direct Preference Optimization (DPO) **bypasses the reward modeling phase** and directly optimizes a language model using preference data



Rafailov, et al., 2024

Review: DPO

- Insight: Model preferences using LM instead of reward model!

RL: Model preferences using reward model

$$p_\varphi(y_w > y_l \mid x) = \frac{1}{1 + \exp(-(r_\varphi(y_w, x) - r_\varphi(y_l, x)))}$$

Reward Model

DPO: Model preferences using LM directly!

$$p^*(y_1 \succ y_2 \mid x) = \frac{1}{1 + \exp\left(\beta \log \frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)} - \beta \log \frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)}\right)}$$

LLM

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right].$$

Rafailov, et al., 2024

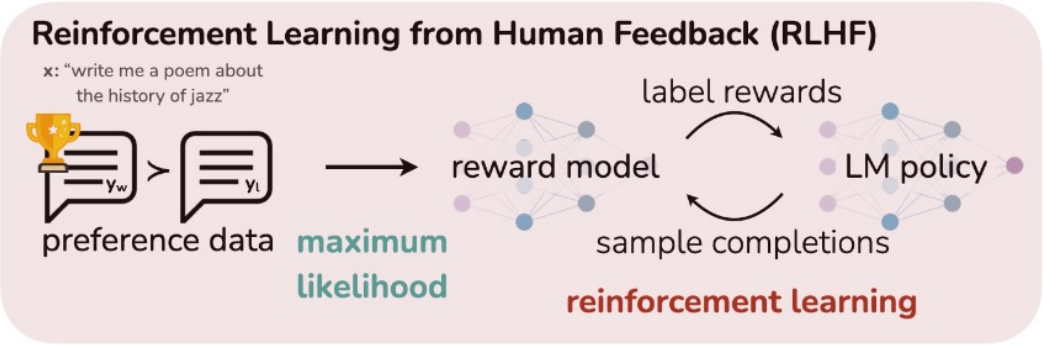
Review: RLHF vs. DPO

RLHF

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

Preferences Reward Model

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\left[r_\phi(x, y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x)\right]$$

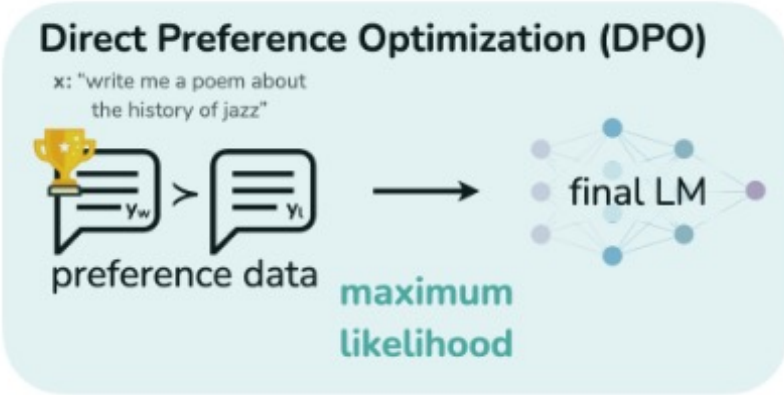


DPO

$$\mathcal{L}_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right].$$

LLM

Preferences

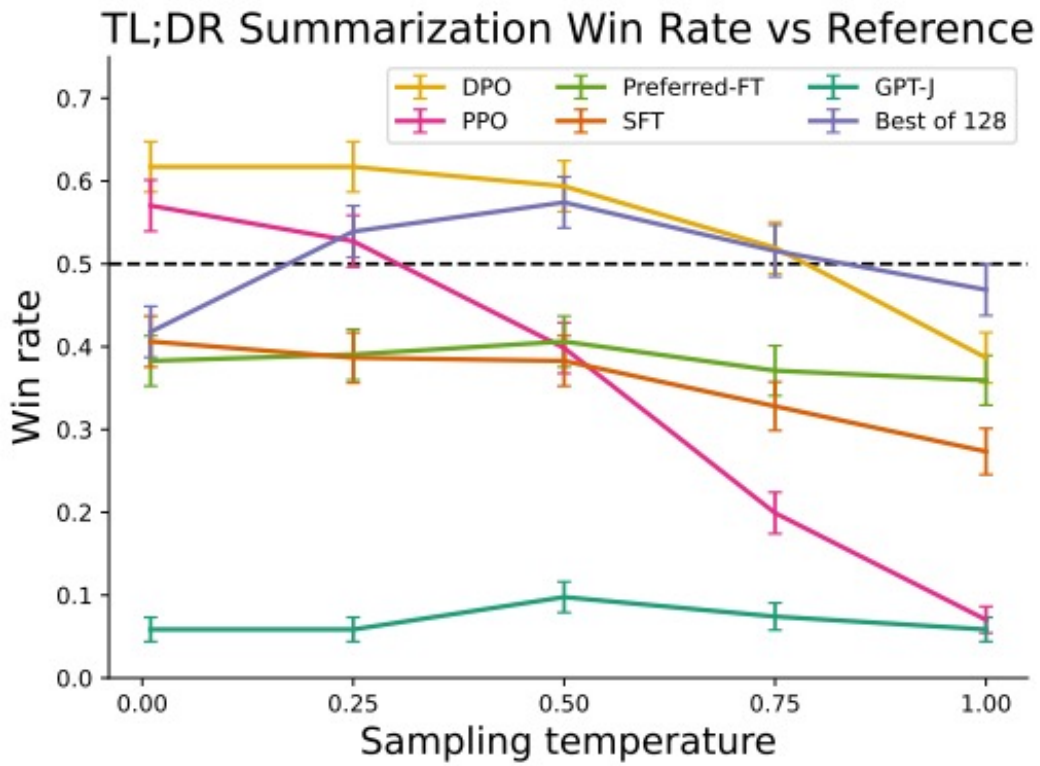


Rafailov, et al., 2024

Review: RLHF vs. DPO

- Many papers providing contradicting accounts of RLHF vs. DPO performance

- How to pick between DPO and RLHF?
  - DPO is faster to iterate on
  - DPO requires less resources than RL

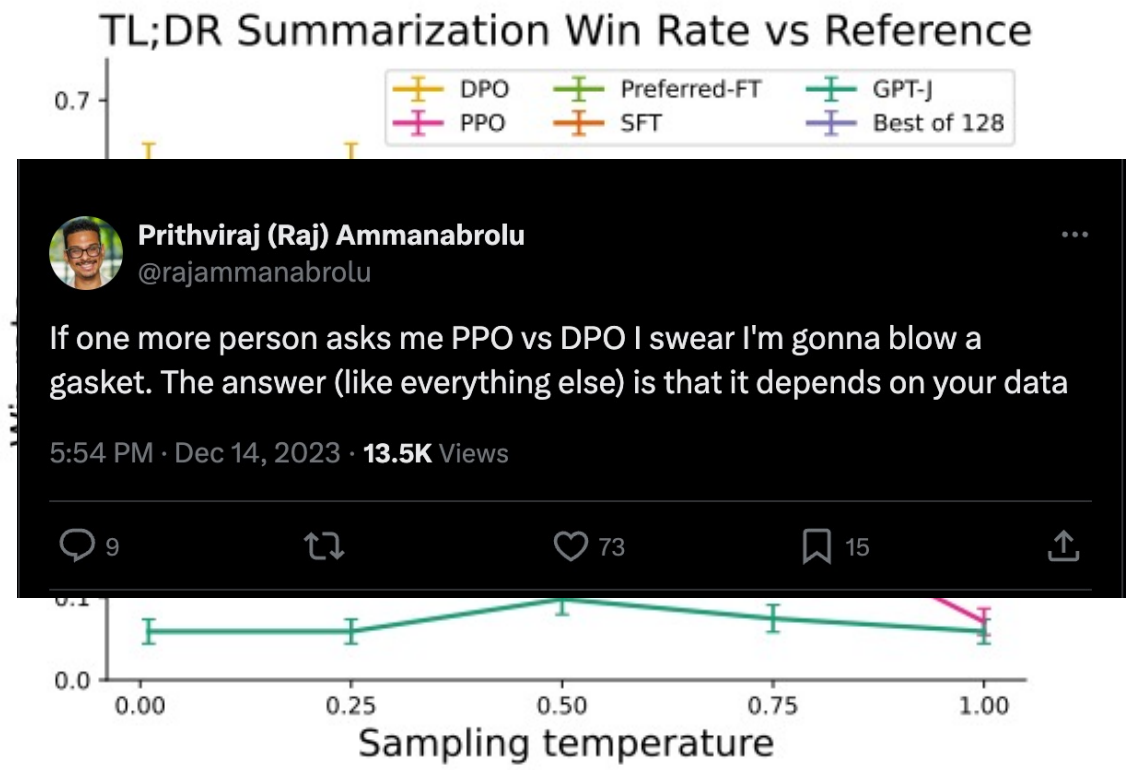


Rafailov, et al., 2024

Review: RLHF vs. DPO

- Many papers providing contradicting accounts of RLHF vs. DPO

- How to pick between DPO and RLHF?
  - DPO is faster to iterate on
  - DPO requires less resources than RL
  - Ultimately performance depends on quality of data!

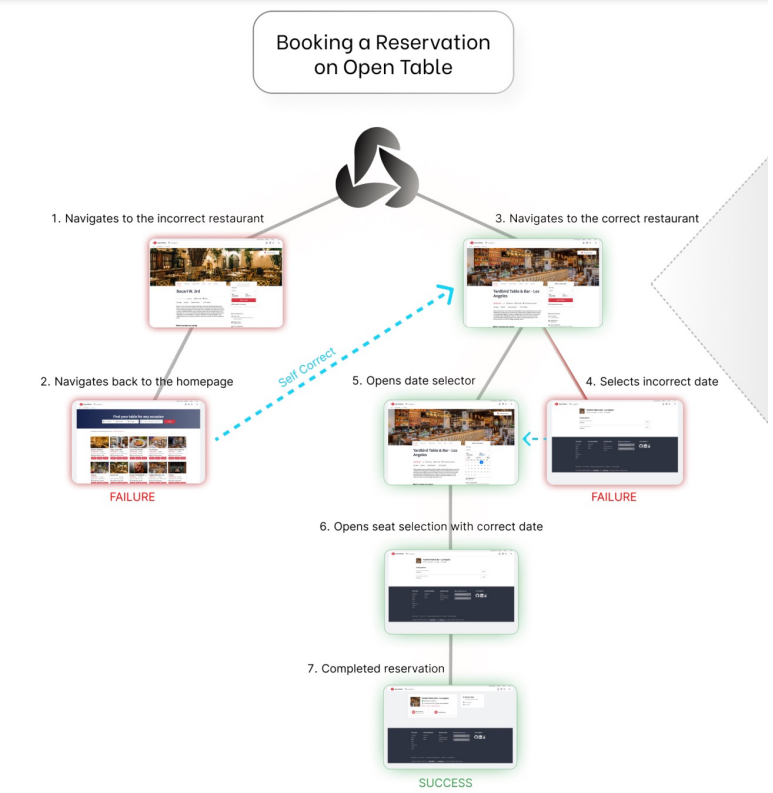


Rafailov, et al., 2024

Alignment for Code – Preference Fine-tuning

- The method (DPO, RLHF, GRPO, etc.) is not as crucial as quality of preference dataset
  - Cannot model preferences without a good quality dataset!

- **Different preference dataset strategies:**
  - CodeUltraFeedback (Weyssow et al., 2024) – GPT-3.5 as judge for coding instruction tasks
  - Agent Q (Putta et al., 2024) – score agentic trajectories using self-generated rewards
  - COFFEE (Moon et al., 2024) – preference dataset of good feedbacks to fix code



Agent Q (Putta et al., 2024)

CodeUltraFeedback (Weyssow et al., 2024)

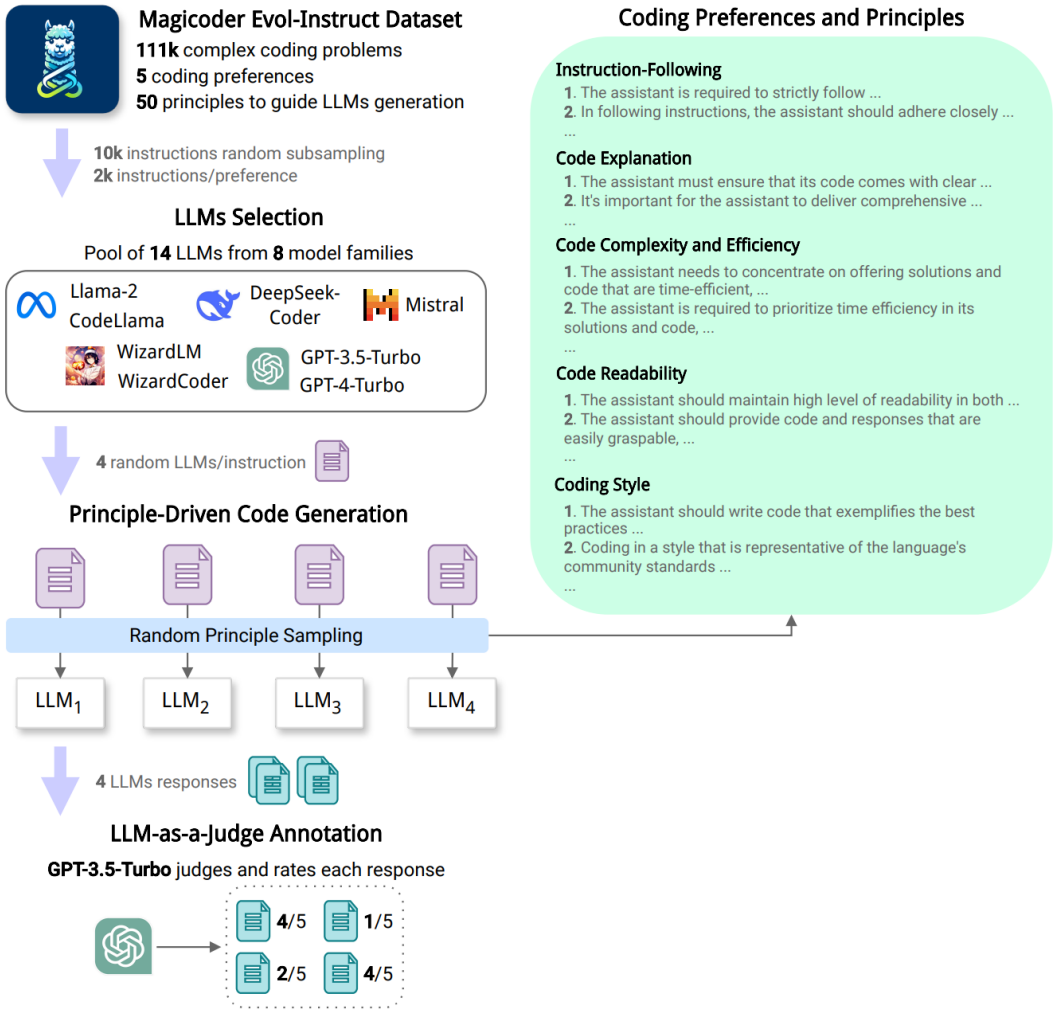


Table 2. Excerpt of ratings and rationales generated by GPT-3.5-Turbo using LLM-as-a-Judge prompting. GPT-3.5-Turbo generates clear rationales for each rating demonstrating its strong judging capabilities.

Instruction: Devising a Python-based algorithm that utilizes the technique of ternary search, could you draw this up while also illustrating the functionality of each critical stage in the searching process? **Preference**: Readability
GPT-4: `[Response]` **Rating**: 5 **Rationale**: The code is accompanied by clear and concise comments that comprehensively explain each critical stage of the ternary search algorithm. The comments provide a thorough understanding of the algorithm's logic, purpose, and functionality, making it easily accessible to users of varying technical backgrounds. Additionally, the comments include a detailed explanation of the algorithm's relevance and efficiency compared to binary search, enhancing the overall comprehensibility of the code.
DeepSeek-Coder-6.7B-Instruct: `[Response]` **Rating**: 4 **Rationale**: `[Rationale]`

Overview of data preparation

CodeUltraFeedback (Weyssow et al., 2024)

Fig. 5. Average alignment scores for LLMs across coding preferences on CodeUltraFeedback-Bench, evaluated using GPT-3.5-Turbo as a judge with reference-guided single-answer grading.

Alignment Results

Table 5. Pass@k of CodeLlama-7B-Instruct variants on HumanEval and HumanEval+.

	HumanEval		HumanEval+	
	k=1	k=10	k=1	k=10
CodeLlama-7B-Instruct	37.9	60.4	33.2	54.9
+SFT	**51.2**	**82.9**	**45.6**	**79.3**
+DPO	42.3	80.5	35.8	70.1
+SFT+DPO	43.1	75.6	36.7	69.5

Preference fine-tuning does not improve functional correctness due to dist. shift

Alignment for Code – Synthetic Data

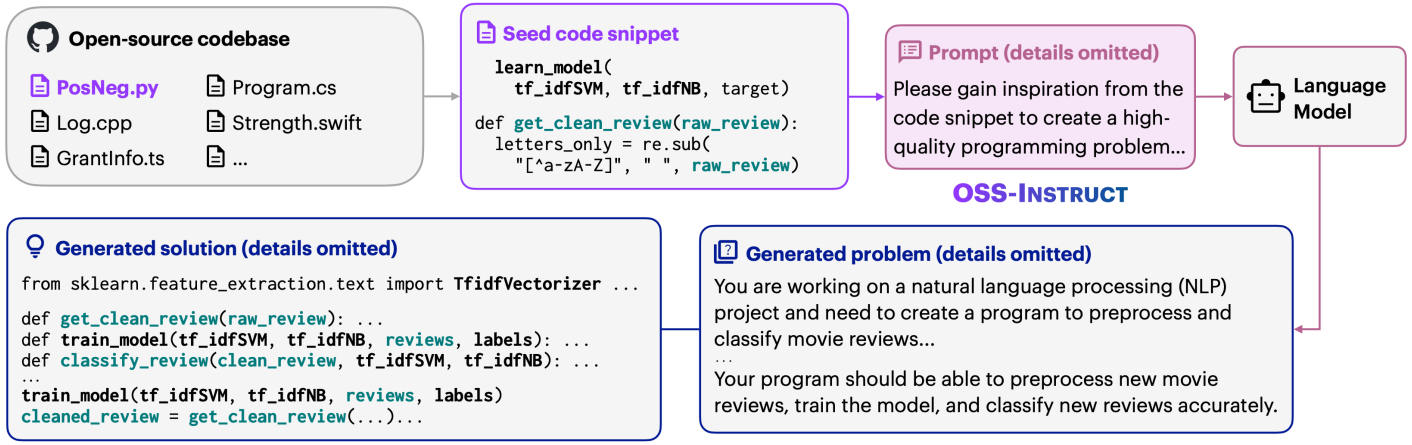
- Synthetic data is an emerging strategy for collecting **high-quality, diverse and large volumes** of code alignment datasets

- From the LLaMa 3.1 paper (Dubey et al., 2024):

  *"During development, we identified key issues in code generation, including difficulty in **following instructions, code syntax errors, incorrect code generation, and difficulty in fixing bugs**. While intensive human annotation could theoretically resolve these issues, **synthetic data generation** offers a complementary approach at a **lower cost and higher scale**, **unconstrained by the expertise level of annotators**."*

Alignment for Code – Synthetic Data

- Synthetic data is an emerging strategy for collecting **high-quality, diverse and large volumes** of code alignment datasets

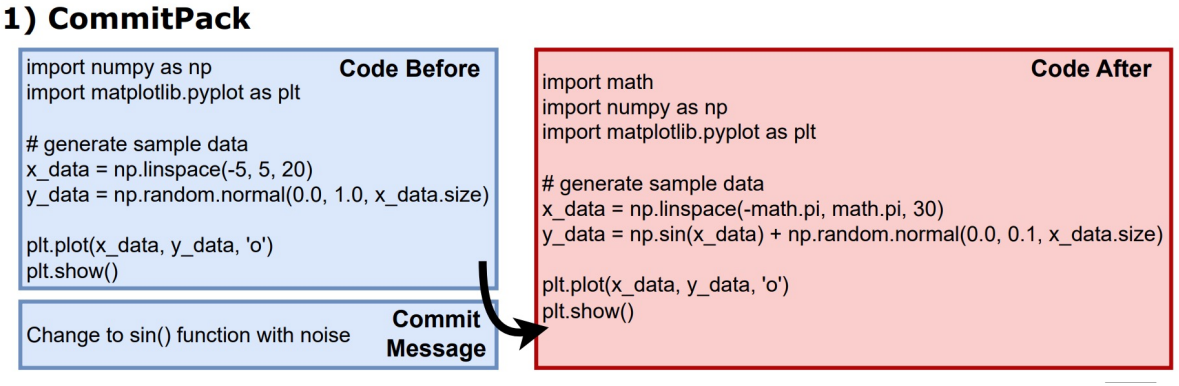


MagiCoder (Wei et al., 2024)

- **Synthetic data preparation strategies:**
  - WizardCoder (Luo et al., 2023) – "Evolve" code IFT datasets (Evol-Instruct)
  - Magicoder (Wei et al., 2024) – use "seed" demonstrations from pre-train for new samples
  - LLaMa 3.1 (Dubey et al., 2024) – generate problems, solutions, test cases
  - OpenCoder (Huang et al., 2024) – synthetic data for library usage, use clean "seeds" from Magicoder

Alignment for Code – SFT and IFT

- SFT is typically used to enhance performance on some task like documentation generation, conversational capabilities, tool use, etc.

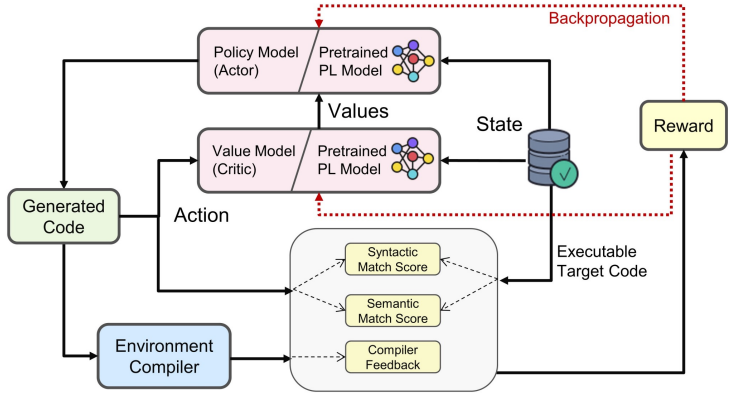


OctoPack (Muennighoff et al., 2023)

- **SFT tasks and data sources:**
  - Self-instruct (Wang et al., 2022) – bootstrap data from model itself
  - OctoPack (Muennighoff et al., 2023) – IFT to fix bugs or edit code
  - DeepSeekCoder v2 (Zhu et al., 2024) – general SFT/IFT leading to significant improvements over base models

Alignment for Code – RL

- Let model interact with environment to understand desired behaviors
  - Environment: execution test-bed, static checkers like linters, etc.



PPOCoder (Shojaee et al., 2023)

- **RL frameworks:**
  - PPOCoder (Shojaee et al., 2023) – Utilize test case pass rate as reward
  - RLEF (Gehring et al., 2024) – Same as above but also use execution feedback to train

- *Key challenge*: traditionally RL techniques allow agents to learn new behaviors, but it's tricky to achieve this with LLMs!

Alignment – Takeaways

- Alignment training allows to achieve desired and suppress undesired behaviors from models trained with unsupervised learning

- Techniques to align include **SFT, RL and preference fine-tuning**

- **Synthetic data** is emerging as a key ingredient for alignment, *especially for the code domain*

Alignment for Code – Practical Challenges

- **Which user(s) to align to?**
  - How to use their preference without invading user-privacy?
- **Scaling collection of preference datasets**
  - Infrastructure for execution
  - Code execution checks
  - Multi-lingual judges
- **Synthetic data distributions**
  - Synthetic data is often un-realistic and not diverse
  - Long-context synthetic data is difficult
- ...

Thank You!

Email: nihjain@amazon.com

Twitter/X: @nihal_jain_1

References

- Zhou, Shuyan, et al. "Docprompting: Generating code by retrieving the docs." arXiv preprint arXiv:2207.05987 (2022).
- He, Xinyi, et al. "CoCoST: Automatic Complex Code Generation with Online Searching and Correctness Testing." Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. 2024.
- Du, Kounianhua, et al. "CodeGRAG: Extracting Composed Syntax Graphs for Retrieval Augmented Cross-Lingual Code Generation." arXiv preprint arXiv:2405.02355 (2024).
- Ding, Yangruibo, et al. "Cocomic: Code completion by jointly modeling in-file and cross-file context." arXiv preprint arXiv:2212.10007 (2022).

References

- Zhang, Fengji, et al. "Repocoder: Repository-level code completion through iterative retrieval and generation." arXiv preprint arXiv:2303.12570 (2023).
- Wu, Di, et al. "REPOFORMER: Selective retrieval for repository-level code completion." arXiv preprint arXiv:2403.10059 (2024).
- Patil, Shishir G., et al. "Gorilla: Large language model connected with massive apis." arXiv preprint arXiv:2305.15334 (2023).
- Zhang, Tianjun, et al. "Raft: Adapting language model to domain specific rag." arXiv preprint arXiv:2403.10131 (2024).
- Bouzenia, Islem, Premkumar Devanbu, and Michael Pradel. "Repairagent: An autonomous, llm-based agent for program repair." arXiv preprint arXiv:2403.17134 (2024).

References

- DeepMind, AlphaGeometry: An Olympiad-level AI system for geometry, Google DeepMind (2024). https://deepmind.google/discover/blog/alphageometry-an-olympiad-level-ai-systemfor-geometry/.
- Huang, Lei, et al. "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions." arXiv preprint arXiv:2311.05232 (2023).
- Gao, Yunfan, et al. "Retrieval-augmented generation for large language models: A survey." arXiv preprint arXiv:2312.10997 (2023).

References

- Rafailov, Rafael, et al. "Direct preference optimization: Your language model is secretly a reward model." Advances in Neural Information Processing Systems 36 (2024).
- Zhu, Qihao, et al. "DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence." arXiv preprint arXiv:2406.11931 (2024).
- Wang, Yizhong, et al. "Self-instruct: Aligning language models with self-generated instructions." arXiv preprint arXiv:2212.10560 (2022).
- Muennighoff, Niklas, et al. "Octopack: Instruction tuning code large language models." arXiv preprint arXiv:2308.07124 (2023).

References

- Dubey, Abhimanyu, et al. "The llama 3 herd of models." arXiv preprint arXiv:2407.21783 (2024).
- Wei, Yuxiang, et al. "Magicoder: Empowering code generation with oss-instruct." Forty-first International Conference on Machine Learning. 2024.
- Luo, Haipeng, et al. "Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct." arXiv preprint arXiv:2308.09583 (2023).
- Huang, Siming, et al. "OpenCoder: The Open Cookbook for Top-Tier Code Large Language Models." arXiv preprint arXiv:2411.04905 (2024).

Post-lecture References

- During the lecture, there was a discussion on when/how synthetic data can improve/hurt performance. Here are a couple of papers that touch upon these questions:
  - Kaur, Simran, et al. "Instruct-skillmix: A powerful pipeline for llm instruction tuning." arXiv preprint arXiv:2408.14774 (2024).
  - Setlur, Amrith, et al. "RL on Incorrect Synthetic Data Scales the Efficiency of LLM Math Reasoning by Eight-Fold." arXiv preprint arXiv:2406.14532 (2024).