

DSPy: Not Your Average Prompt Engineering

OUR MISSION

Jina AI provides **the search foundation (SF)**, a new frontier in neural information retrieval.

SF includes embeddings, rerankers, prompt optimizers, and core infra. They work in concert to revolutionize how we utilize data.

OUR EXCELLENCE

2020

FOUNDED IN

BERLIN

OUR HQ

38

EMPLOYEES

30+

OPENSOURCE PROJECTS

1,000+

DEVELOPERS EMPOWERED

400,000+

TOTAL USERS REGISTERED

OUR INVESTORS

Canaan
PARTNERS

GGVCAPITAL



MANGO
CAPITAL

SAP.iO

SAPPHIRE
VENTURES

OUR PARTNERS

Microsoft

aws

Google Cloud

SAP

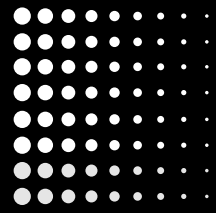
mongoDB

NVIDIA

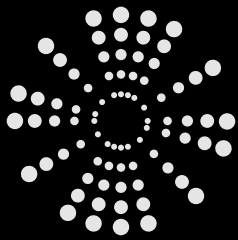
intel



WHAT IS **S**EARCH **F**OUNDATION



Embeddings are the cornerstones of modern search system, representing multimodal data into vectors of numbers. This process enables a more nuanced and contextual understanding of content, far beyond simple keyword matching.



Rerankers take the initial results from the embeddings and refine them, ensuring that the most relevant results are presented to the user. This is crucial for delivering high-quality search results that meet the user's intent.



Prompt optimizers enhance the input and output of the search system, including those used in queries expansion and results rewriting. This ensures that the the search understands better and results better.

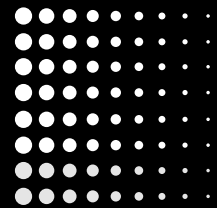


Core infra provides a cloud-native layer for developing, deploying and orchestration search foundation models both in the public cloud and on-premises, enabling services to scale up and down effortlessly.

HOW **SF** WORKS

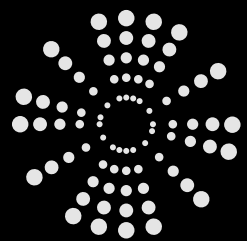
DATA PROCESSING WITH EMBEDDINGS

Embeddings transform multimodal data into a uniform, vectorized format, making diverse content searchable on equal footing. This process enables the search system to understand and categorize content beyond simple keyword.



SEARCH PRECISION WITH RERANKERS

Rerankers adjust initial search results based on deep contextual relevance, pushing the most applicable results to the top. This adjustment refines the ranking to better match what users are likely to find useful.



PROMPT OPTIMIZATION

Prompt optimizers elevate search experiences by refining queries and results through LLMs. They enhance final results' relevance and presentation, aligning closely with user intents.



SF

Increased Relevance & Reduced Search Time

Improved User Satisfaction and Trust

Higher Engagement and Conversion Rates

Direct Increase in Sales Volume

New Applications to Unlock Business Growth

A vertical cyan bar on the left side of the text.

DSPy:
Not Your Average Prompt Engineering

What is Prompt Engineering?

PROMPT ENGINEERING

- Prompt Engineering, also known as In-Context Prompting or In-Context Learning.
- It refers to methods for how to communicate with LLM to steer its behavior for desired outcomes *without* updating the model weights.
- It is the non-parametric counterpart of model fine-tuning.



PROMPT ENGINEERING

Question in early 2023, *“will prompt engineering become obsolete as LLMs keep evolving?”*

Today:

- No.
- It is even more important than before; and more popular than model fine-tuning.
- 20% of EMNLP'23 publications are about prompt engineering.
- Popularity of string templates library: LangChain, LlamaIndex.



WHY DONT YOU LIKE PROMPT ENGINEERING

- It is simple, effective and cost-efficient.

→ At almost no cost on GPU

→ Take you 5 mins to validate the effectiveness via LLM UI/API

→ Most tricks can be explained in one or few-sentences, instead of 8-page

- It is brittle and lacks of a systematic way to improve it.

PROMPT ENGINEERING – THE URLY PARTS

- Cheesy, anthropomorphic:
 - “My grandma is dying”
 - “I will tip you \$50 if you can get it right”
 - “But ChatGPT can do it”
- A lot of tricks, requiring heavy experimentation and heuristics.
- Results can not be universally applied to all LLMs/VLMs, or even to the different versions of the same LLMs (gpt3->3.5->4)
- The loop is not closed: **easy to start but hard to iterate.**

A vertical cyan bar on the left side of the text.

DSPy:
Not Your Average Prompt Engineering

Basic Prompt Engineering Modules

ZERO-SHOT

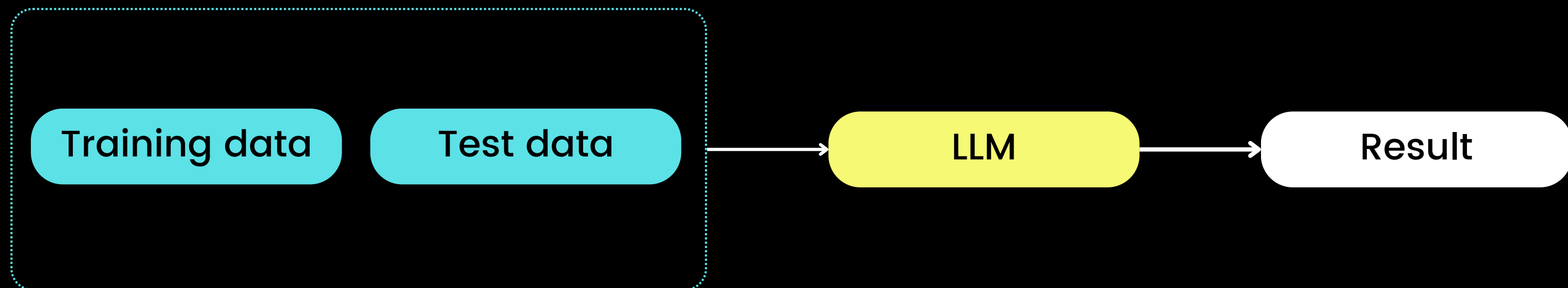
```
Zero-shot  
1 Text: Oh, great. Another meeting. Just what I needed to make my day even more exciting.  
2 Sentiment:
```



FEW-SHOT

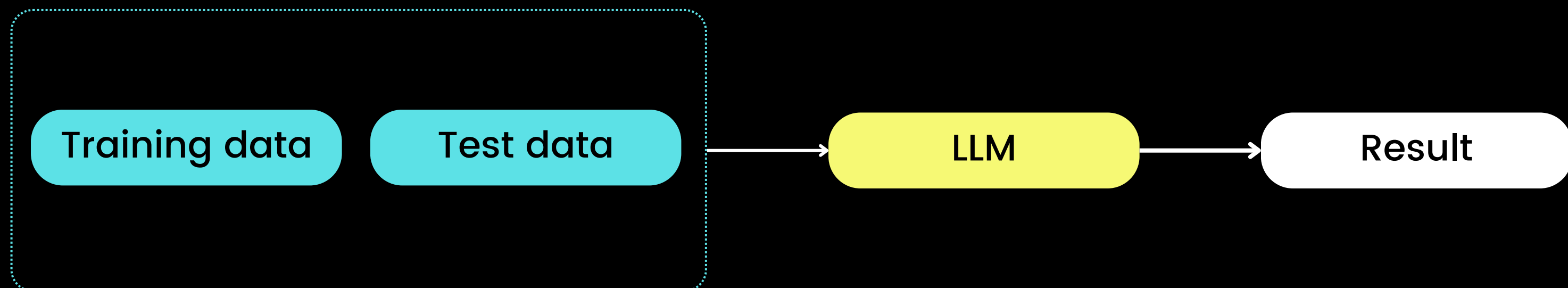
```
Few-shot examples

1 Text: Wow, thanks for leaving your dirty dishes in the sink for the tenth time this week. It really adds to the ambiance of our kitchen.
2 Sentiment: negative
3 Text: Oh, wonderful. Another software update. I'm thrilled to see what new bugs and glitches it brings.
4 Sentiment: negative
5 Text: Wow, you're so good at interrupting people. I wish I had that skill too.
6 Sentiment: negative
7 Text: Oh, fantastic! The traffic is absolutely amazing today. I can't wait to spend another hour sitting in my car.
8 Sentiment: negative
9 Text: I had an amazing time on my vacation. The sights, the food, and the people were all incredible.
10 Sentiment:
```



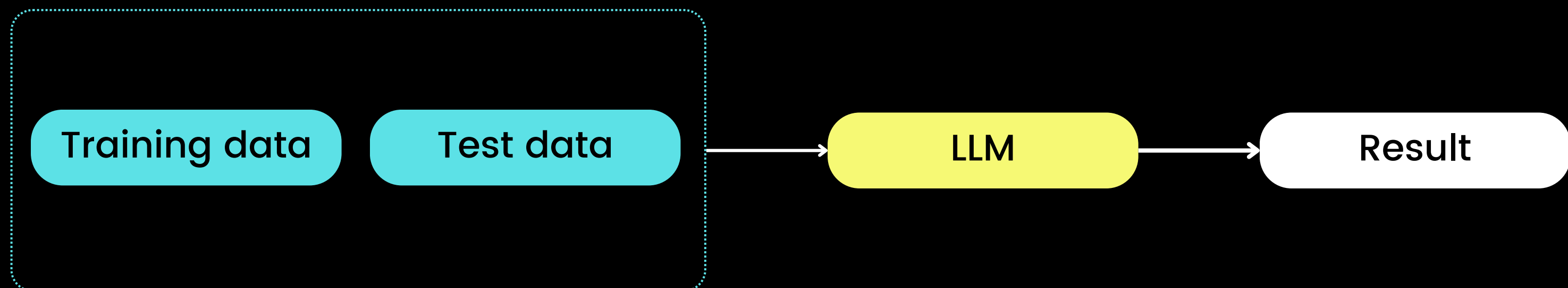
FEW-SHOT

- Presents a set of high-quality demonstrations, each consisting of both input and desired output, on the target task.
- Performance is influenced by
 - Training examples, and the order of the examples
 - Example string template



FEW-SHOT BIASES

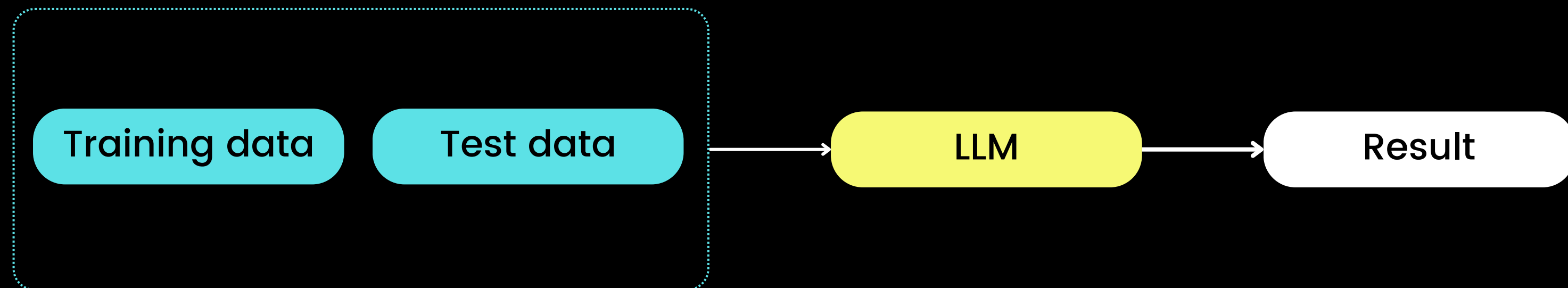
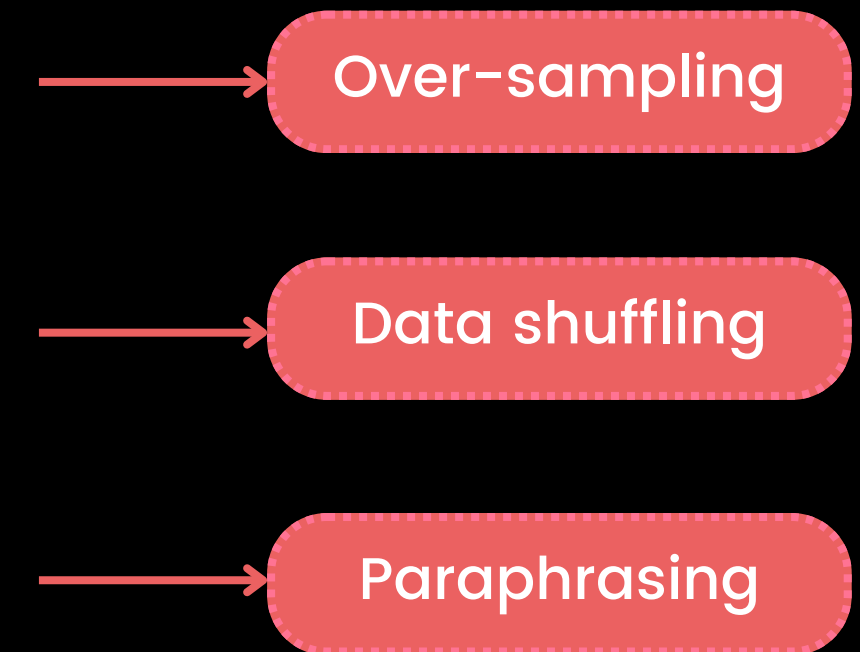
- **Majority label bias** exists if distribution of labels among the examples is unbalanced;
- **Recency bias** refers to the tendency where the model may repeat the label at the end;
- **Common token bias** indicates that LLM tends to produce common tokens more often than rare tokens.



FEW-SHOT BIASES

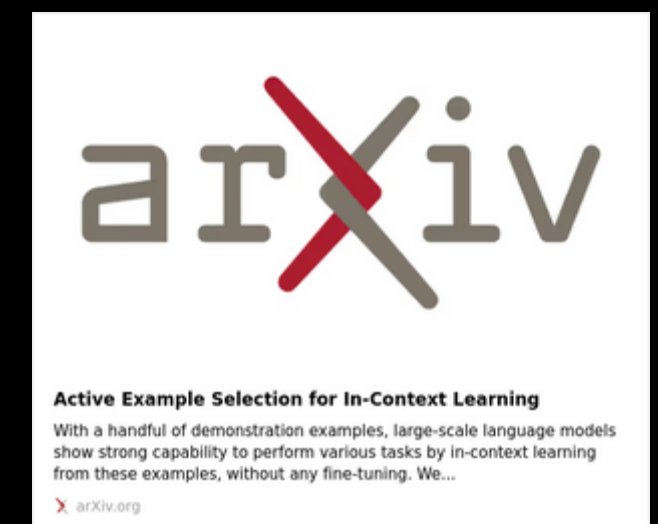
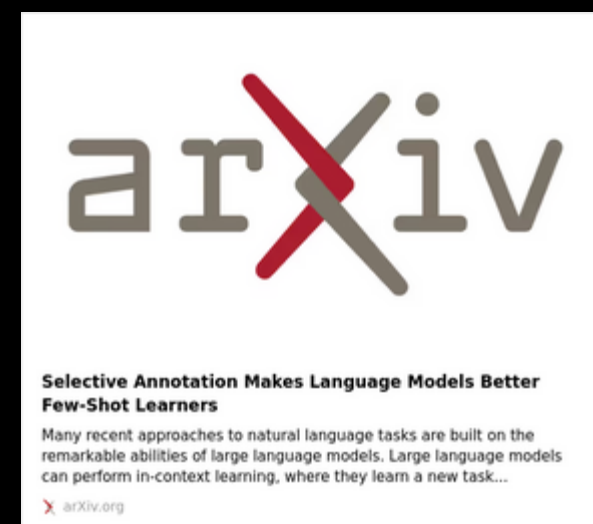
- **Majority label bias** exists if distribution of labels among the examples is unbalanced;
- **Recency bias** refers to the tendency where the model may repeat the label at the end;
- **Common token bias** indicates that LLM tends to produce common tokens more often than rare tokens.

In classic ML



FEW-SHOT SELECTION

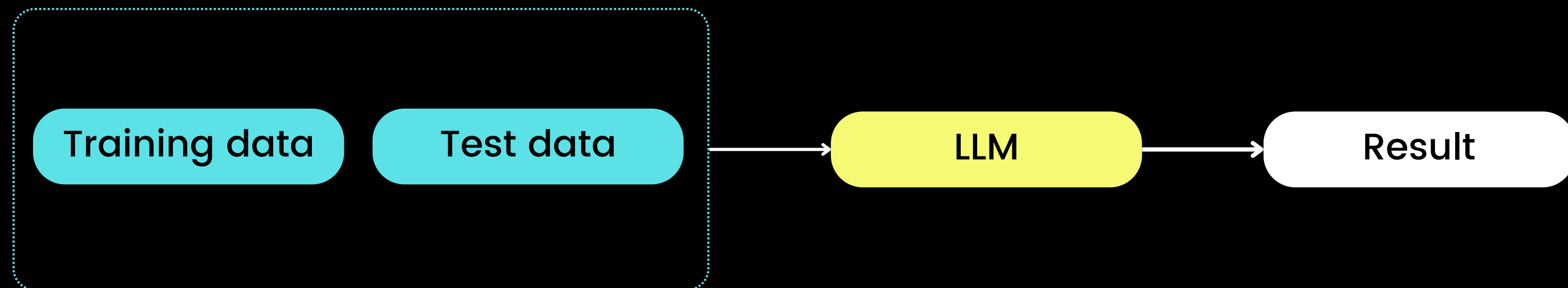
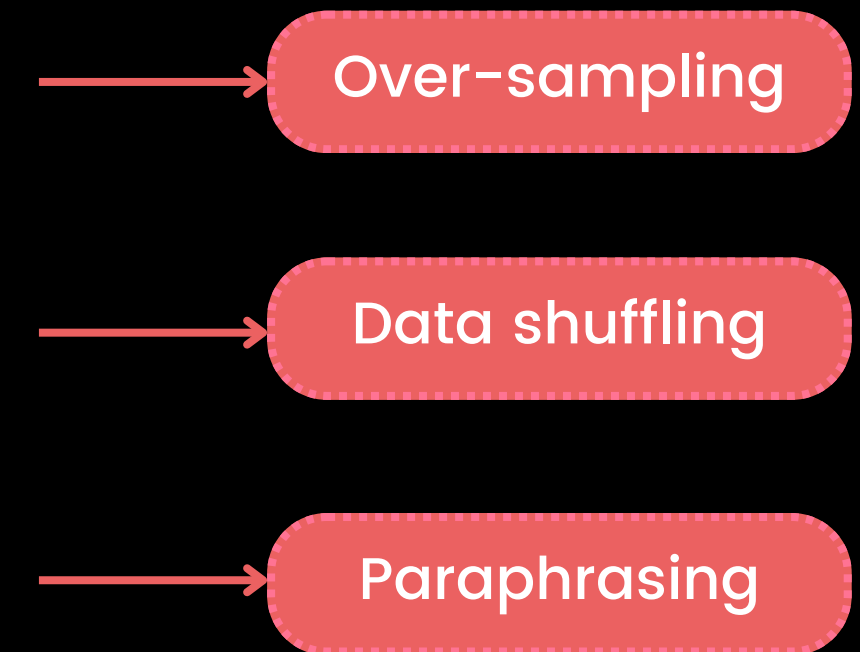
- Liu et al., (2021) choose examples that are semantically similar to the test example using k-NN clustering in the embedding space.
- Su et al. (2022) proposed to use a graph-based approach.
- Rubin et al. (2022) proposed to train embeddings via contrastive learning specific to one training dataset for in-context learning sample selection.
- Zhang et al. (2022) tried Q-Learning to do sample selection.
- Diao et al. (2023) suggested to identify examples with high disagreement or entropy among multiple sampling trials. Then annotate these examples to be used in few-shot prompts.



FEW-SHOT BIASES

- **Majority label bias** exists if distribution of labels among the examples is unbalanced;
- **Recency bias** refers to the tendency where the model may repeat the label at the end;
- **Common token bias** indicates that LLM tends to produce common tokens more often than rare tokens.

In classic ML

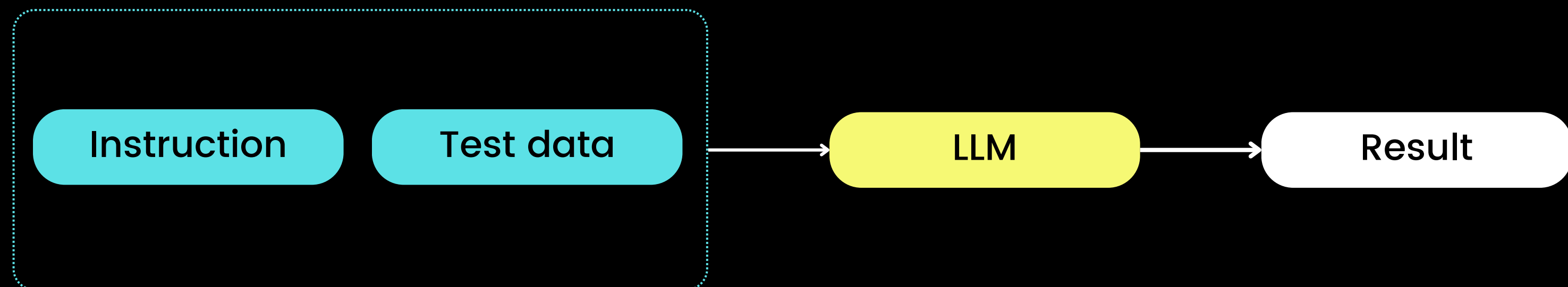


INSTRUCTION PROMPT



Instruction prompt

```
1 Please label sentiment of the sentence below into "positive", "negative" and "neutral".  
2 Text: I had an amazing time on my vacation. The sights, the food, and the people were  
   all incredible.  
3 Sentiment:
```



INSTRUCTION PROMPT

- Explicitly telling model what to do, instead of showing a set of demonstrations (i.e. few-shot) and let model immitate.



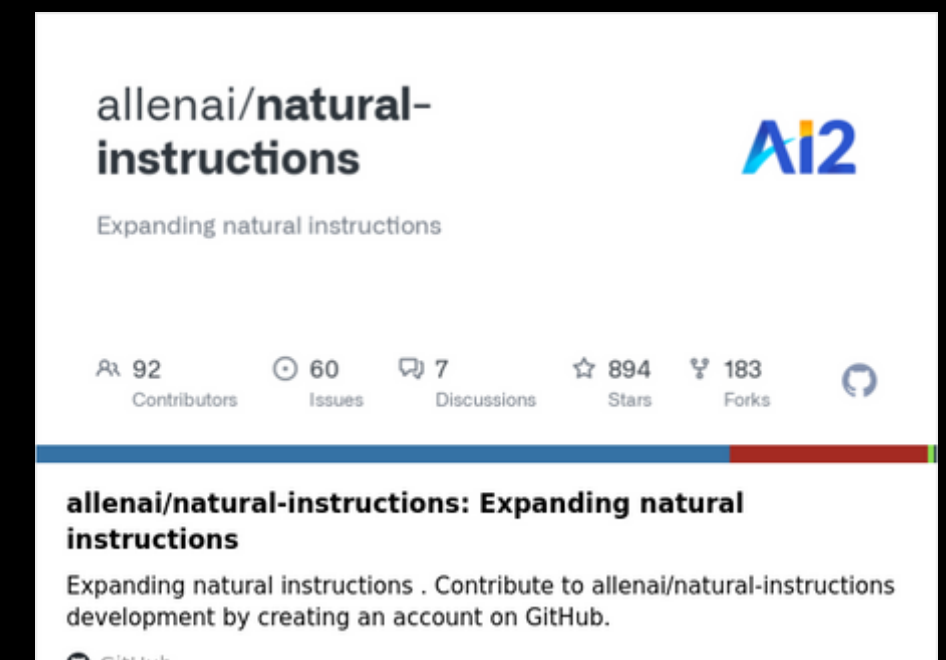
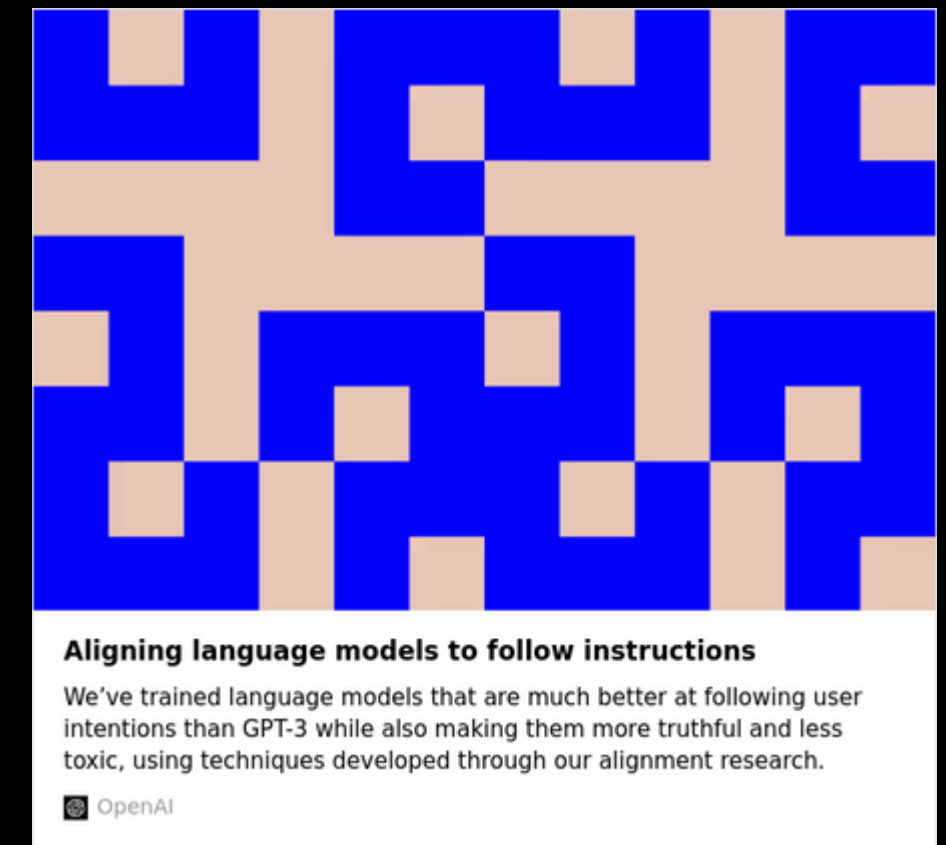
Instruction few-shot prompting



Few-shot instruction prompting

INSTRUCTION-FOLLOWING MODELS

- Instruction-following is **not** for granted. Think about:
 - An embedding model where “question:” and “answer:” prefix give different embeddings for the same sentence
 - A reranker to “find most **un**related articles of ...”
- Instructed LM finetunes a pretrained model with high-quality tuples of **(task instruction, input, ground truth output)** to make LM better understand user intention and follow instruction.
- RLHF (Reinforcement Learning from Human Feedback) is a common method to do so. The benefit of instruction following style fine-tuning improves the model to be more aligned with human intention and greatly reduces the cost of communication.



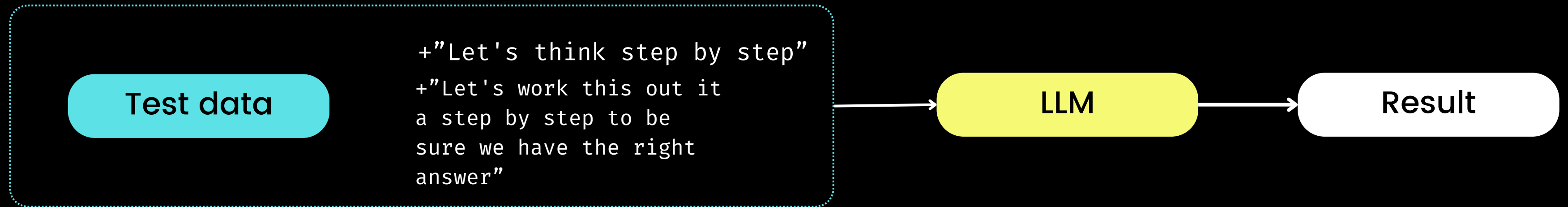
CHAIN-OF-THOUGHT (COT)

Chain of thoughts prompt

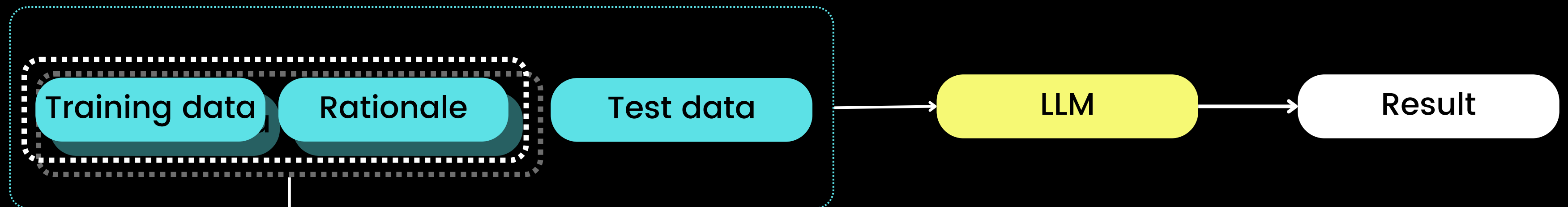
- 1 Question: Tom and Elizabeth have a competition to climb a hill. Elizabeth takes 30 minutes to climb the hill. Tom takes four times as long as Elizabeth does to climb the hill. How many hours does it take Tom to climb up the hill?
- 2 Answer: It takes Tom $30 \times 4 = \ll 30 \times 4 = 120 \gg 120$ minutes to climb the hill.
- 3 It takes Tom $120 / 60 = \ll 120 / 60 = 2 \gg 2$ hours to climb the hill.
- 4 So the answer is 2.
- 5 \equiv
- 6 Question: Jack is a soccer player. He needs to buy two pairs of socks and a pair of soccer shoes. Each pair of socks cost \$9.50, and the shoes cost \$92. Jack has \$40. How much more money does Jack need?
- 7 Answer: The total cost of two pairs of socks is $\$9.50 \times 2 = \$\ll 9.5 \times 2 = 19 \gg 19$.
- 8 The total cost of the socks and the shoes is $\$19 + \$92 = \$\ll 19 + 92 = 111 \gg 111$.
- 9 Jack need $\$111 - \$40 = \$\ll 111 - 40 = 71 \gg 71$ more.
- 10 So the answer is 71.
- 11 \equiv
- 12 **Question: There are three birds on the tree, shot one down, how many are left on the tree?**
- 13 **Answer:**

CHAIN-OF-THOUGHT (COT)

Zero-shot CoT



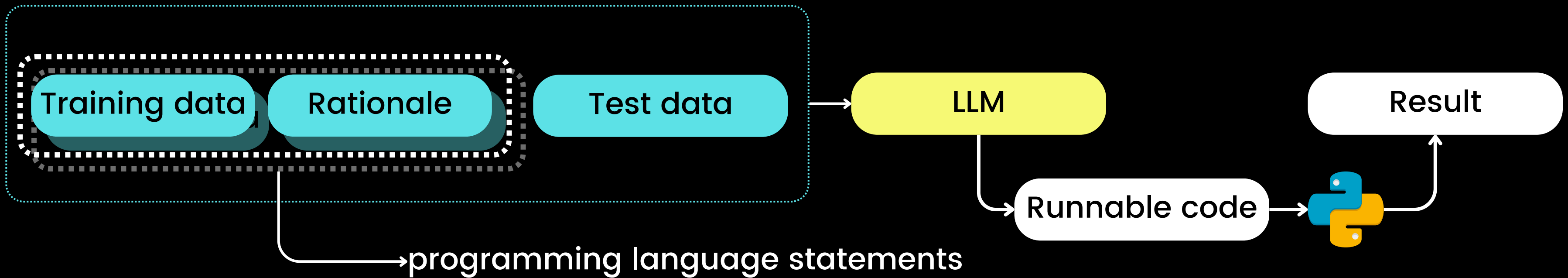
Few-shot CoT



→ manually written/model-generated high-quality reasoning chains.

PROGRAM-OF-THOUGHT (POT)

Few-shot PoT



Question: In Fibonacci sequence, it follows the rule that each number is equal to the sum of the preceding two numbers. Assuming the first two numbers are 0 and 1, what is the 50th number in Fibonacci sequence?

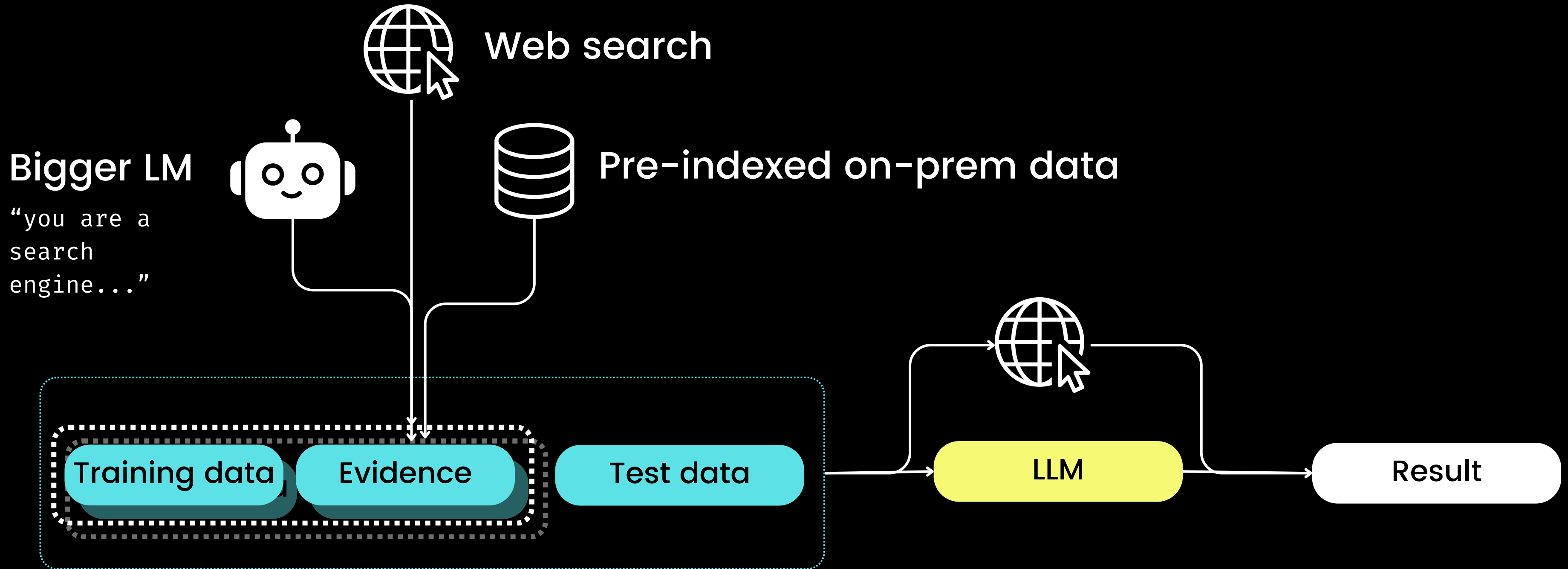
<p>The first number is 0, the second number is 1, therefore, the third number is 0+1=1. The fourth number is 1+1=2. The fifth number is 1+2=3. The sixth number is 2+3=5. The seventh number is 3+5=8. The eighth number is 5+8=13. (Skip 1000 tokens) The 50th number is 32,432,268,459.</p> <p>CoT</p>	<pre>length_of_fibonacci_sequence = 50 fibonacci_sequence = np.zeros(length_of_) fibonacci_sequence[0] = 0 fibonacci_sequence[1] = 1 For i in range(3, length_of_fibonacci_sequence): fibonacci_sequence[i] = fibonacci_sequence[i-1] + fibonacci_sequence[i-2] ans = fibonacci_sequence[-1]</pre> <p>PoT</p>
--	---

32,432,268,459 ❌

python

12,586,269,025 ✅

RETRIEVER



READER

Convert any URL to an LLM-friendly input with a simple prefix `https://r.jina.ai/`



READER

<p>Enter your URL https://arxiv.org/abs/2310.19923</p> <p>Click below to fetch the source code of the page directly</p>	<p>Reader URL https://r.jina.ai/https://arxiv.org/abs/2310.19923</p> <p><input type="checkbox"/> Stream Mode Click below to obtain the content through our Reader API</p>
<p>↓ FETCH CONTENT</p>	
<pre><?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD, <html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en"> <head> <title>[2310.19923] Jina Embeddings 2: 8192-Token General-Purpose Text Embeddings for Lc <meta name="viewport" content="width=device-width, initial-scale=1"> <link rel="apple-touch-icon" sizes="180x180" href="/static/browse/0.3.4/images/icons/apple-to <link rel="icon" type="image/png" sizes="32x32" href="/static/browse/0.3.4/images/icons/favico <link rel="icon" type="image/png" sizes="16x16" href="/static/browse/0.3.4/images/icons/favico <link rel="manifest" href="/static/browse/0.3.4/images/icons/site.webmanifest"> <link rel="mask-icon" href="/static/browse/0.3.4/images/icons/safari-pinned-tab.svg" color="#! <meta name="msapplication-TileColor" content="#da532c"> <meta name="theme-color" content="#ffffff"> <link rel="stylesheet" type="text/css" media="screen" href="/static/browse/0.3.4/css/arXiv.cs <link rel="stylesheet" type="text/css" media="print" href="/static/browse/0.3.4/css/arXiv-pri <link rel="stylesheet" type="text/css" media="screen" href="/static/browse/0.3.4/css/browse_sc</pre>	<p>Title: Jina Embeddings 2: 8192-Token General-Purpose Text Embeddings for Long Documents</p> <p>URL Source: https://arxiv.org/abs/2310.19923</p> <p>Markdown Content: Authors: [Michael Günther](https://arxiv.org/search/cs?searchtype=author&query=G%C3%BCnther,+M), [View PDF](https://arxiv.org/pdf/2310.19923) [HTML (experimental)](https://arxiv.org/html/2310.19923.v1) </p> <p>> Abstract:Text embedding models have emerged as powerful tools for transforming sentences into > To address these challenges, we introduce Jina Embeddings 2, an open-source text embedding model</p> <p>Submission history ----- From: Han Xiao \[[view email](https://arxiv.org/show-email/11a0a836/2310.19923)\] **Date: 2023-10-25 10:25:20 UTC (707 KB)</p>
<p>Pose a Question Summarize the content in two sentences.</p> <hr/> <p>Input a question and combine it with the fetched content for LLM to generate an answer</p>	
<p>The content discusses the challenges faced by existing open-source text embedding models in representing lengthy documents and introduces Jina Embeddings 2 as a solution to address these challenges.</p>	<p>The paper introduces Jina Embeddings 2, an open-source text embedding model that can handle up to 8192 tokens, addressing the limitation of existing models in representing long documents. The model achieves state-of-the-art performance on various embedding-related tasks and matches the performance of OpenAI's ada-002 model, with experiments showing that extended context improves performance in tasks like NarrativeQA.</p>

ALL BASIC MODULES

Prompt

“Parameter” of the prompt

Zero-shot

Test data

LLM

Result

Few-shot

Training data

Test data

LLM

Result

Instruction
few-shot

Instruction

Training data

Test data

LLM

Result

Chain of
Thought

Training data

Rationale

Test data

LLM

Result

Retrieval

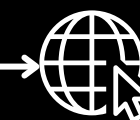
Training data

Evidence

Test data

LLM

Result



ALL BASIC MODULES

Prompt

“Parameter” of the prompt

Zero-shot

Test data

LLM

Result

Few-shot

Training data

Test data

LLM

Result

Instruction
few-shot

Instruction

Training data

Test data

LLM

Result

Chain of
Thought

Training data

Rationale

Test data

LLM

Result

Retrieval

Training data

Evidence

Test data

LLM

Result



ALL BASIC MODULES

Prompt

“Parameter” of the prompt

Zero-shot

Test data

LLM

Result

Few-shot

Training data

Test data

LLM

Result

Instruction
few-shot

Instruction

Training data

Test data

LLM

DSPy

Chain of
Thought

Training data

Rationale

Test data

*DSPy is a framework for algorithmically **optimizing LM prompts** and weights, especially when LMs are used one or more times within a pipeline.*

Result

Retrieval

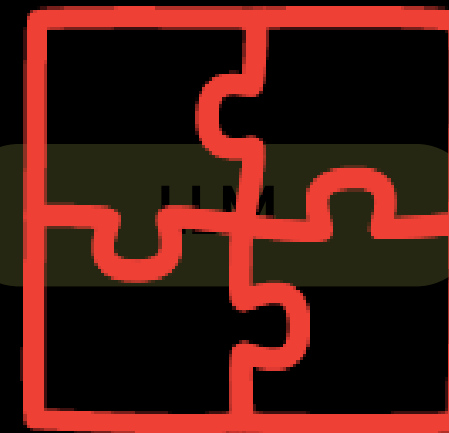
Training data

Evidence

Test data

LLM

Result

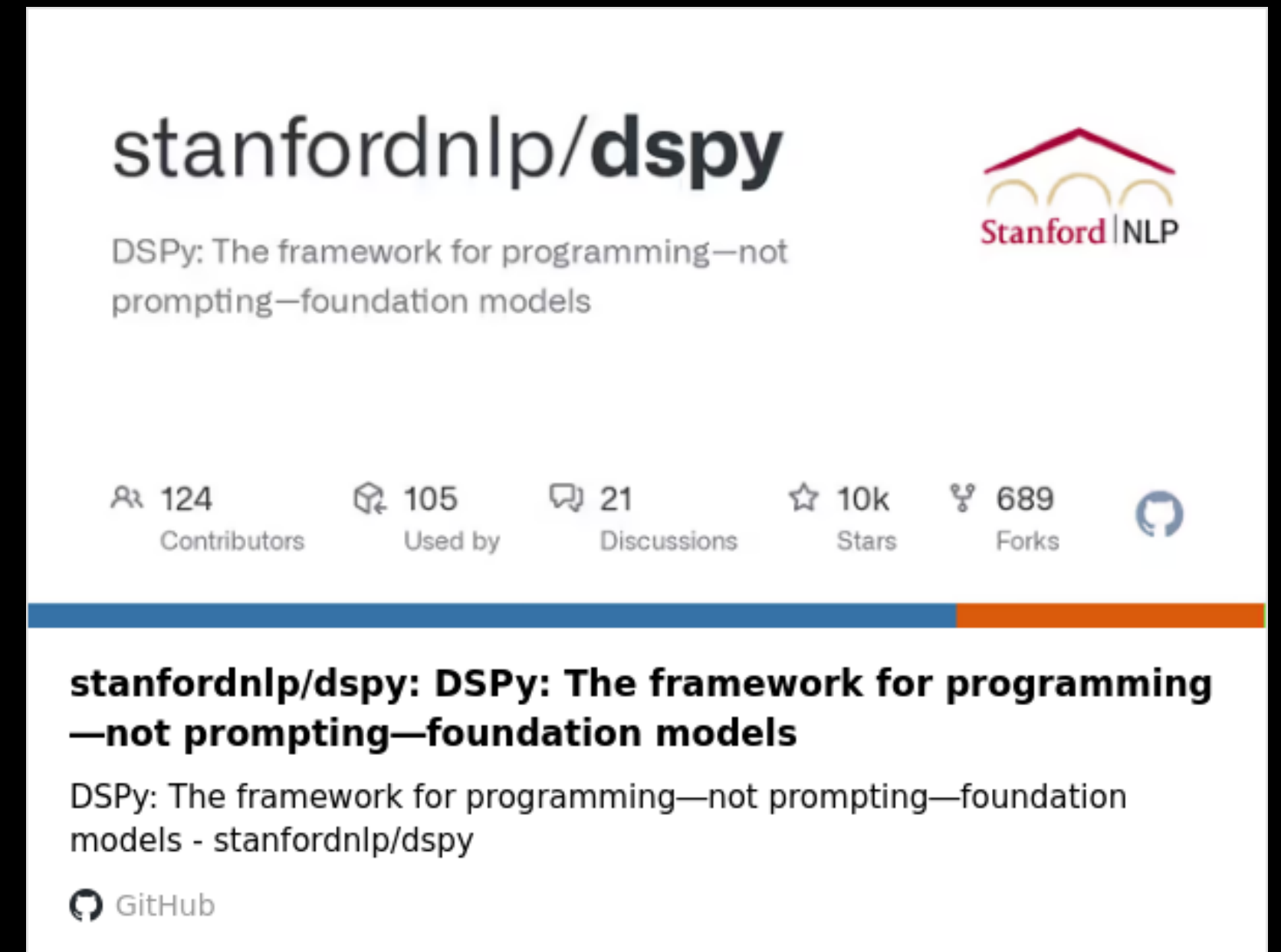


DSPy:
Not Your Average Prompt Engineering

What is DSPy

DSPY

- **Declarative Self-improving Language Programs**, pythonically.
- DSPy is a framework for algorithmically optimizing prompts and LM weights, especially in a prompt pipeline.
- However, it is hard to learn.
 - *"Yeah man, I have been seeing DSPy everywhere but haven't found time to check it out yet"* - almost everyone I talk to about the project.



The screenshot shows the GitHub repository page for `stanfordnlp/dspy`. The repository name is displayed in a large, bold font. Below it, the description reads: "DSPy: The framework for programming—not prompting—foundation models". To the right of the description is the Stanford NLP logo, which consists of a red roof-like shape above the text "Stanford | NLP".

Below the description, there are several statistics and icons: 124 Contributors, 105 Used by, 21 Discussions, 10k Stars, and 689 Forks. A GitHub logo icon is also present.

A blue and orange horizontal bar is located below the statistics. Below this bar, the repository name and description are repeated in a smaller font: **stanfordnlp/dspy: DSPy: The framework for programming—not prompting—foundation models**. Below this, the description is repeated: "DSPy: The framework for programming—not prompting—foundation models - stanfordnlp/dspy". At the bottom left, there is a GitHub logo icon and the text "GitHub".

UNDERSTANDING DSPY

- DSPy closes the loop of prompt engineering;
- DSPy separates the logic (what) from textual representation (how).

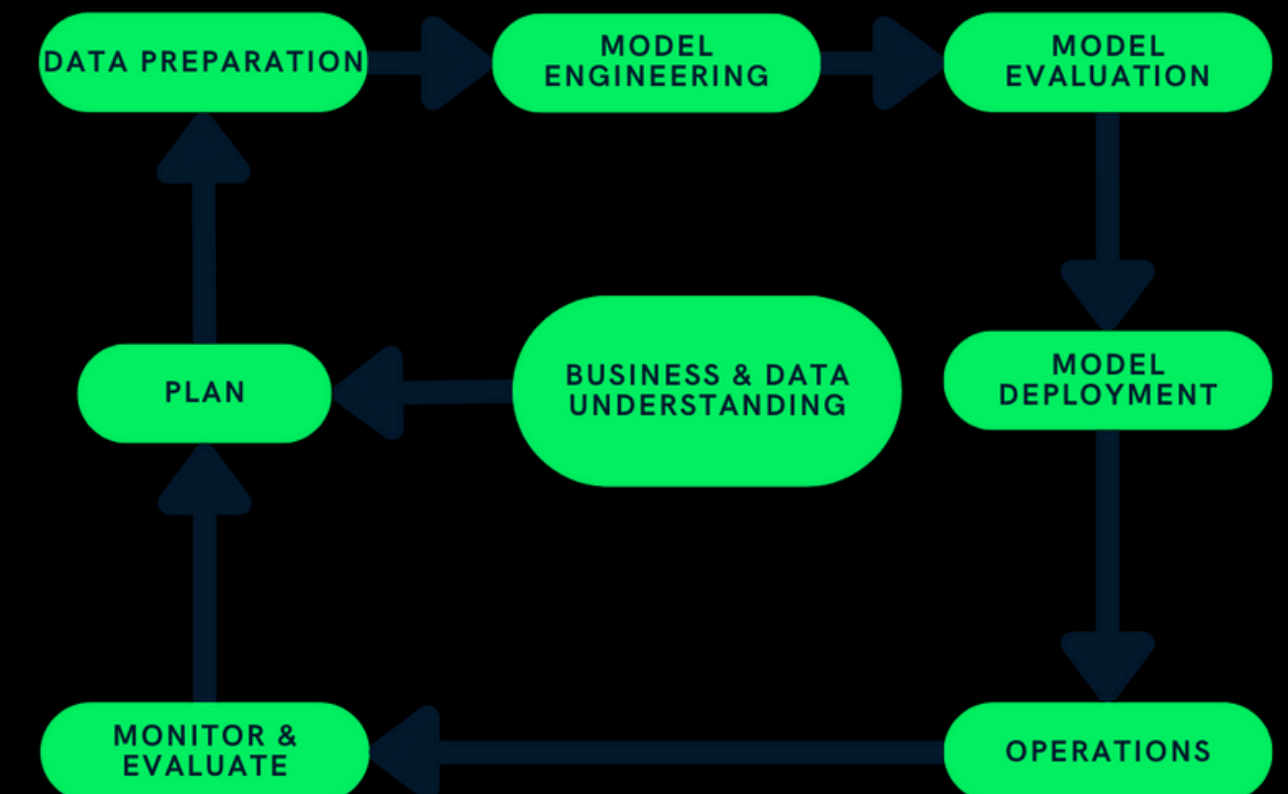
UNDERSTANDING DSPY

- DSPy closes the loop of prompt engineering;

Transforming prompt engineering from what is often a *manual, handcrafted process* into a *structured, well-defined machine learning workflow*: i.e. preparing datasets, defining the model, training, evaluating, and testing. In my opinion, this is the most revolutionary aspect of DSPy.

UNDERSTANDING DSPY

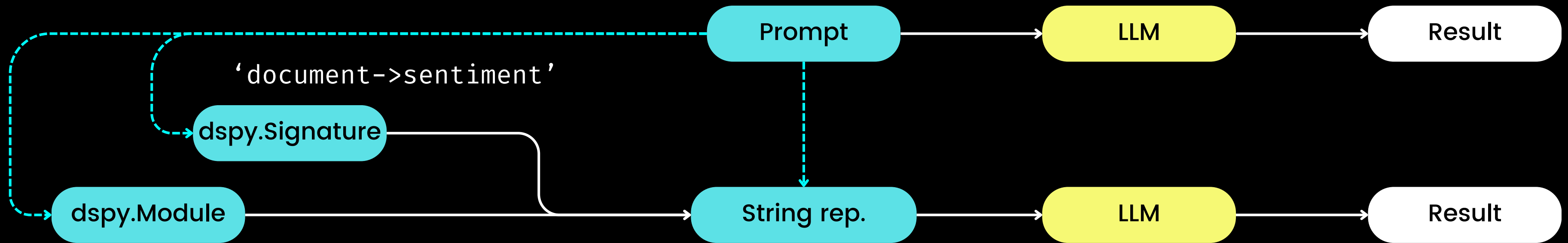
- Cheesy, anthropomorphic:
 - “My grandma is dying”
 - “I will tip you \$50 if you can get it right”
 - “But ChatGPT can do it”
- A lot of tricks, requiring heavy experimentation and heuristics.
- Results can not be universally applied to all LLMs/VLMs, or even to the different versions of the same LLMs (gpt3->3.5->4)



UNDERSTANDING DSPY

- DSPy separates the logic (what) from textual representation (how).

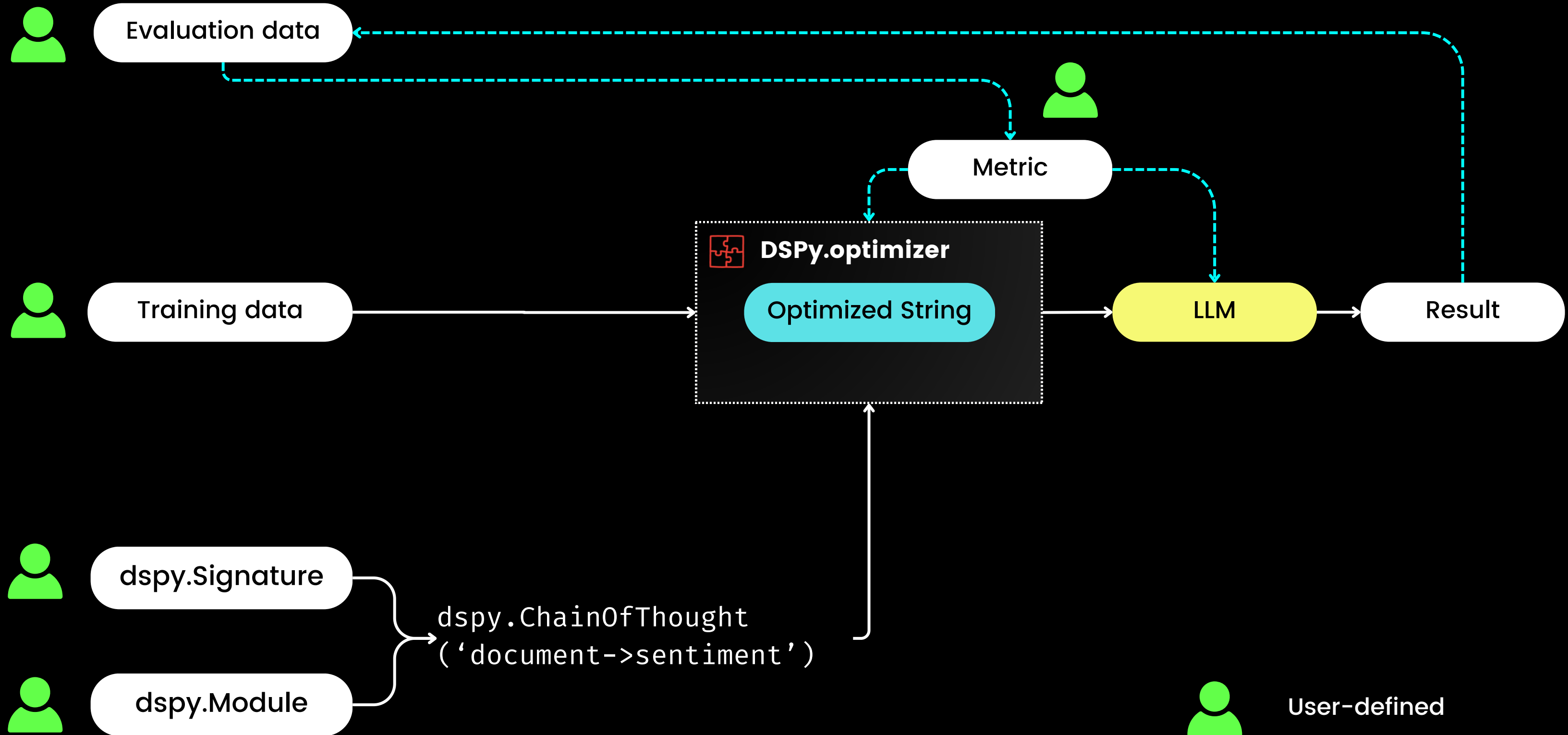
“This is important to me, I will lose my job if I can’t get the sentiment classification correct ...”



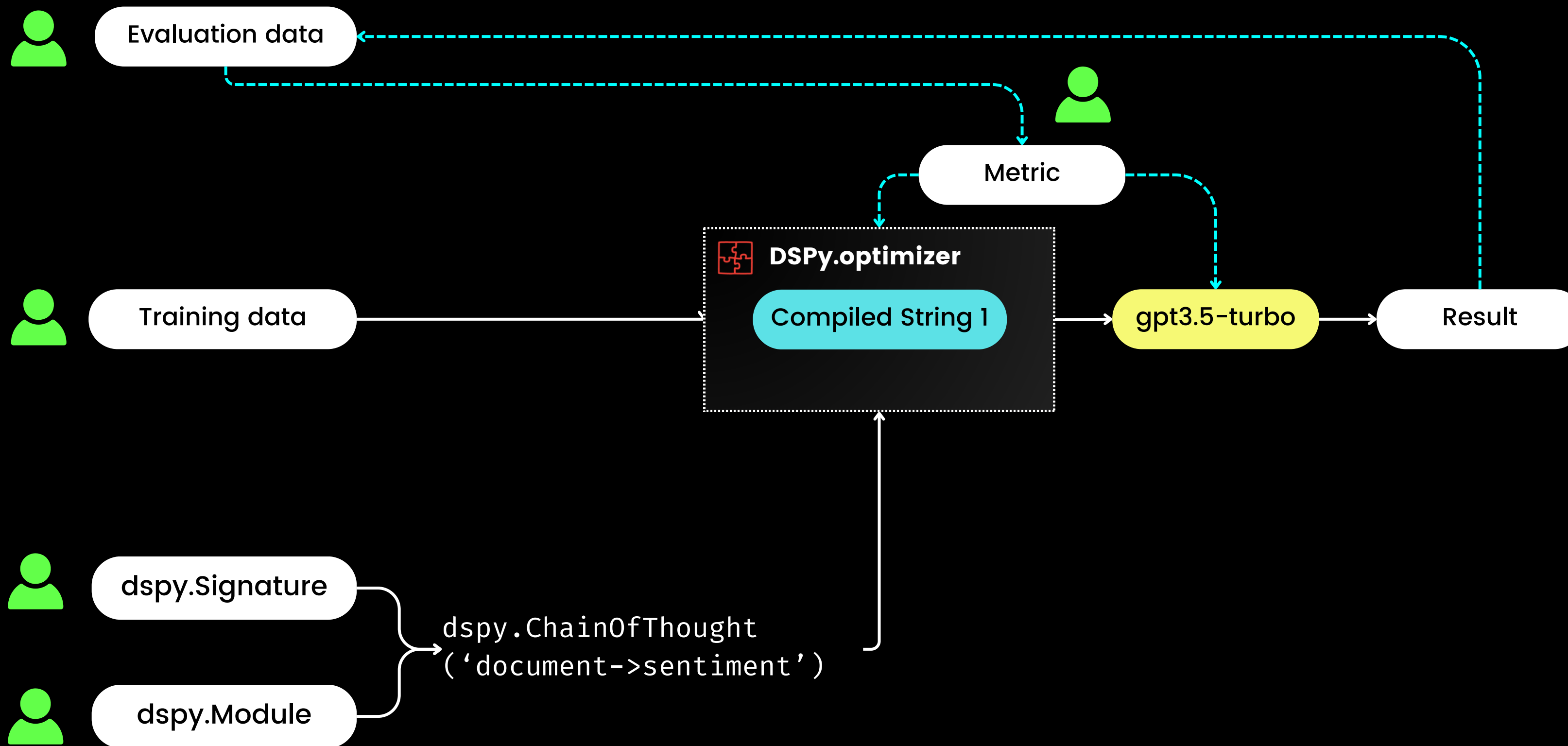
Predict
ChainOfThought
ReAct
ProgramOfThought

“... get the sentiment classification correct ... important ... lose my job ...”

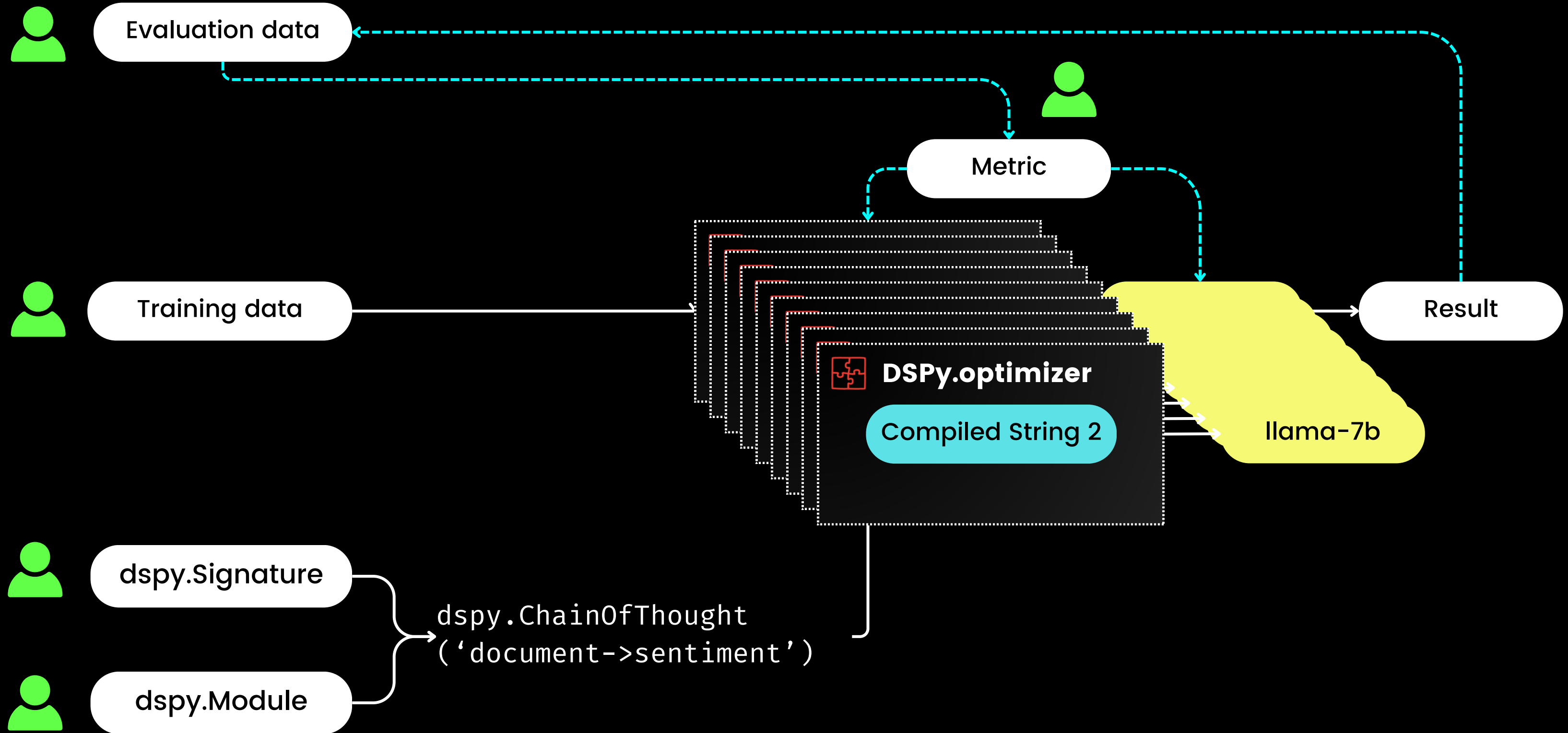
DSPY.OPTIMIZER.COMPILE



DSPY.OPTIMIZER.COMPILE



DSPY.OPTIMIZER



ALL BASIC MODULES

Prompt

“Parameter” of the prompt

Zero-shot

Test data

LLM

Result

Few-shot

Training data

Test data

LLM

Result

Instruction
few-shot

Instruction

Training data

Test data

LLM

Result

Chain of
Thought

Training data

Rationale

Test data

LLM

Result

Retrieval

Training data

Evidence

Test data

LLM

Result



WHAT EXACTLY DSPY.COMPILE OPTIMIZE

The compile function acts at the heart of this optimizer, akin to calling `optimizer.optimize()`. Think of it as the DSPy equivalent of training. This `compile()` process aims to tune:

- the few-shot demonstrations
- the instructions
- the LLM weights

You can imagine DSPy as a toolbox of **discrete optimization methods**.

DSPy:
Not Your Average Prompt Engineering

Demo