Large Language Models

& the **Compound AI Systems** they enable us to build

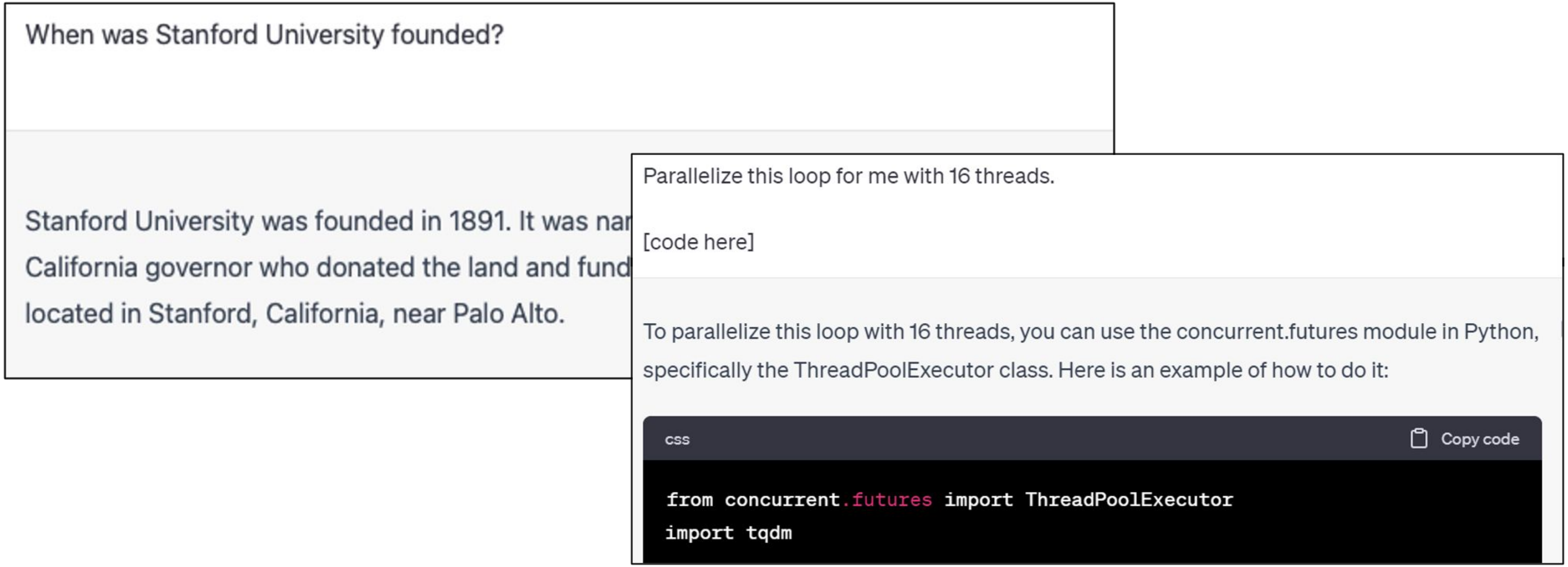
Guest Lecture for **CMU 11-785** Introduction to Deep Learning (Fall 2024)

Omar Khattab

Nov 06, 2024

Adapted from material by Bhiksha Raj, Rita Singh, Chris Manning, Anna Goldie, John Hewitt, Tatsu Hashimoto, Yann Dubois, Archit Sharma, Jesse Mu, Michael Ryan, and Krista Opsahl-Ong.

It’s never been easier to prototype impressive AI assistants & demos.



How deep learning got us to this stage — an outline

1. **Neural Language Models**: Using *Transformers* to model language and for autoregressive decoding.

2. **Pre-Training**: Giving the LMs *broad knowledge* of language, the world, and maybe some "reasoning".

3. **Post-Training**: Teaching the LMs how to *behave as assistants* that are instruction-following, safe, etc.

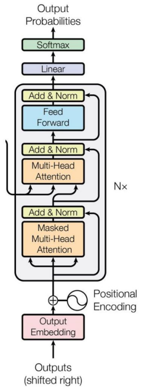
4. **Compound AI Systems**: *Composing LM skills* into modular user-facing systems and optimizing them in various ways for downstream tasks.

Neural Language Models: Using Transformers for autoregressive decoding.

- **In the previous lecture, we learned about Transformers.**
- **Recap: Autoregressive decoding.**
- While we haven't finished the sequence:
    1. **Tokenize** the input text.
    2. **Forward pass:** Process the current sequence through the Transformer model.
    3. **Sample next token:** Predict and sample the next token based on model output.
    4. **Append** to sequence and repeat until completion.

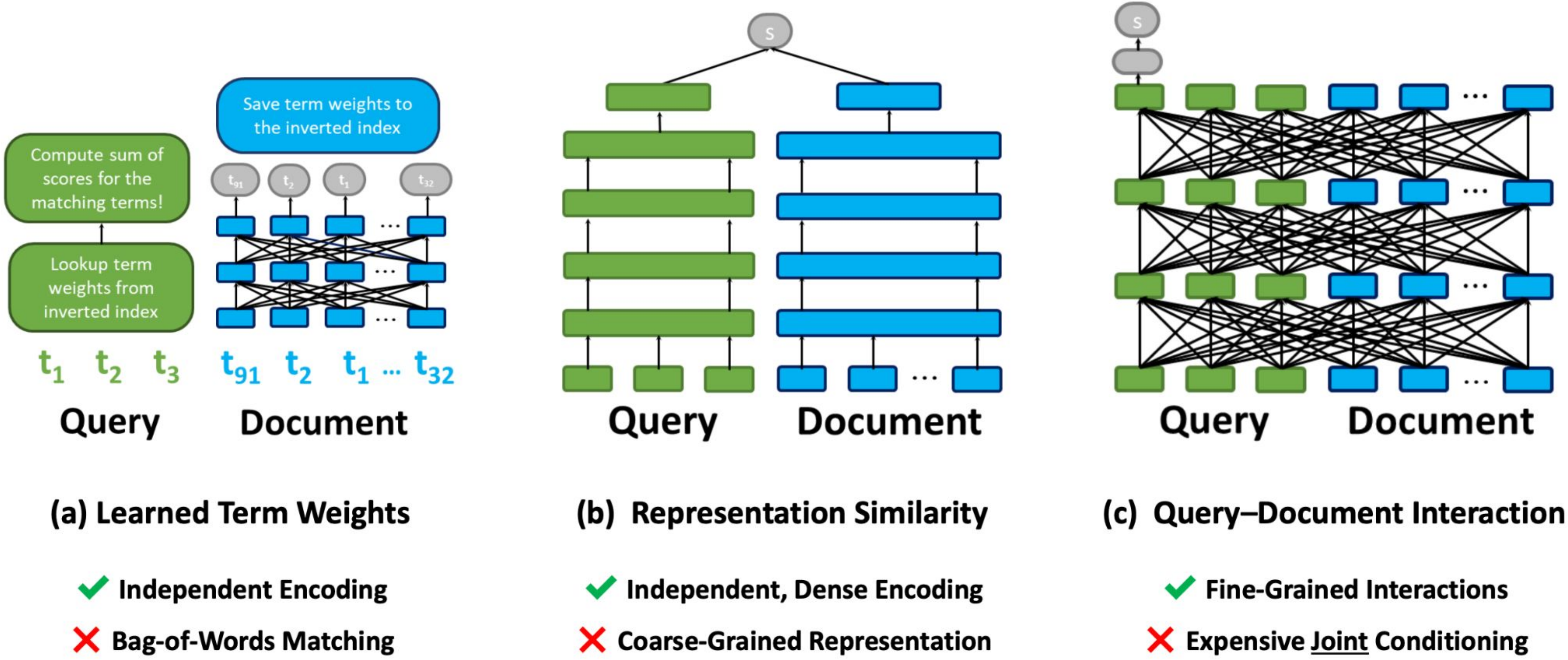
Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



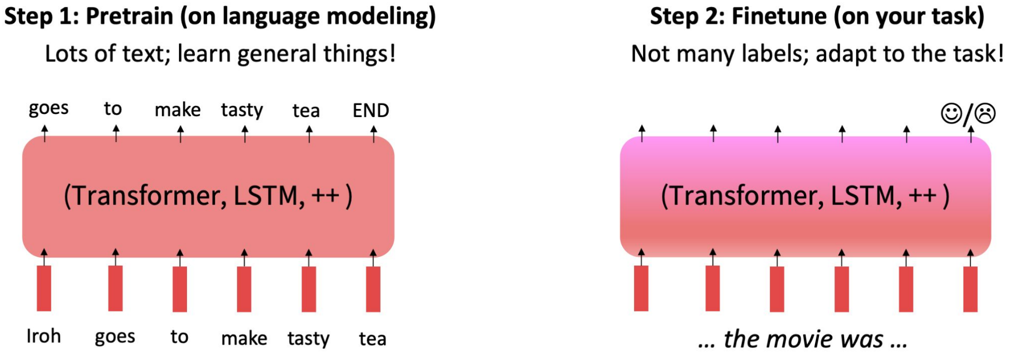
This could capture a lot of tasks. How do we train a Transformer to be able to do this well?

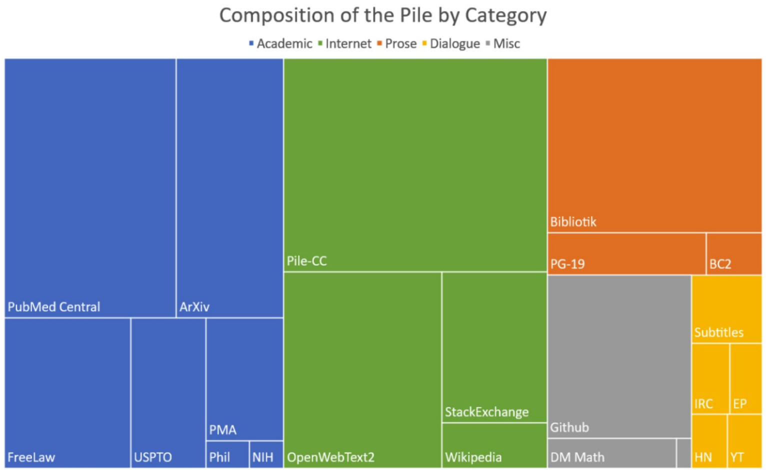
We'll focus on decoders, but encoders are still the backbone of many applications, like information retrieval!



Pre-Training: Giving the LMs broad knowledge by training

- On broad Web data — massive Web crawls, but with aggressive filtering and cleaning
- Via the task of Language Modeling, or next word prediction
  - $P(w_t \mid w_{1:t-1})$ with a standard classification cross-entropy loss





Why does such pre-training on broad data help? Perhaps it helps the gradients flow better during fine-tuning. Or maybe SGD likes to stick close to initialization parameters, so finding a local minima during fine-tuning gives us parameters that would generalize well.

Illustration from CS224N Slides by Chris Manning et al.

What does pre-training teach a Transformer? It **builds strong representations of language** and gives us a **broad foundation** that we can adapt to downstream tasks!

- *Stanford University is located in ___________, California.* [Trivia]
- *I put ___ fork down on the table.* [syntax]
- *The woman walked across the street, checking for traffic over ___ shoulder.* [coreference]
- *I went to the ocean to see the fish, turtles, seals, and _____.* [lexical semantics/topic]
- *Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.* [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the ______. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____ [some basic arithmetic; they don't learn the Fibonacci sequence]

List of Cloze sentences from CS224N Slides by Chris Manning et al.

Scaling helps: 100s of billions of parameters, trained on trillions of tokens.

Scaling predictably follows empirical patterns, which can help us make informed choices — by tuning our hyperparameters at small scale.

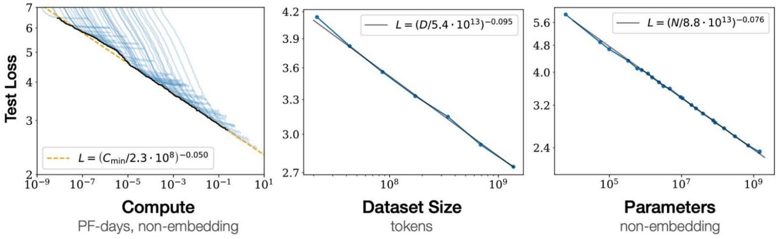
Fundamental tradeoffs: Given a fixed budget for pre-training compute (# of GPU–days), should you increase parameters or tokens seen?

What if you want to minimize *total* compute, including inference, instead?

Scaling Law

For decoder-only models, the final performance is only related to **Compute**, **Data Size**, and **Parameter Size**

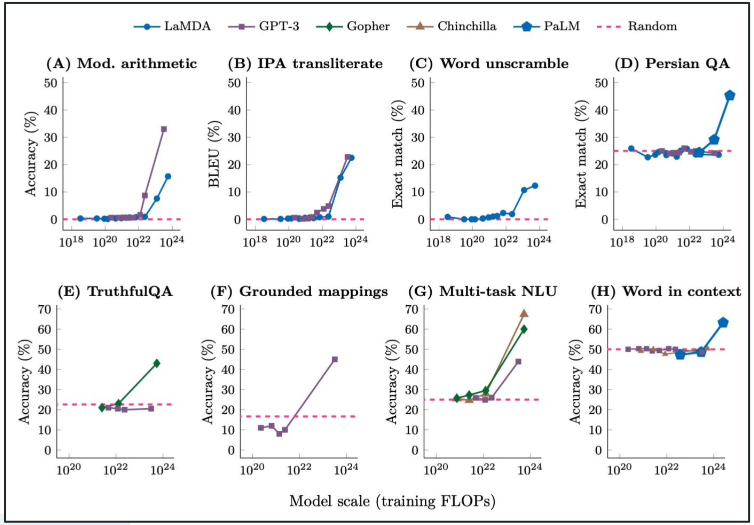
- power law relationship for each factor
- w/o constraints by the others



Kaplan et al. Scaling Laws for Neural Language Models. 2020.

99

Emergent Behavior: Scaling (appears) to also create “sudden” jumps like the capacity for In-Context Learning.



Traditional fine-tuning

```
sea otter => loutre de mer
```

gradient update

```
peppermint => menthe poivrée
```

gradient update

...

```
cheese =>
```

Few-shot

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Wei et al. (2020)

Emergent Behavior: Scaling (appears) to also create "sudden" jumps like the capacity for Chain-Of-Thought Reasoning.

Standard Prompting

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: 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 answer is 27. ✗

Chain-of-Thought Prompting

Model Input

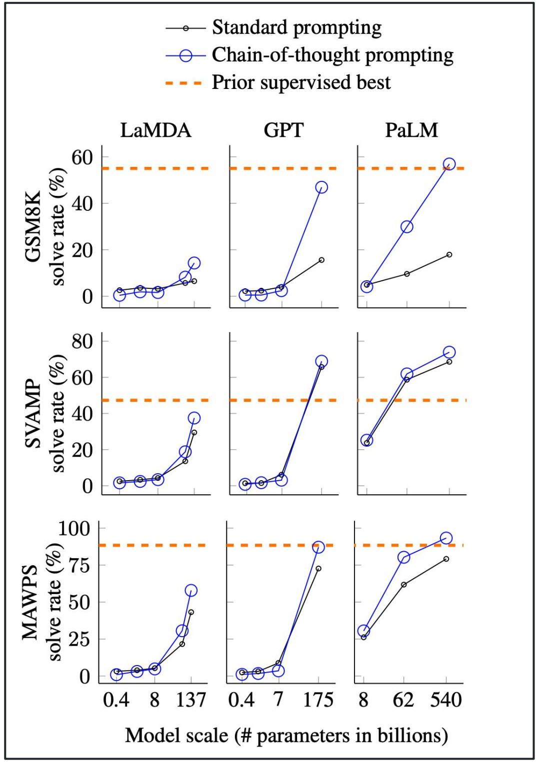
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

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

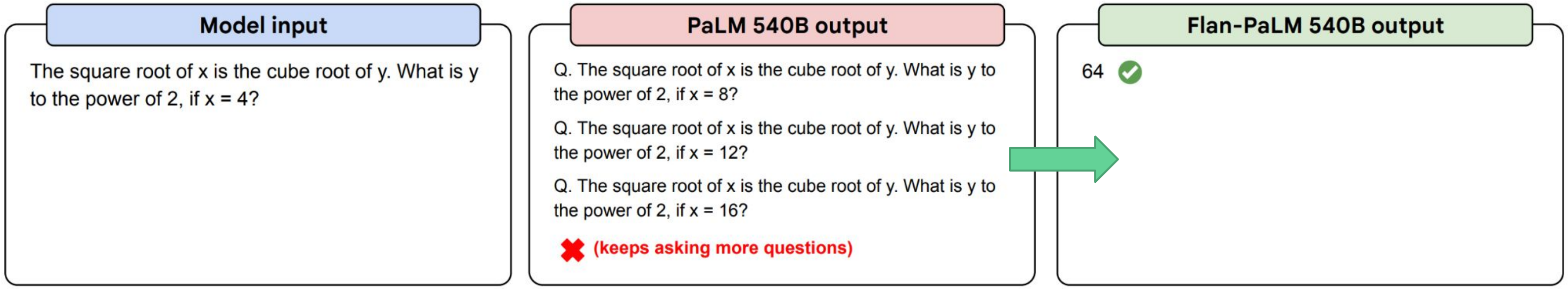
A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

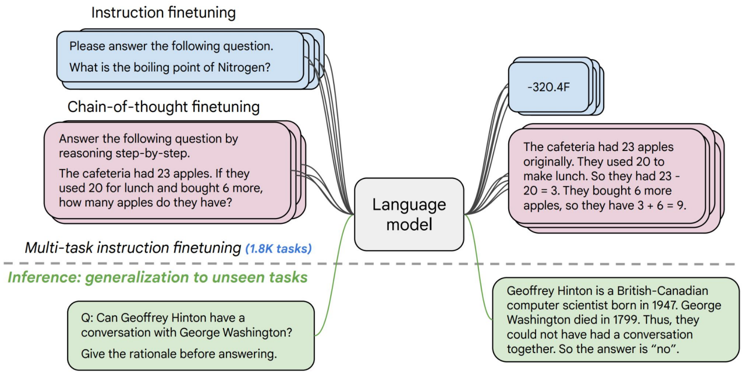


Wei et al. (2022)

Post-Training: Teaching the LMs how to behave as assistants that are instruction-following, safe, etc. How should we do that?

*One approach is **Instruction Fine-Tuning**: labeling examples of <prompt, response> pairs that spans many tasks and training on them.*

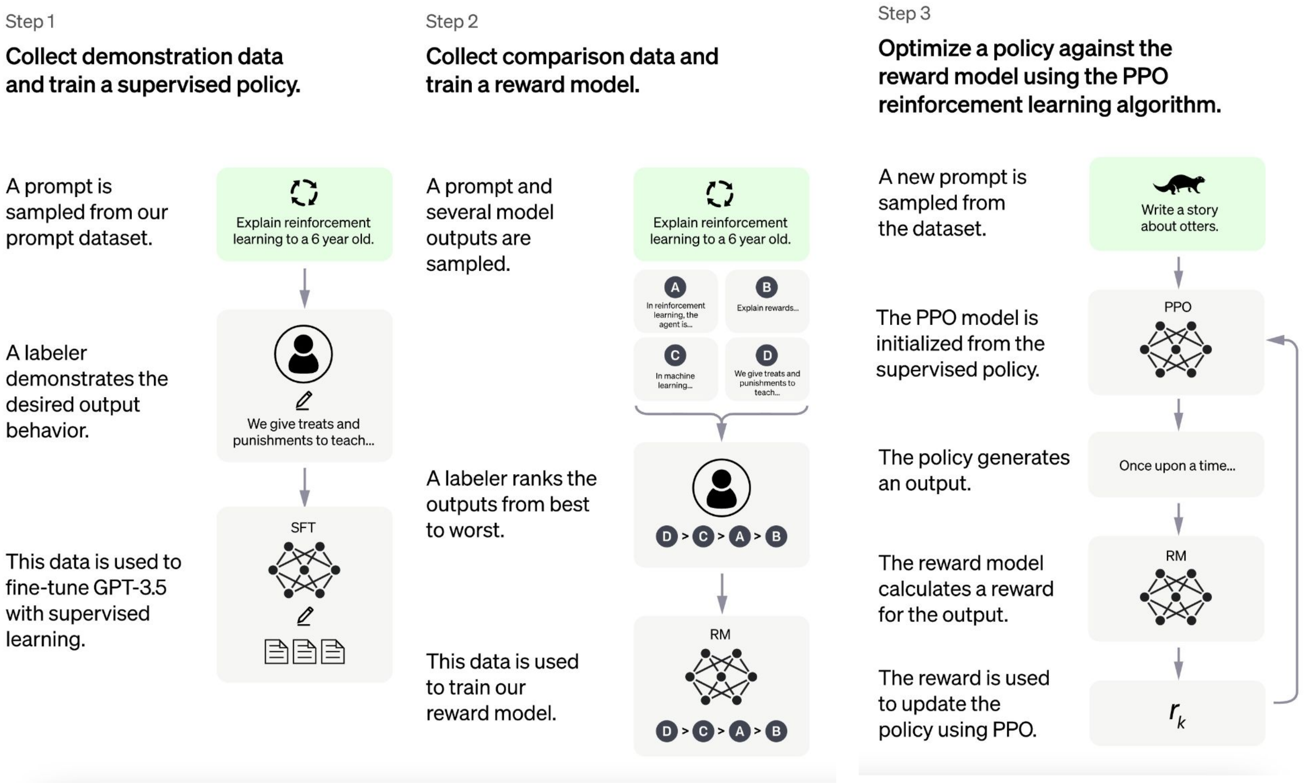




Unfortunately, this is expensive and unscalable. It also doesn't quite teach the right thing for longer or open-ended generation: poor credit assignment, encourages hallucination, etc.

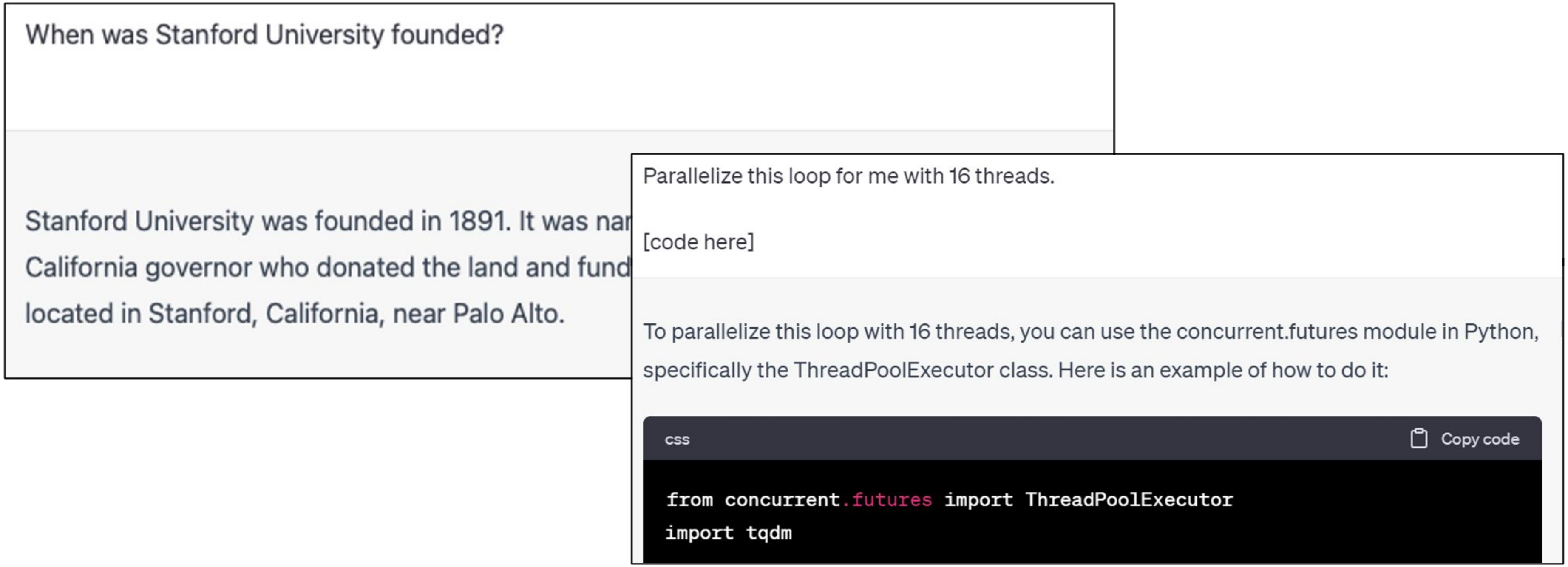
Chung et al. (2022)

As an alternative, what if we **allow models to learn from trial and error**?
Use our best models to sample responses and rely on **human preferences** as sources of rewards. This is called **Reinforcement Learning from Human Feedback**.

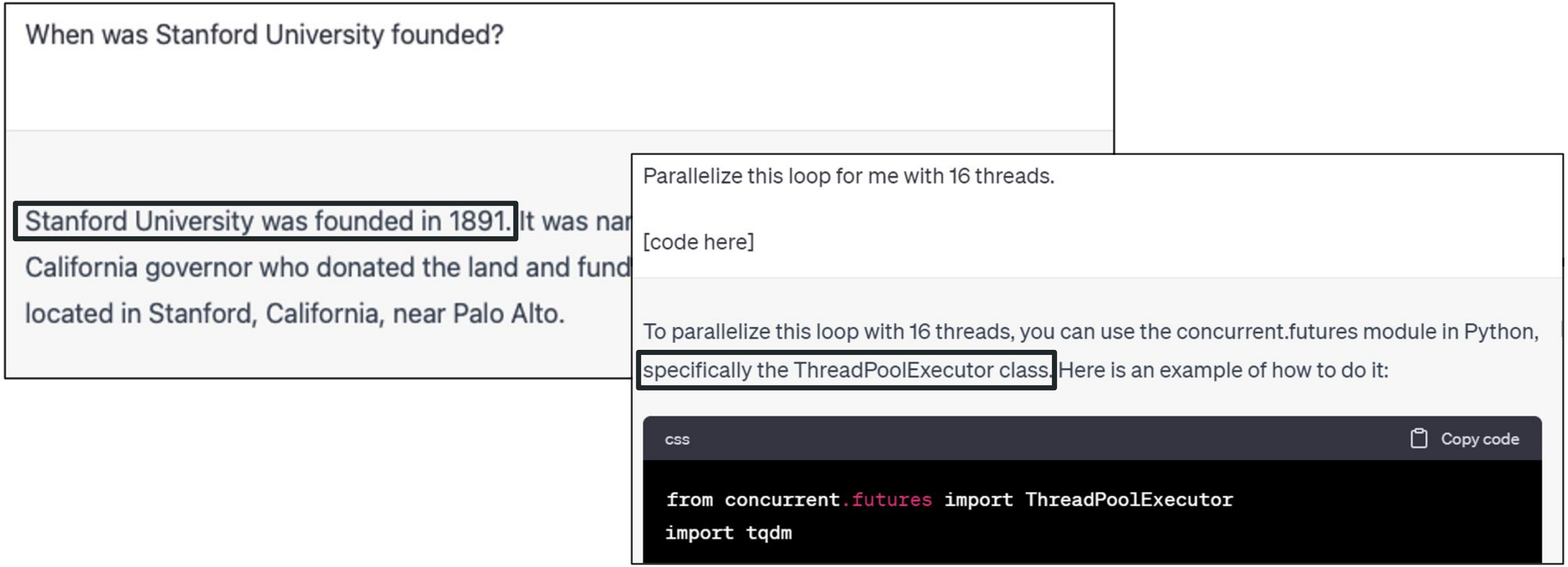


OpenAI (2022)

It's never been easier to prototype impressive AI assistants & demos.



Turning monolithic LMs into reliable AI systems remains challenging.



The Register®

Air Canada must pay damages after chatbot lies to grieving passenger about discount

Airline tried arguing virtual assistant was solely responsible for its own actions

Every AI system will make mistakes.

But the monolithic nature of LMs makes them hard to control, debug, and improve.

To tackle this, AI researchers increasingly build Compound AI Systems,

i.e. modular programs that use LMs as specialized components

Compound AI Systems, *i.e. modular programs that use LMs as specialized components*

Example: Retrieval-Augmented Generation

What compounds protect the digestive system? ⇨ **Monolithic LM** ⇨ **The stomach is protected by gastric acid and proteases.**

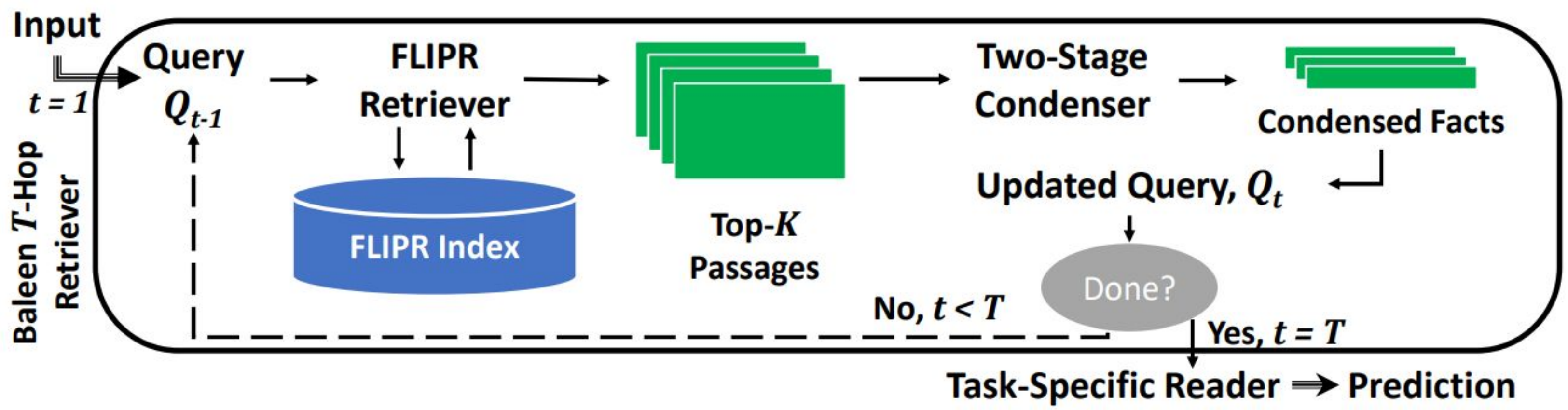
⚡ **Transparency:** can debug traces & offer user-facing attribution

⚡ **Efficiency:** can use smaller LMs, offloading knowledge & control flow

Literature: DrQA (Chen et al., 2017), ORQA (Lee et al., 2019), RAG (Lewis et al., 2020), ColBERT-QA (Khattab et al., 2020)

Compound AI Systems, *i.e. modular programs that use LMs as specialized components*

Example: *Multi-Hop* Retrieval-Augmented Generation



⚡ **Control**: can iteratively improve the system & ground it via tools

Literature: GoldEn (Qi et al., 2019), DecompRC (Min et al., 2019), MDR (Xiong et al., 2020), Baleen (Khattab et al., 2021)

Compound AI Systems, *i.e. modular programs that use LMs as specialized components*

Example: Compositional Report Generation, *i.e. brainstorming an outline, collecting references, etc.*

“The ColBERT retriever”

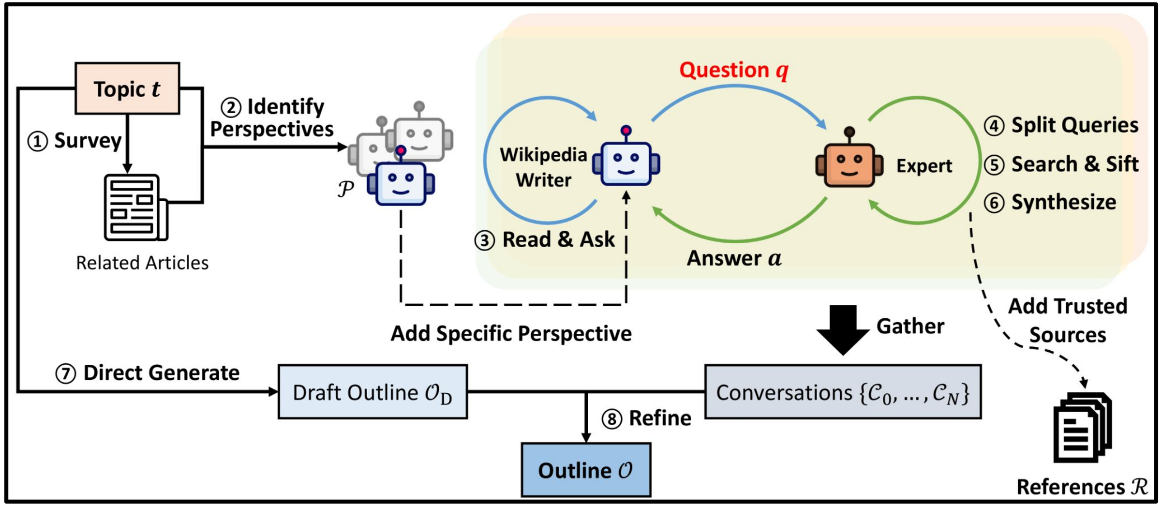
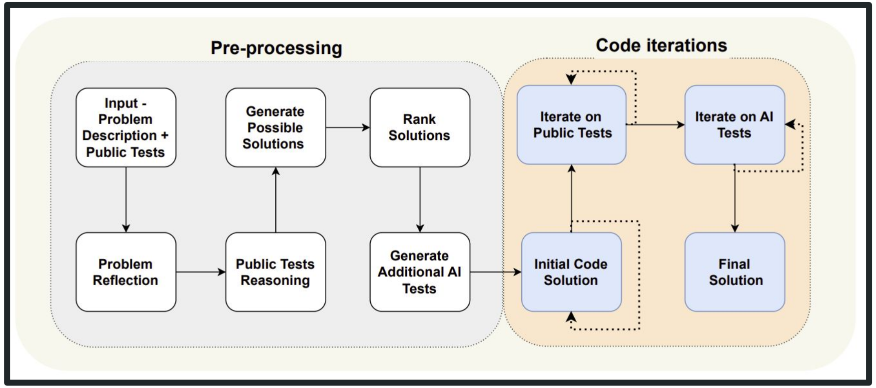


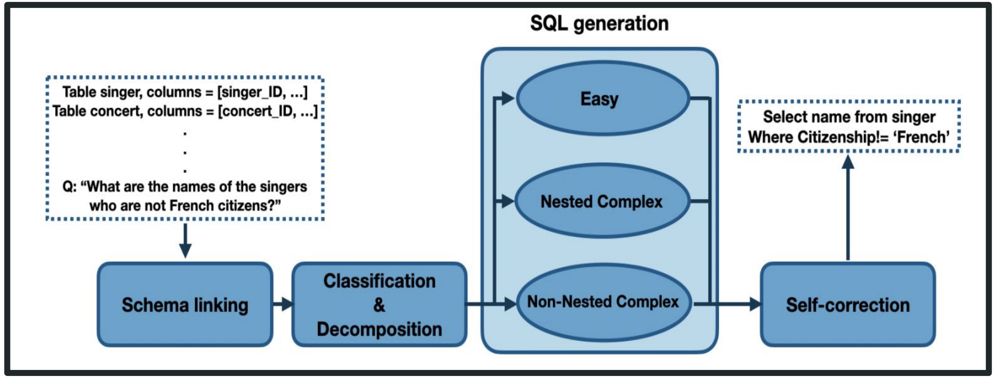
Table of contents

⚡ **Quality**: more reliable composition of better-scoped LM capabilities

STORM: Assisting in Writing Wikipedia-like Articles From Scratch with Large Language Models (Shao et al., 2024)

Compound AI Systems, *i.e. modular programs that use LMs as specialized components*



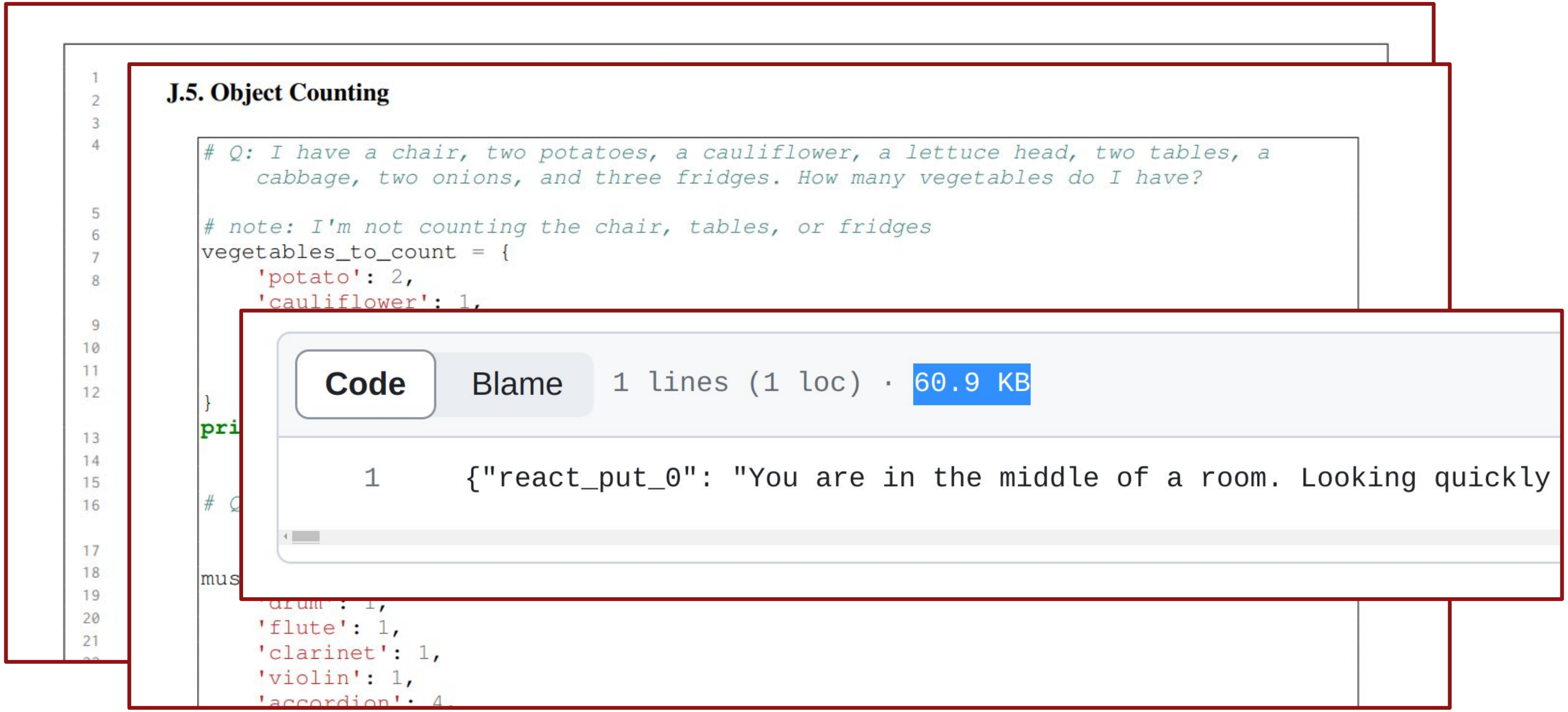


+ Task-agnostic prompting strategies, e.g. Best-of-N, Chain Of Thought, Program of Thought, ReAct, Reflexion, Archon, …

⚡ **Inference-time Scaling**: systematically searching for better outputs

Literature: AlphaCodium (Ridnik, 2024), DIN-SQL (Pourreza & Rafiei, 2023), RARR (Gao et al., 2023), and many others

Unfortunately, LMs are highly sensitive to how they're instructed to solve tasks, so under the hood…



Unfortunately, LMs are highly sensitive to how they're instructed to solve tasks, so under the hood…

Each "prompt" couples five very different roles:

1. The core *input → output* behavior, a **Signature**.
2. The computation specializing an inference-time strategy to the signature, a **Predictor**.
3. The computation formatting the signature's inputs and parsing its typed outputs, an **Adapter**.
4. The computations defining objectives and constraints on behavior, **Metrics** and **Assertions**.
5. The process of finding the strings & weights that teach LMs desired behavior, an **Optimizer**.

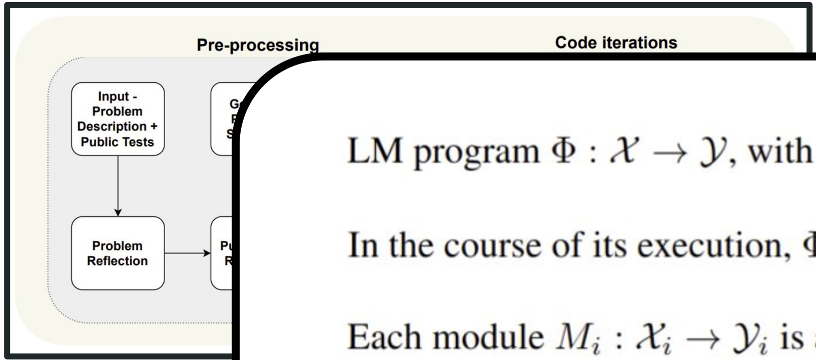
Existing Compound AI Systems are modular in principle, but are too "stringly-typed":
they couple the fundamental system architecture with incidental choices
not portable to new LMs, objectives, or pipelines.

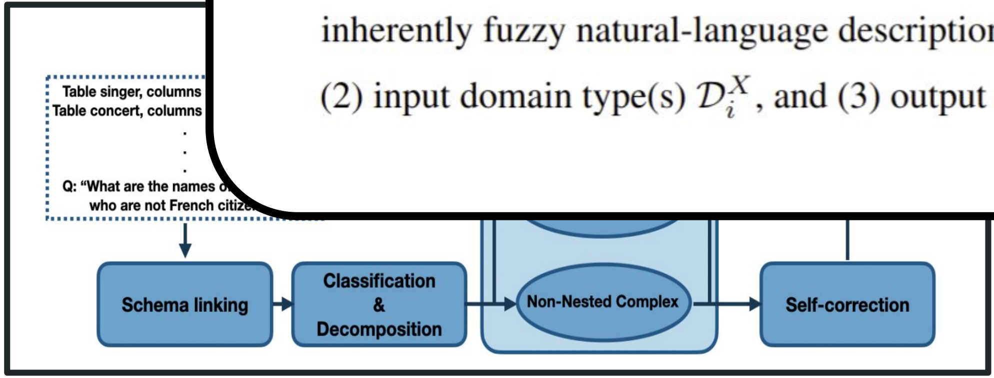
We *know* how to build controllable systems & improve them modularly.

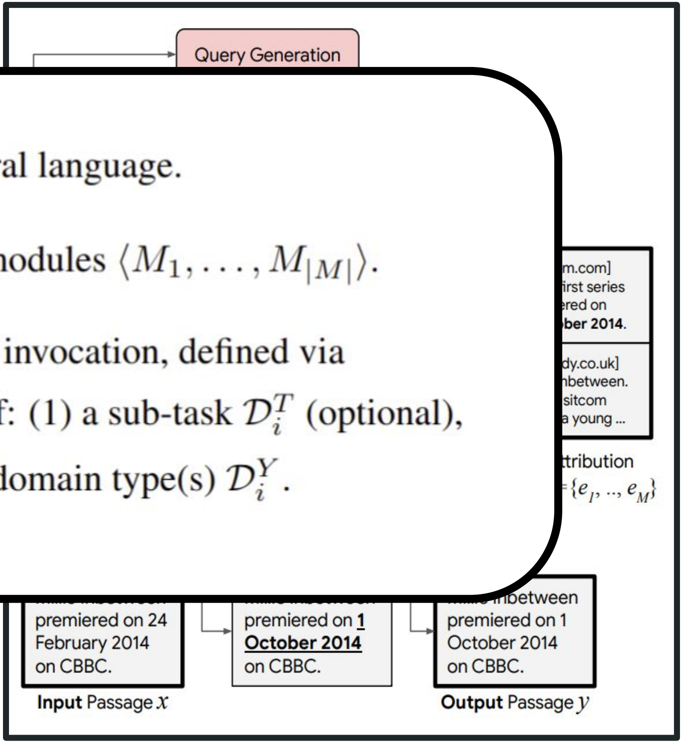
That is called...

What if we could abstract Compound AI Systems as programs with fuzzy natural-language-typed modules that learn their behavior?

DSPy







LM program $\Phi : \mathcal{X} \to \mathcal{Y}$, with $\mathcal{X}$ and $\mathcal{Y}$ in natural language.

In the course of its execution, Φ makes calls to modules $\langle M_1, \ldots, M_{|M|} \rangle$.

Each module $M_i : \mathcal{X}_i \to \mathcal{Y}_i$ is a *declarative* LM invocation, defined via inherently fuzzy natural-language descriptions of: (1) a sub-task $\mathcal{D}_i^T$ (optional), (2) input domain type(s) $\mathcal{D}_i^X$, and (3) output co-domain type(s) $\mathcal{D}_i^Y$.

```python
fact_checking = dspy.ChainOfThought('claims -> verdicts: list[bool]')
fact_checking(claims=["Python was released in 1991.", "Python is a compiled language."])
```

```
Prediction(
    reasoning='The first claim states that "Python was released in 1991," which is true.
Python was indeed first released by Guido van Rossum in February 1991. The second claim s
tates that "Python is a compiled language." This is false; Python is primarily an interpr
eted language, although it can be compiled to bytecode, it is not considered a compiled l
anguage in the traditional sense like C or Java.',
    verdicts=[True, False]
)
```

For each module M_i, determine the:

1. String prompt Π_i in which inputs $\mathcal{X}_i$ are plugged in.
2. Weights Θ_i assigned to the LM.

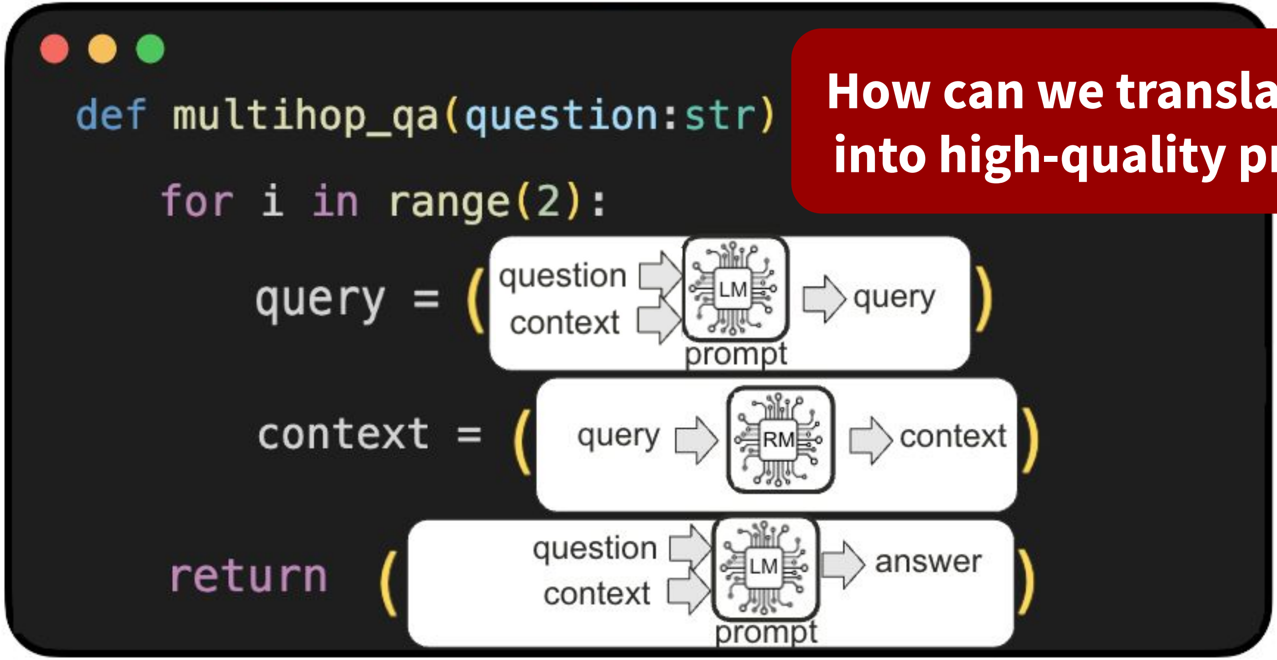
in the optimization problem defined by:

$$\arg\max_{\Theta,\Pi} \frac{1}{|X|} \sum_{(x,m)\in X} \mu(\Phi_{\Theta,\Pi}(x), m)$$

given a small training set $X = \{(x_1, m_1), \ldots, (x_{|X|}, m_{|X|})\}$ and a metric $\mu : \mathcal{Y} \times \mathcal{M} \to \mathbb{R}$ for labels or hints $\mathcal{M}$.

This is hard. We don't have gradients or intermediate labels to optimize each module! How should we go about this?

As an example, let's say we wanted to build this simple pipeline for *multi-hop retrieval-augmented generation*



How can we translate these into high-quality prompts?

First, modules are translated into basic prompts using Adapters and Predictors.

```
self.generate_query = dspy.ChainOfThought("context, question -> query")
```

```
dspy.Adapter(self.generate_query)
```

Predefined *Adapters* are used to translate modules into basic prompts

```
Given the fields “context” and “question”, respond with the field “query”.

Follow the following format:
Context: <context>
Question: <question>
Reasoning: Let’s think step by step to <..>
Query: <query>
```

Then, Prompt Optimizers (or RL) can tune the modules

i.e., tune the prompts and/or weights for all modules in your program

```
Given the fields “context” and “question”, respond with the field “query”.

Follow the following format:
Context: <context>
Question: <question>
Reasoning: Let’s think step by step to <..>
Query: <query>
```

Program Score: 37%

```
optimizer = MIPROv2(metric=..., trainset=...)
optimized_program = optimizer.compile(program)
```

```
Carefully read the provided `context` and `question`. Your task is to formulate a concise
and relevant `query` that could be used to retrieve information from a search engine to
answer the question most effectively. The `query` should encapsulate...

Follow the following format:
Context: <context>
Question: <question>
Reasoning: Let’s think step by step to <..>
Query: <query>

Here are some examples: <...>
```

Program Score: 55%

Instead of tweaking brittle prompts...

Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be three types:

(1) Search[entity], which searches the exact entity on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to search.

(2) Lookup[keyword], which returns the next sentence containing keyword in the current passage.

(3) Finish[answer], which returns the answer and finishes the task.

Here are some examples.

Question: What is the elevation range for the area that the eastern sector of the Colorado orogeny extend
Thought 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado oroge
elevation range of the area.
Action 1: Search[Colorado orogeny]
Observation 1: The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and
Thought 2: It does not mention the eastern sector. So I need to look up eastern sector.
Action 2: Lookup[eastern sector]
Observation 2: (Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Pla

[... truncated …]

Scores

33%

with **GPT-3.5**

on a multi-hop QA task

Multi-Hop Retrieval-Augmented Generation (HotPotQA)

Program	Optimized	GPT 3.5	Llama2-13b-Chat

Multi-Hop Retrieval-Augmented Generation (HotPotQA)

Program	Optimized	GPT 3.5	Llama2-13b-Chat
`dspy.Predict("question -> answer")`	❌	34.3	27.5
`dspy.RAG (with CoT)`	❌	36.4	34.5
	✅	42.3	38.3
`MultiHop`	❌	36.9	34.7
	✅	**54.7**	**50.0**

Compiling `MultiHop` into a small LM (T5-770M) with dspy.BootstrapFinetune, starting from 200 answers, scores 39%

Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs

Krista Opsahl-Ong[1*], Michael J Ryan[1*], Josh Purtell[2], David Broman[3], Christopher Potts[1], Matei Zaharia[4], Omar Khattab[1]

[1]Stanford University, [2]Basis, [3]KTH Royal Institute of Technology [4]UC Berkeley

Slides adapted from
Krista Opsahl-Ong & Michael Ryan

Fine-Tuning and Prompt Optimization: Two Great Steps that Work Better Together

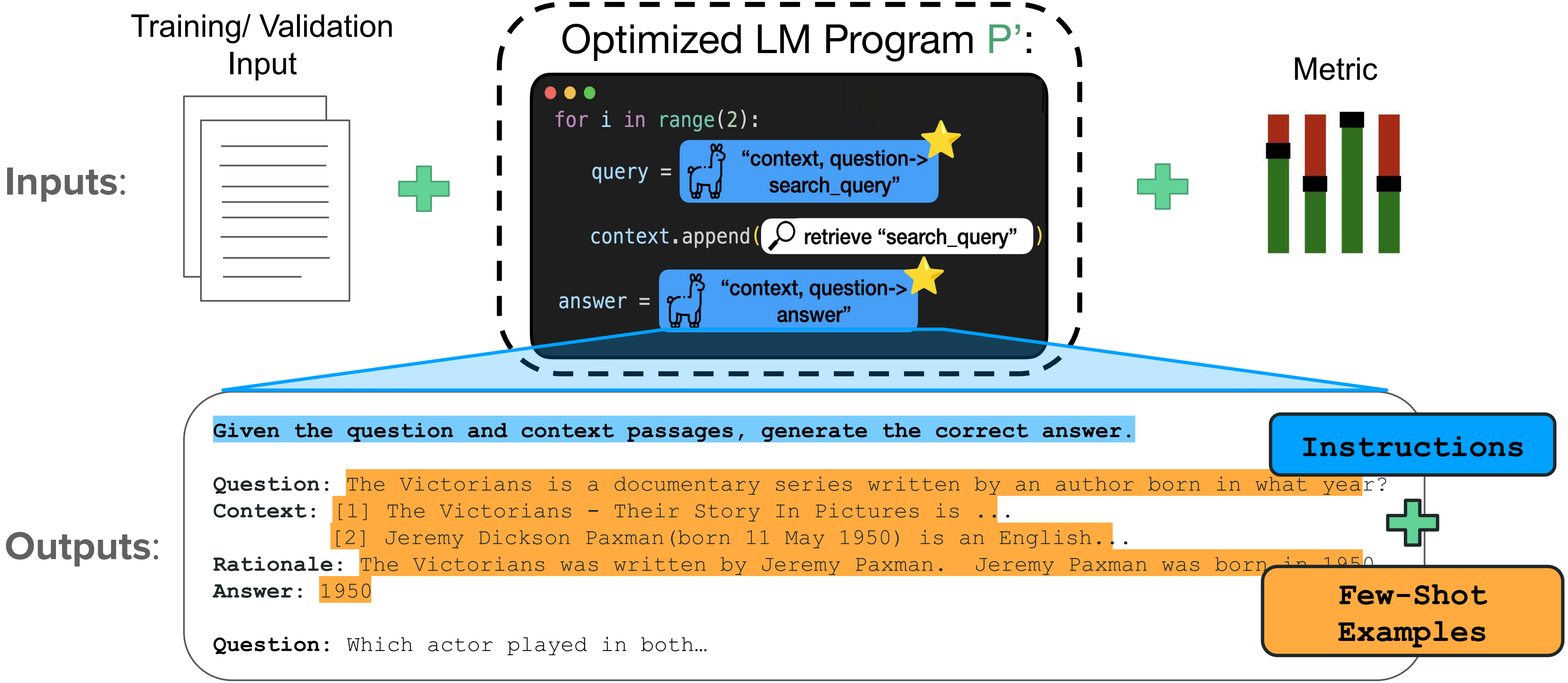
Dilara Soylu **Christopher Potts** **Omar Khattab**

Stanford University

GROUNDING BY TRYING: LLMS WITH REINFORCEMENT LEARNING-ENHANCED RETRIEVAL

Sheryl Hsu[1], Omar Khattab[1,2], Chelsea Finn[1,3] & Archit Sharma[1,4]
[1]Stanford University,[2]Databricks,[3]Physical Intelligence,[4]Google DeepMind
{sherylh,architsh}@stanford.edu

Problem Setting



Constraints / Assumptions

1. **No access to log-probs or model weights**: Developers may want to optimize LM programs for use on API only models.

2. **No intermediate metrics / labels**: We assume no access to manual ground-truth labels for intermediate stages.

3. **Budget-Conscious**: We want to limit the number of input examples we require and the number of program calls we make.

Methods

1. Bootstrap Few-shot
2. Extending OPRO
3. MIPRO

Methods

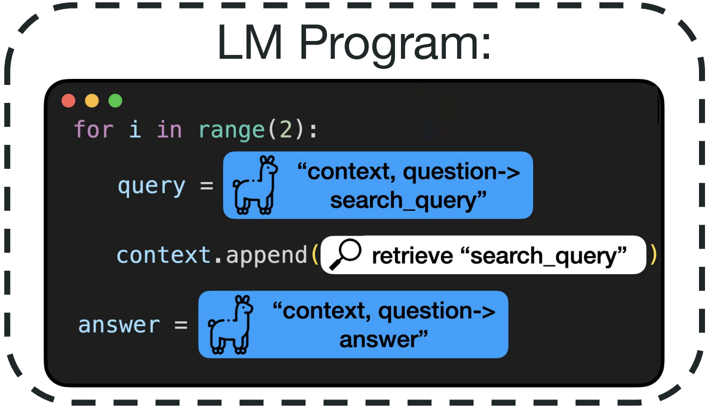
1. Bootstrap Few-shot
2. Extending OPRO
3. MIPRO

💡 *Bootstrap Few-shot examples with simple rejection sampling*

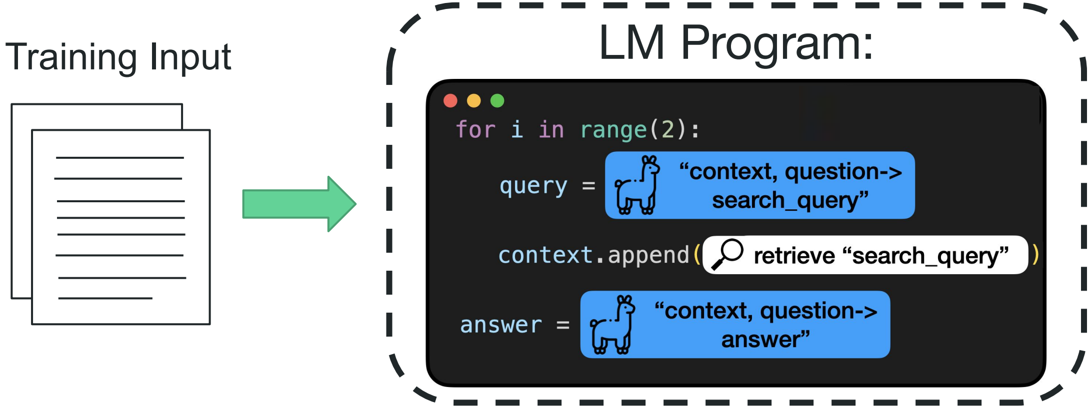
Bootstrap Few-Shot Examples

O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, C. Potts “DSPY: COMPILING DECLARATIVE LANGUAGE MODEL CALLS INTO SELF-IMPROVING PIPELINES”

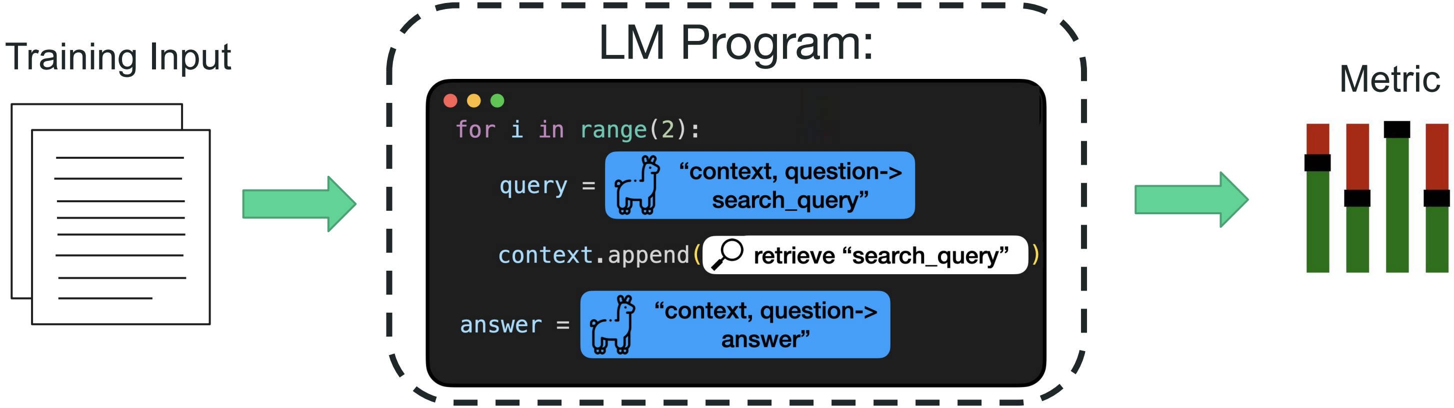
Bootstrap Few-Shot Examples



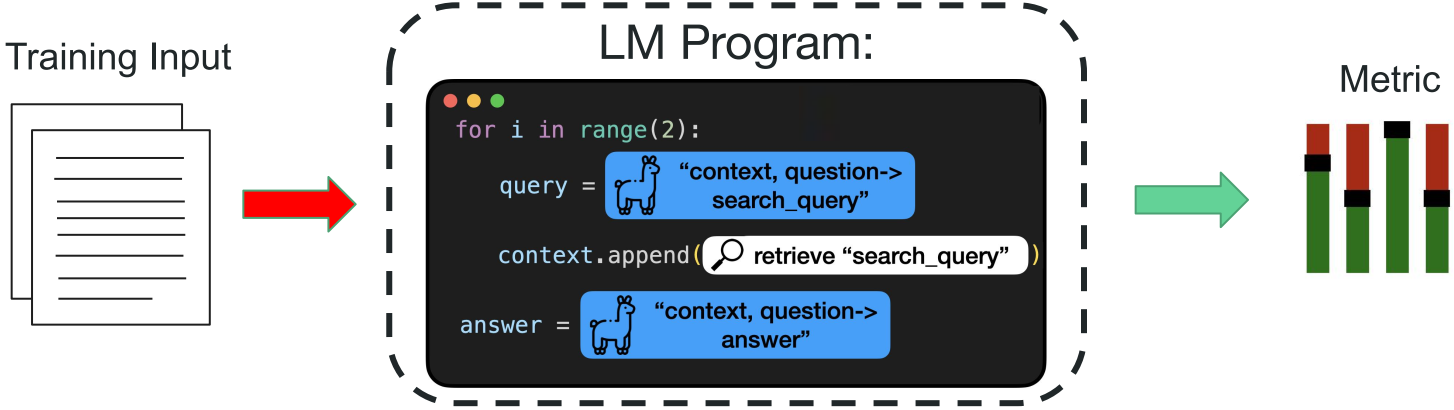
Bootstrap Few-Shot Examples



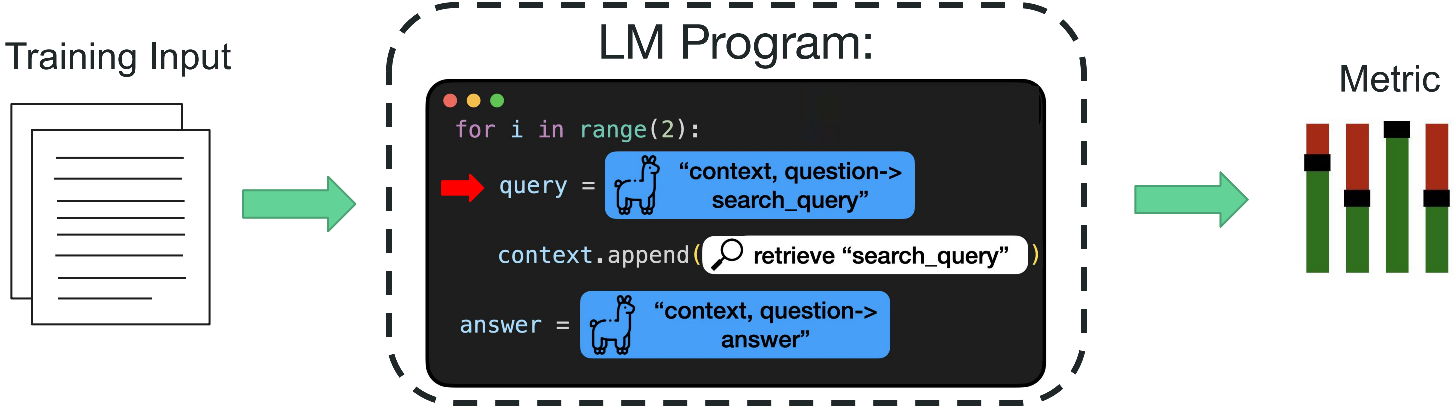
Bootstrap Few-Shot Examples



Bootstrap Few-Shot Examples

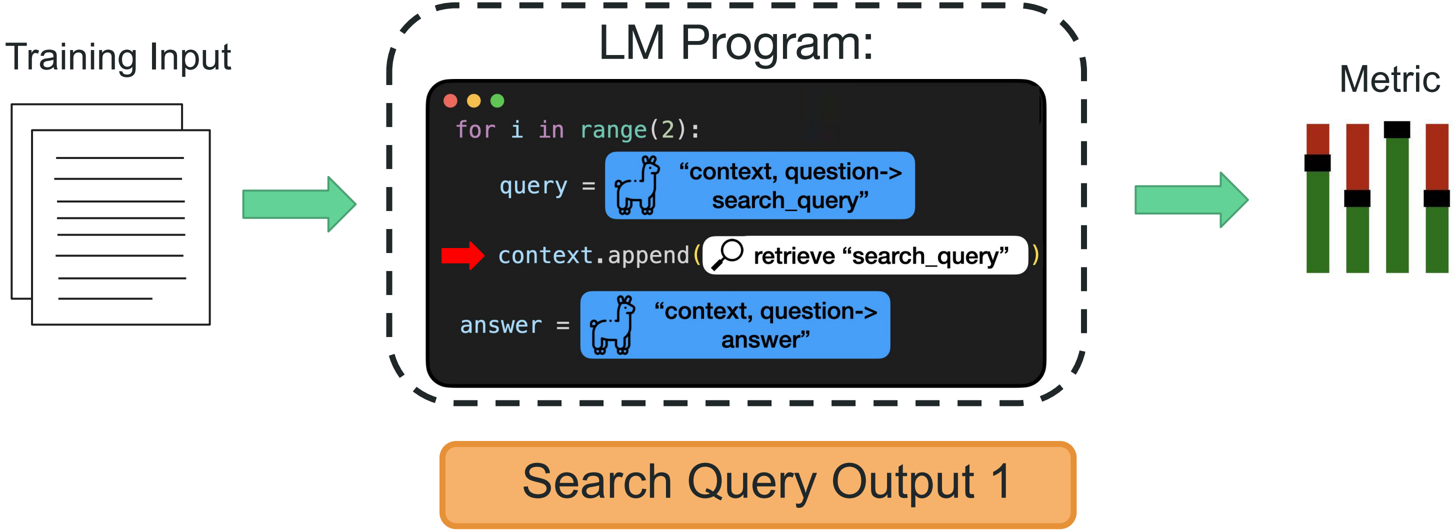


Bootstrap Few-Shot Examples

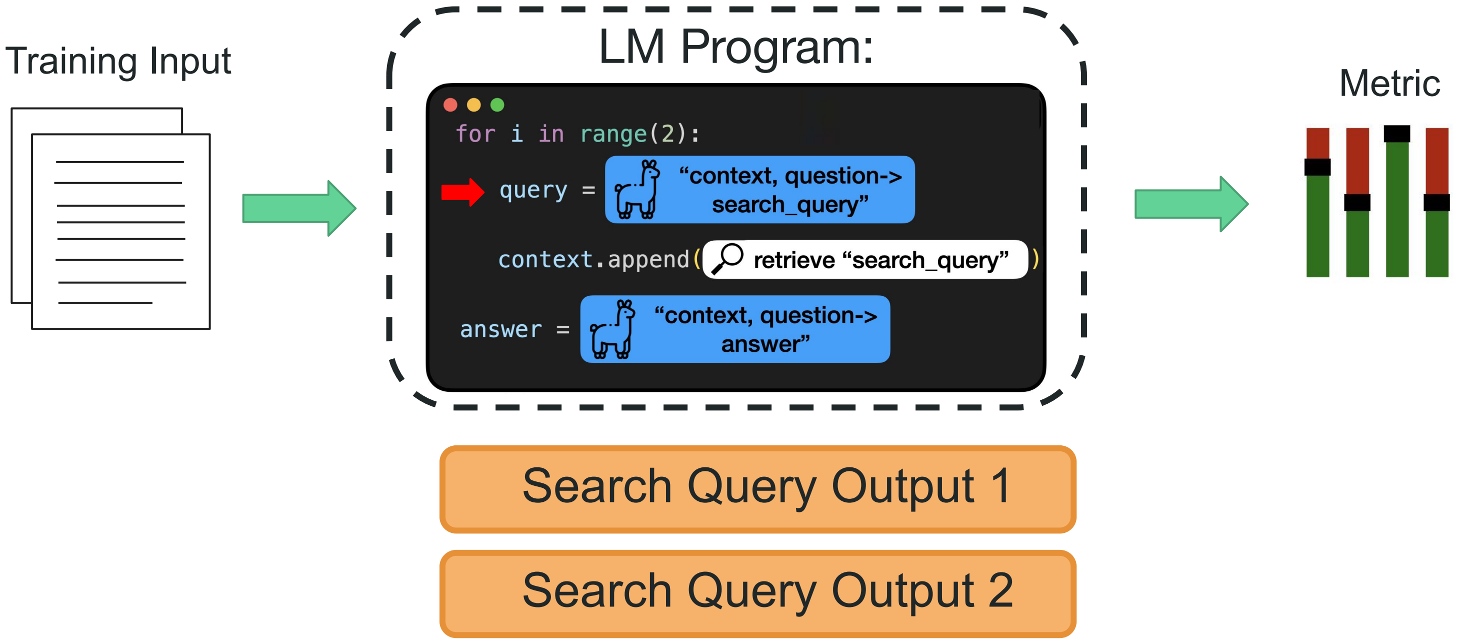


Search Query Output 1

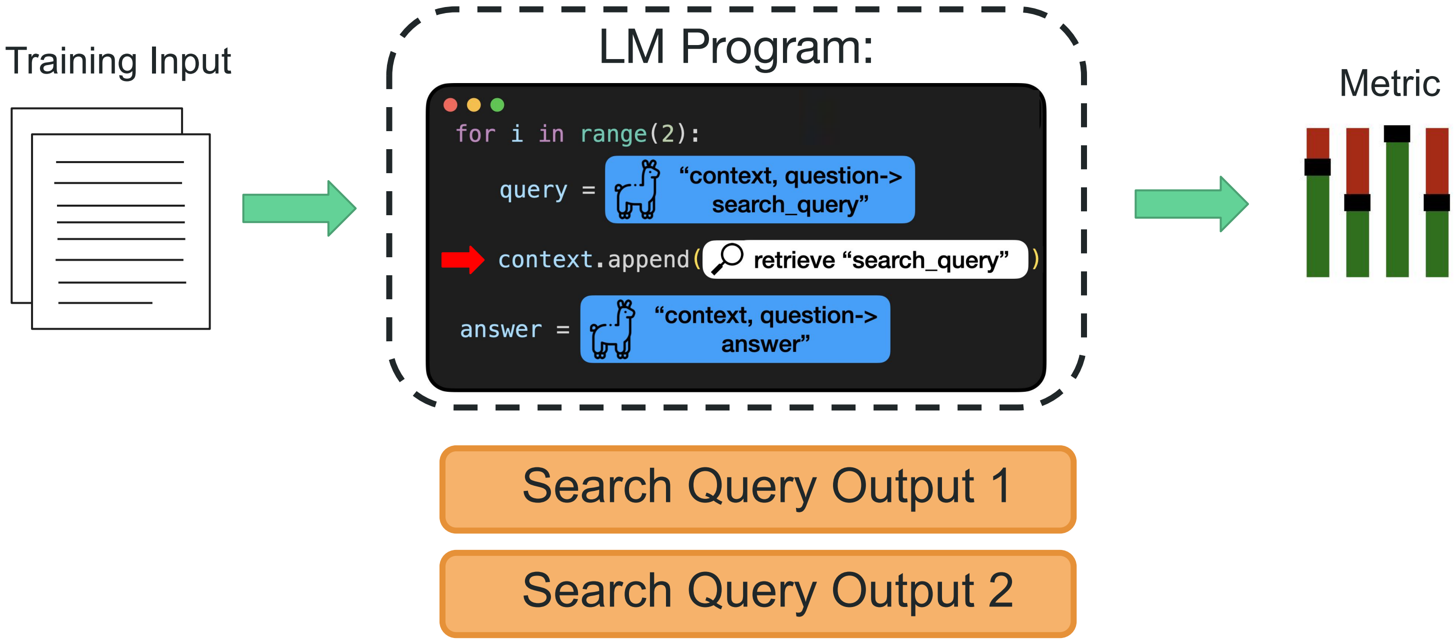
Bootstrap Few-Shot Examples



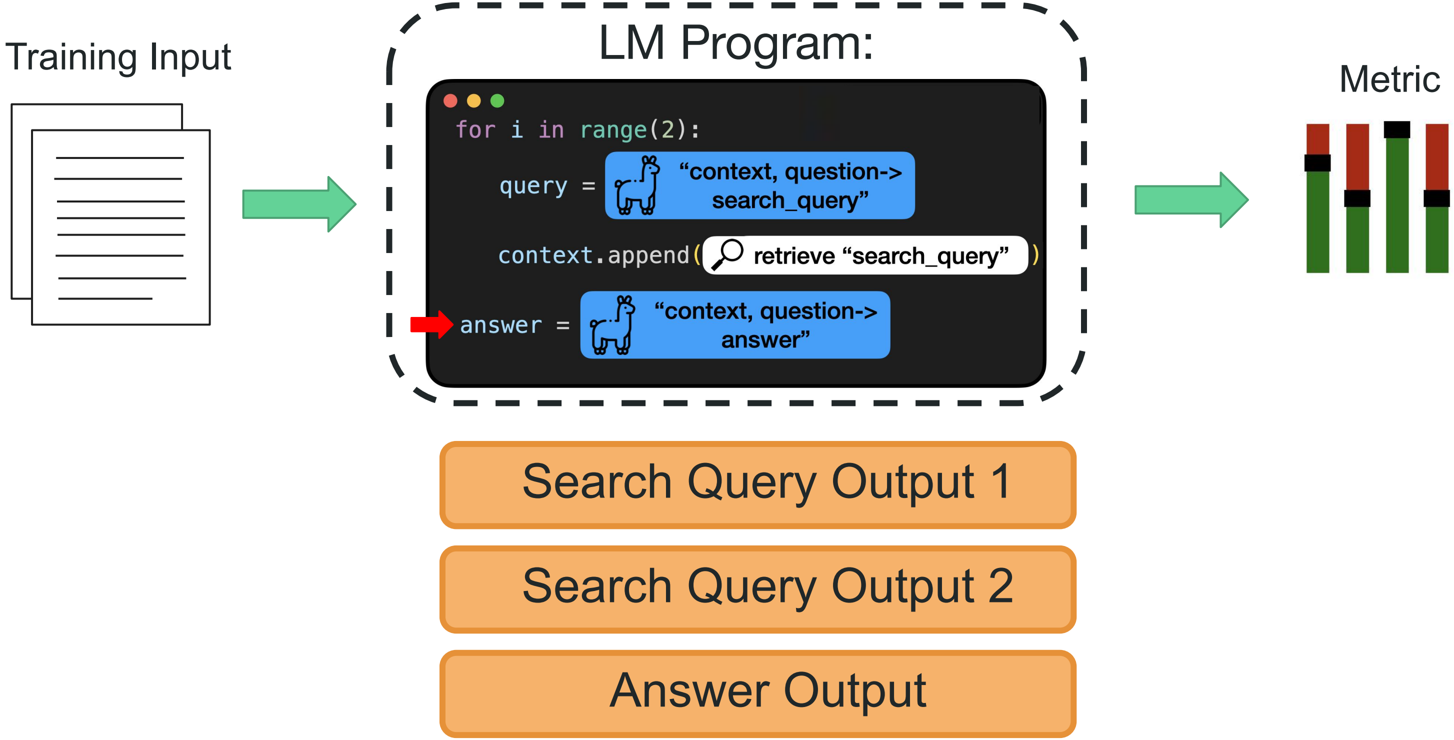
Bootstrap Few-Shot Examples



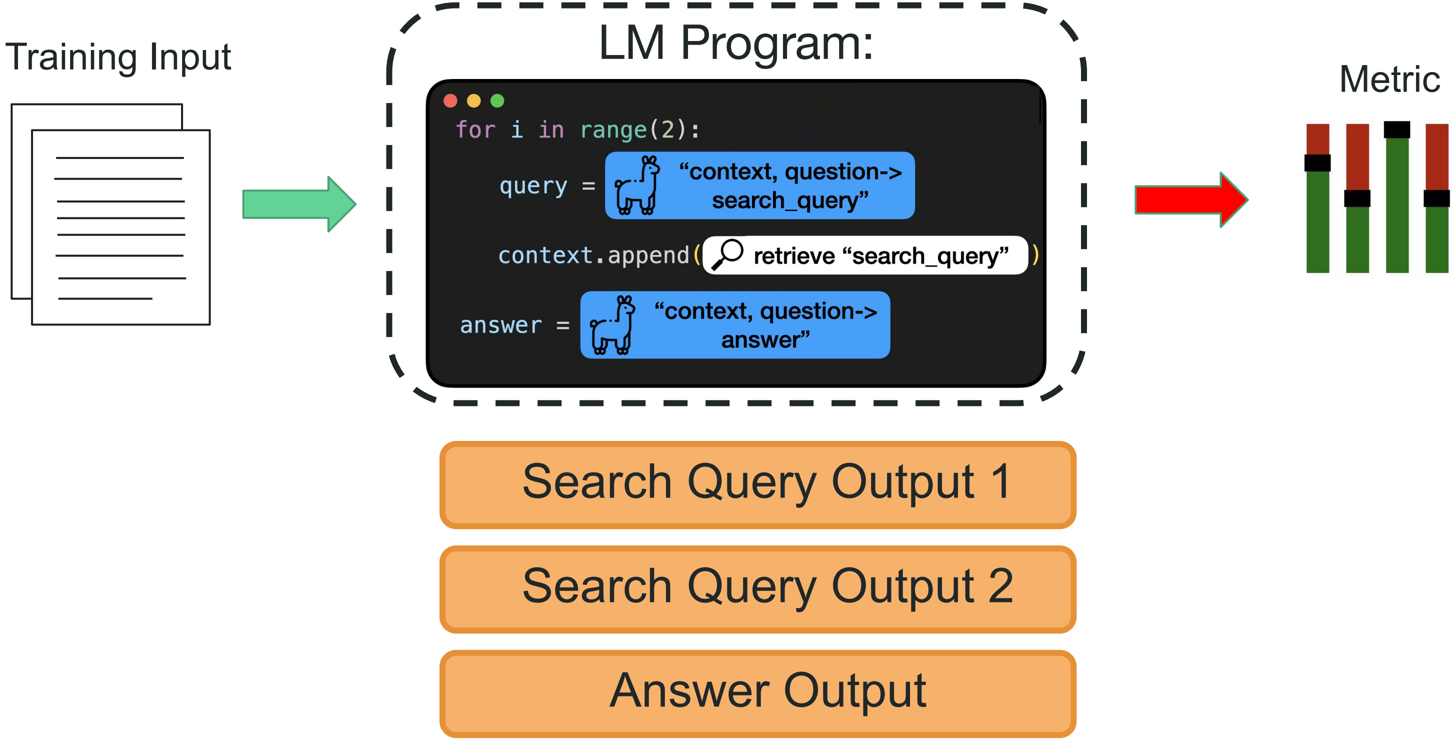
Bootstrap Few-Shot Examples



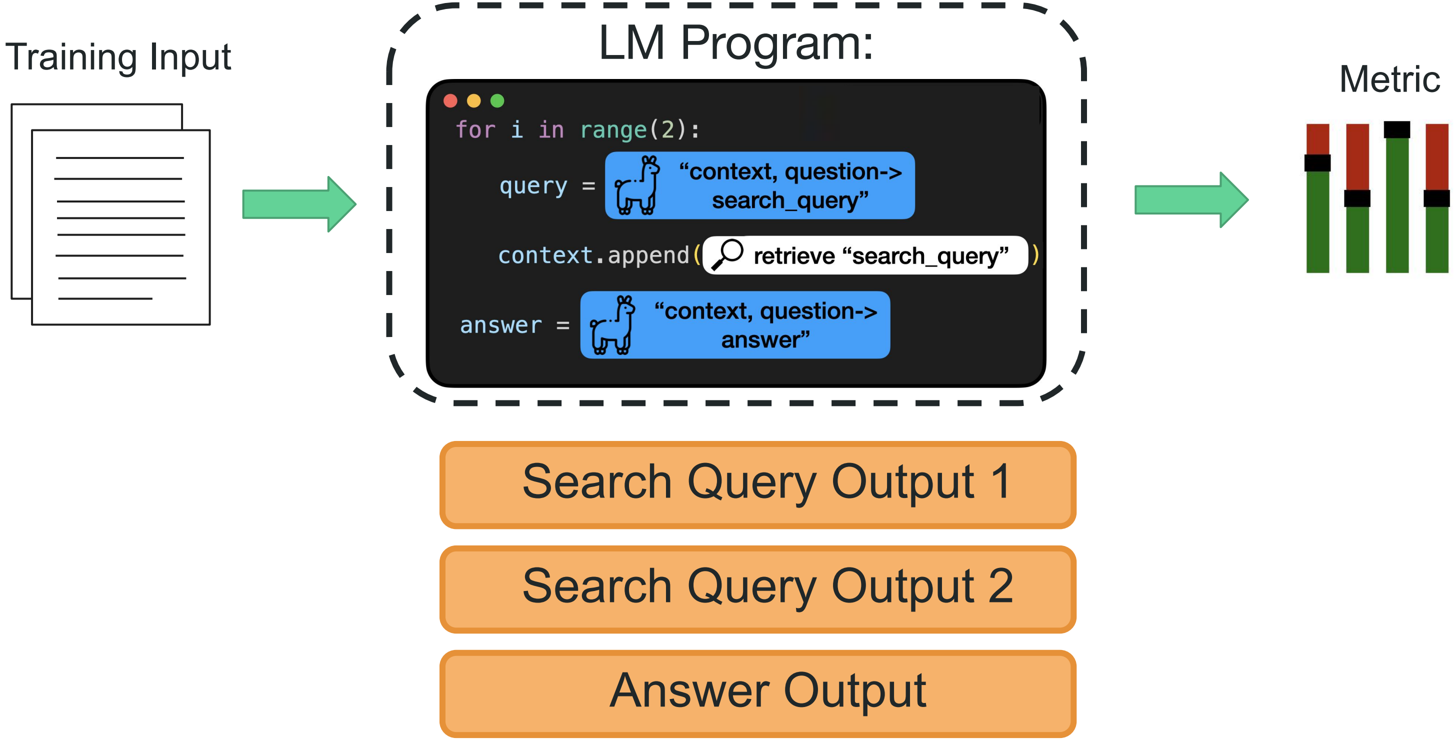
Bootstrap Few-Shot Examples



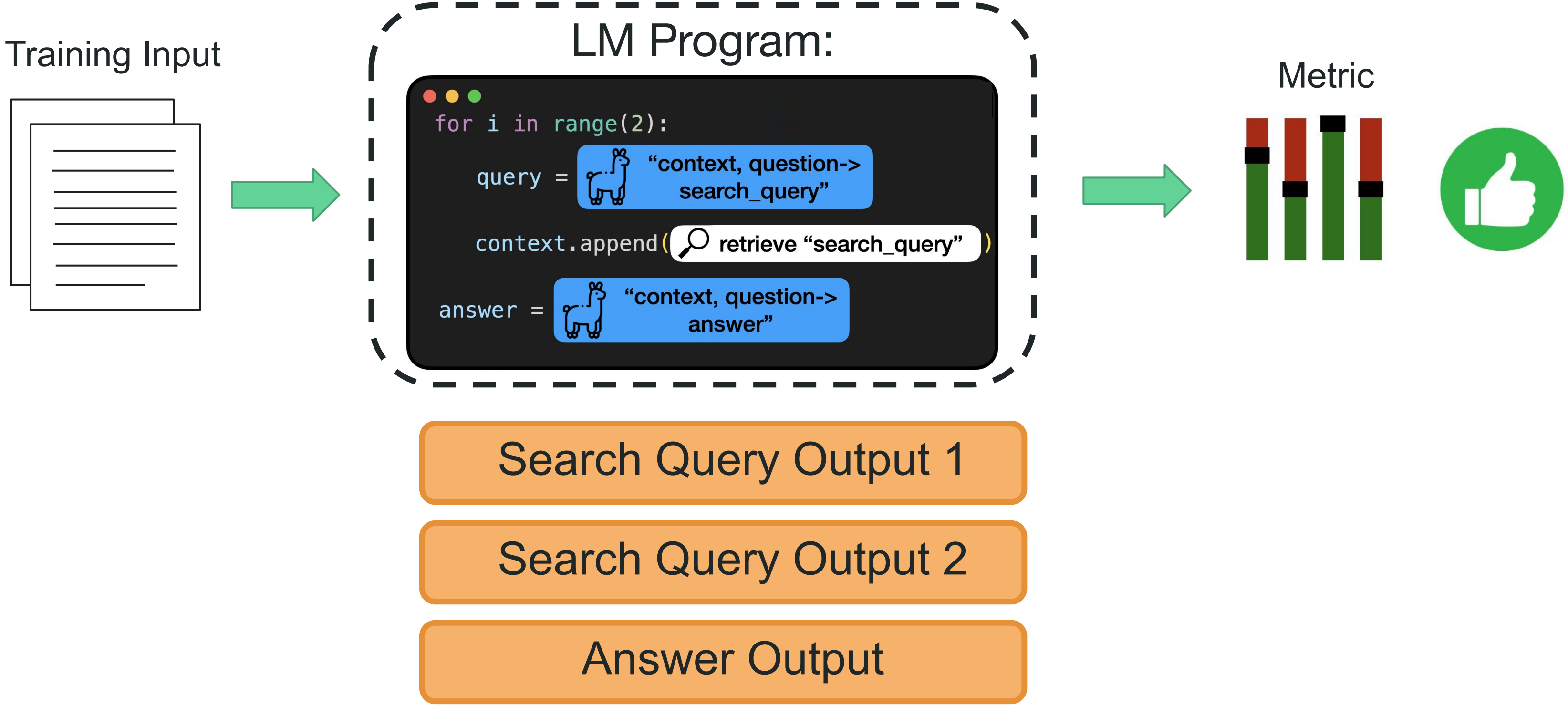
Bootstrap Few-Shot Examples



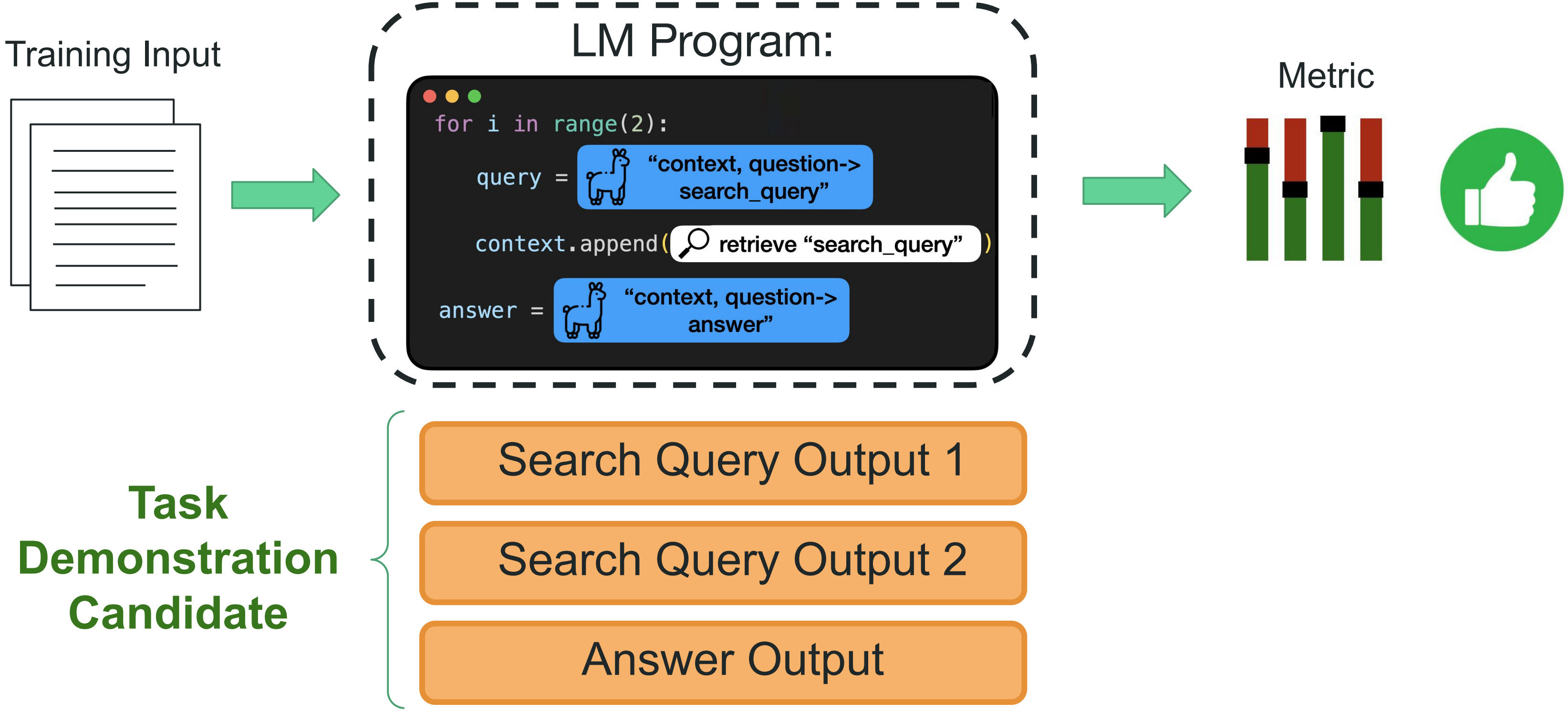
Bootstrap Few-Shot Examples



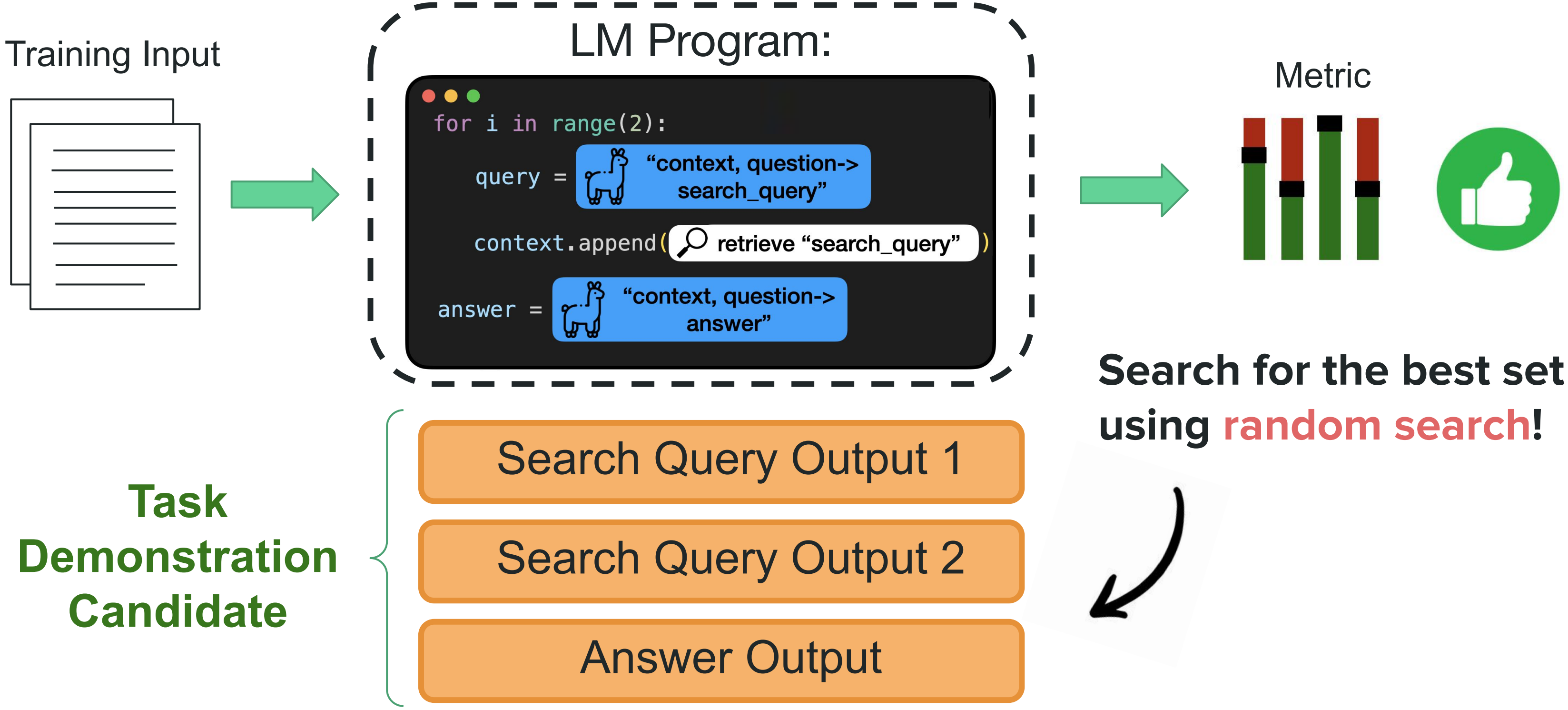
Bootstrap Few-Shot Examples



Bootstrap Few-Shot Examples



Bootstrap Few-Shot (w/ Random Search)



Bootstrap Few-Shot (w/ Random Search)

Given the context passages and a question, generate the correct answer.

Context: [1] The Victorians - Their Story In Pictures is ...
[2] Jeremy Dickson Paxman(born 11 May 1950) is an English...
Question: The Victorians is a documentary series written by an author born in what year?
Rationale: The Victorians was written by Jeremy Paxman. Jeremy Paxman was born in 1950.
Answer: 1950

...

Task Demonstration Candidate

- Search Query Output 1
- Search Query Output 2
- Answer Output

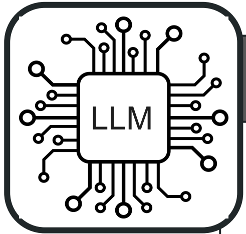
Methods

1. Bootstrap Few-shot
2. Extending OPRO
3. MIPRO

💡 *Extend existing instruction opt. method (OPRO) to multi-stage*

What is OPRO? Optimization through Prompting

“Proposer LM”



Prompt Proposals

“Think step by step”

“Take a deep breath and think step by step”

“Carefully solve the problem”

“Let’s do the math”

Evaluate

Propose More Prompts

Given prompts/scores propose more prompts.

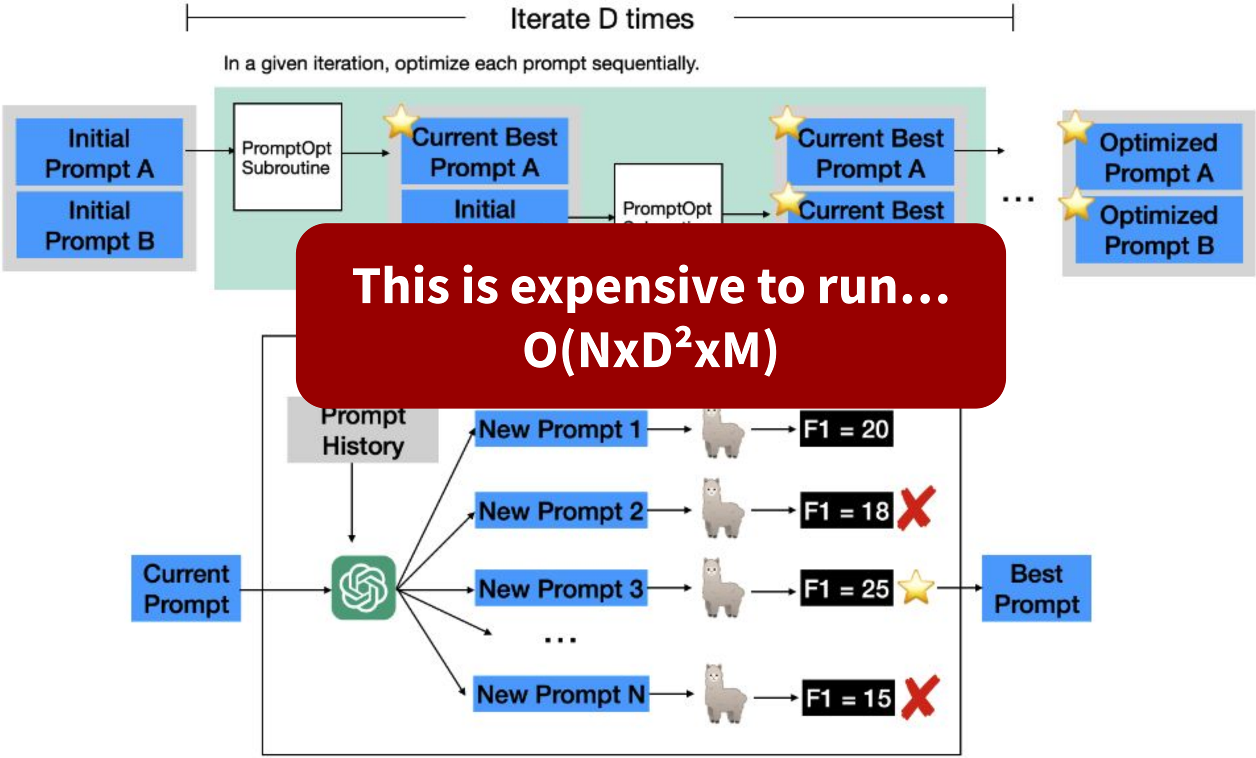
“Think step by step”
Score: 31

“Take a deep breath and think step by step”
Score: 42

C. Yang*, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, X. Chen* “Large Language Models as Optimizers”

Initial extension to multi-stage: CA-OPRO

Coordinate-Ascent OPRO



This is expensive to run…

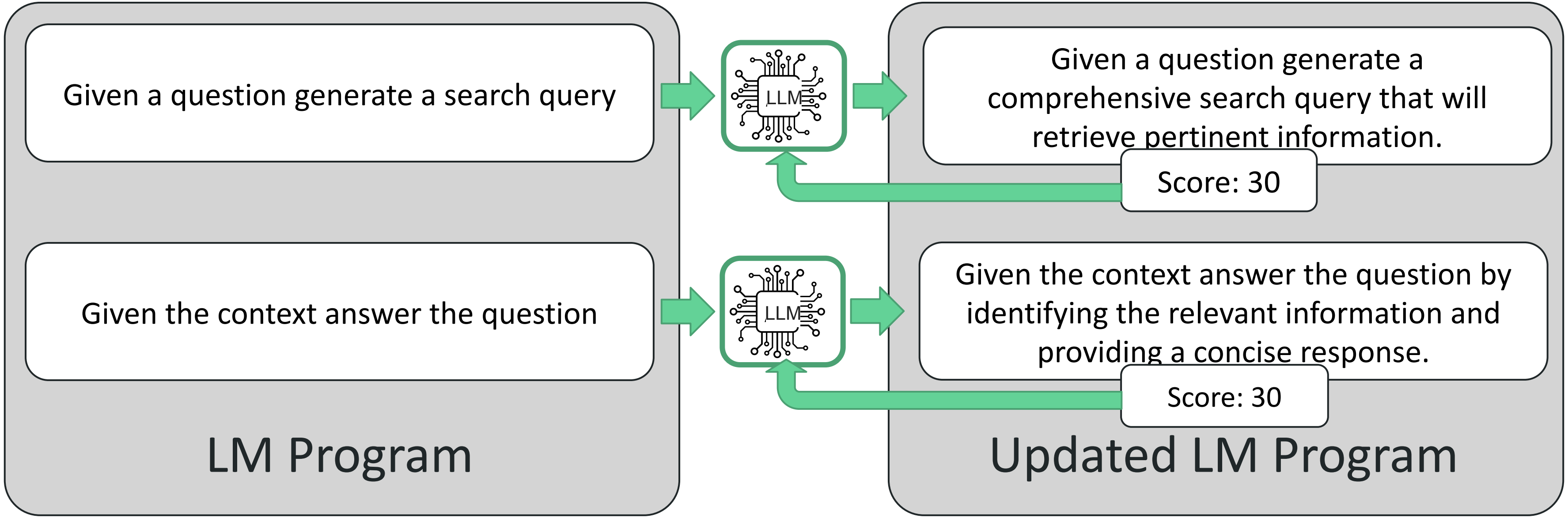
$O(NxD^2xM)$

Module-Level OPRO

Key Idea: Coordinate-Ascent was expensive, maybe we don't need explicit credit assignment? Let's just change both prompts at a time in parallel!

Module-Level OPRO

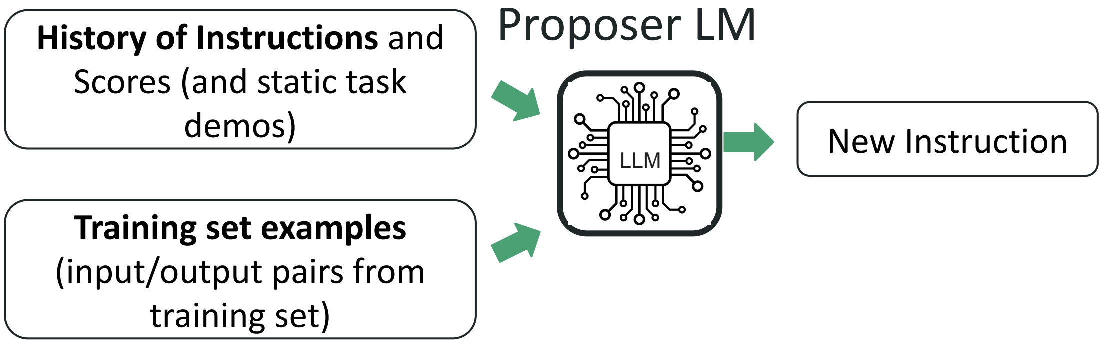
Key Idea: Coordinate-Ascent was expensive, maybe we don't need explicit credit assignment? Let's just change both prompts at a time in parallel!



Score: 30

Finally, Grounding!

Hypothesis: Providing our proposer LM with more information relevant to the task can help us propose better instructions.



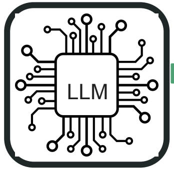
Finally, Grounding!

Key idea: What if we built a multi-stage LM program to bootstrap and synthesize information about the task for use in instruction proposal?

History of Instructions and Scores (and static task demos)

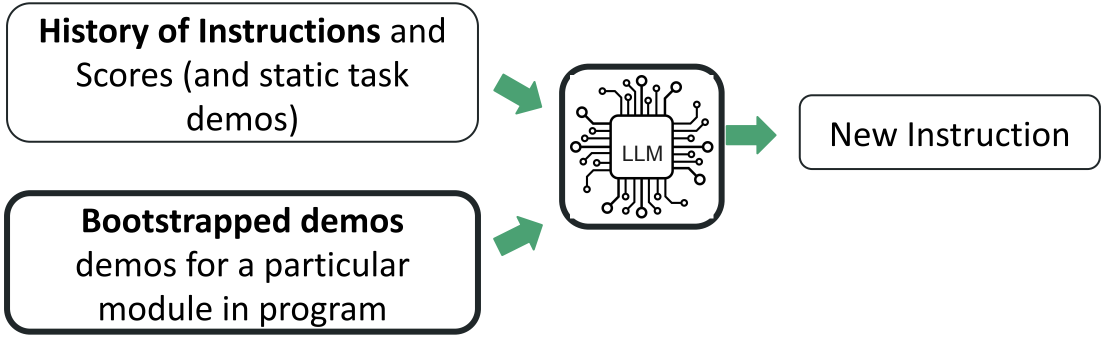
Training set examples (input/output pairs from training set)

Proposer LM



New Instruction

Finally, Grounding!



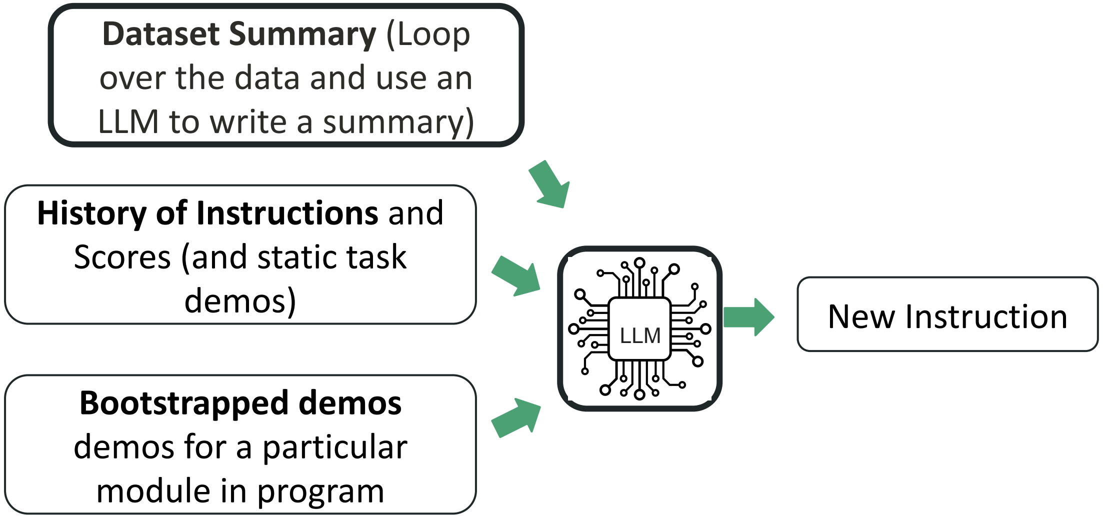
Bootstrapped demo example:

Question: The Victorians - Their Story In Pictures is a documentary series written by an author born in what year?

Reasoning: Let's think step by step in order to find the search query. We need to find the author's birth year. We can search for the author's name along with the phrase "birth year" or "birthday" to get the desired information.

Search Query: "author of The Victorians - Their Story In Pictures birth year" or "author of The Victorians - Their Story In Pictures birthday"

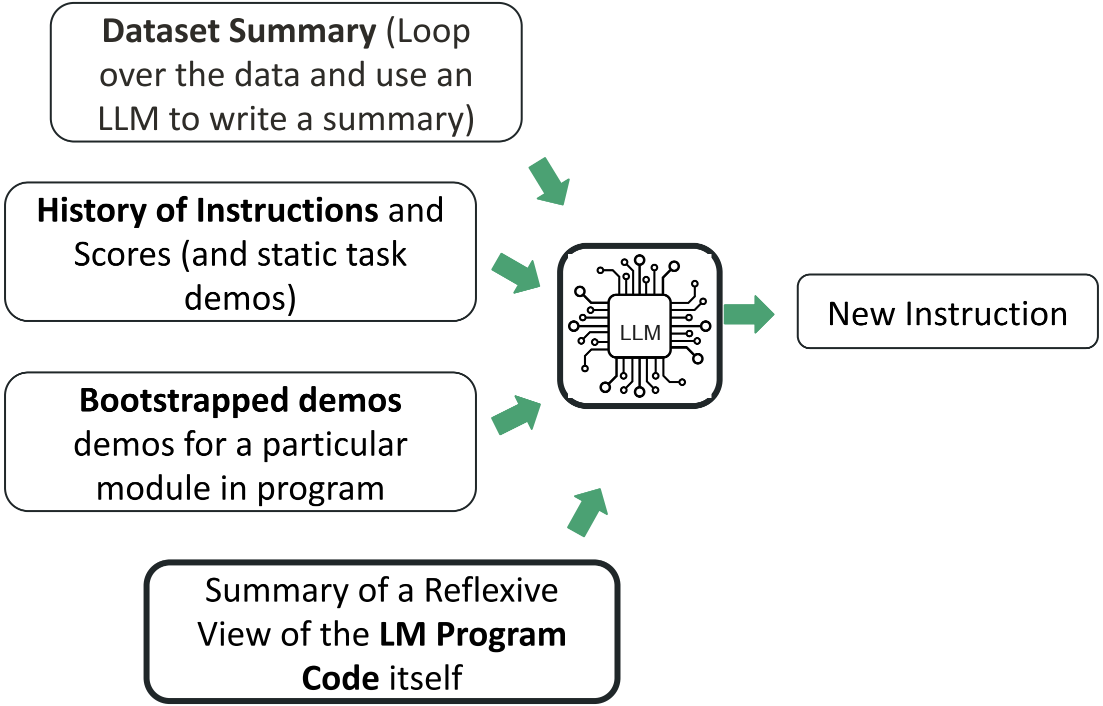
Finally, Grounding!



Dataset summary example:

"The dataset **consists of factual, trivia-style questions** across a wide range of topics, presented in a clear and concise manner. These questions are likely designed for use in trivia games..”

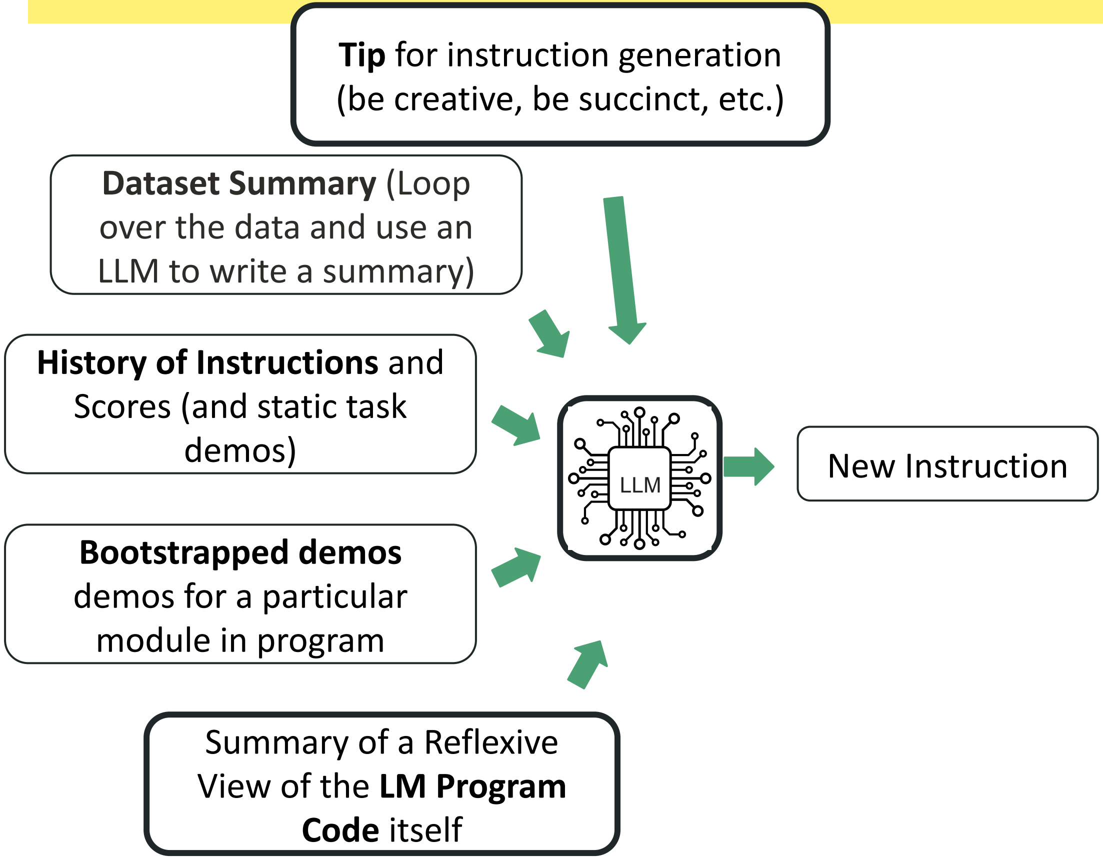
Finally, Grounding!



Program Summary example:

"The program code appears to be **designed to answer complex questions by retrieving and processing information from multiple sources** or passages. In this case, the program is set up for two hops, … The **module `self.generate_query` in this program is responsible for generating a search query** based on the context and question provided."

Finally, Grounding!



Tip example:

"Don't be afraid to be creative when generating the new instruction"

"Keep the instruction clear and concise."

"Make sure your instruction is very informative and descriptive."

Methods

1. Bootstrap Few-shot
2. Extending OPRO
3. MIPRO

💡 *Co-optimize instructions & few-shot examples efficiently*

MIPRO works in 3 steps:

Multi-prompt Instruction PRoposal Optimizer

Prompt Proposal

1. Bootstrap Task Demonstrations

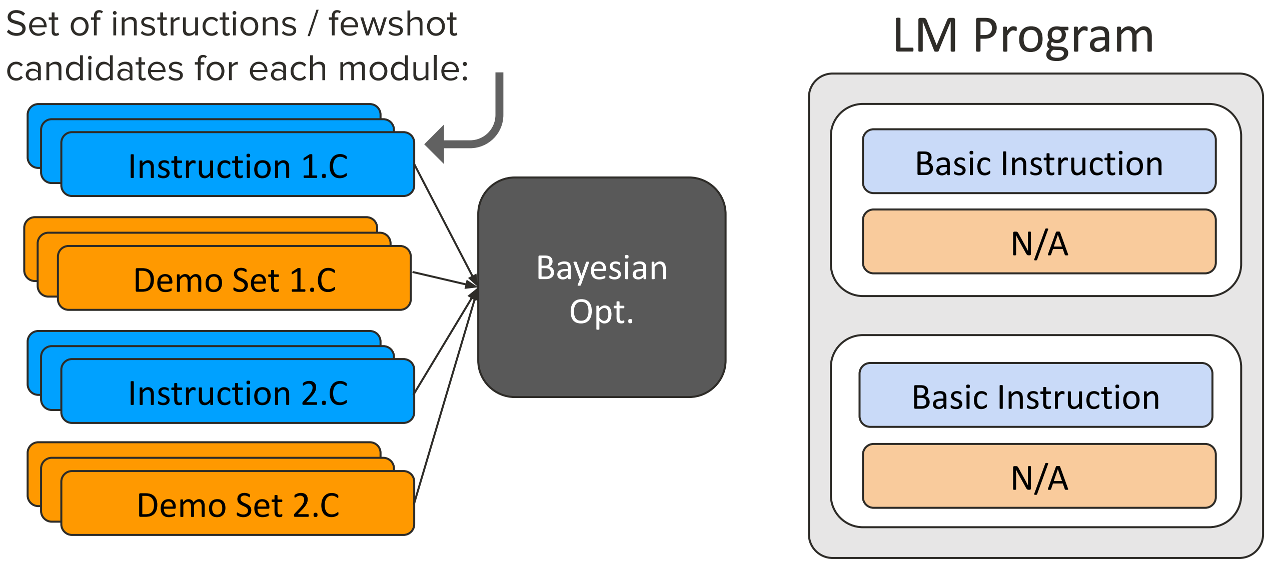
2. Propose Instruction Candidates using an LM Program

Credit Assignment

3. Jointly tune with a Bayesian hyperparameter optimizer

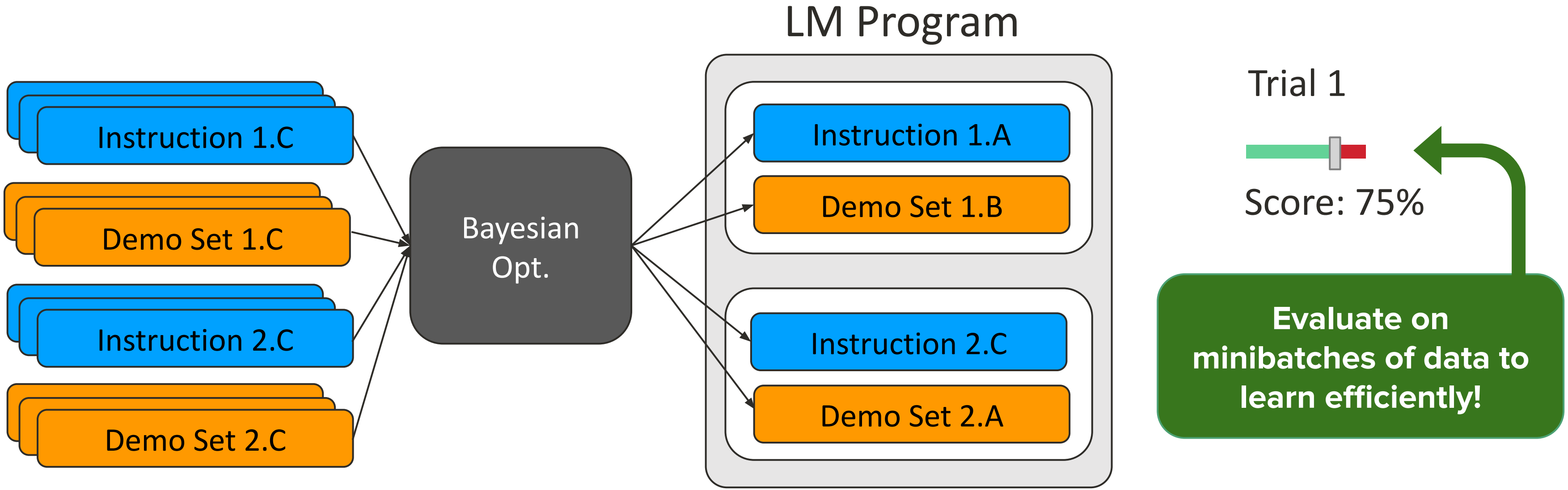
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



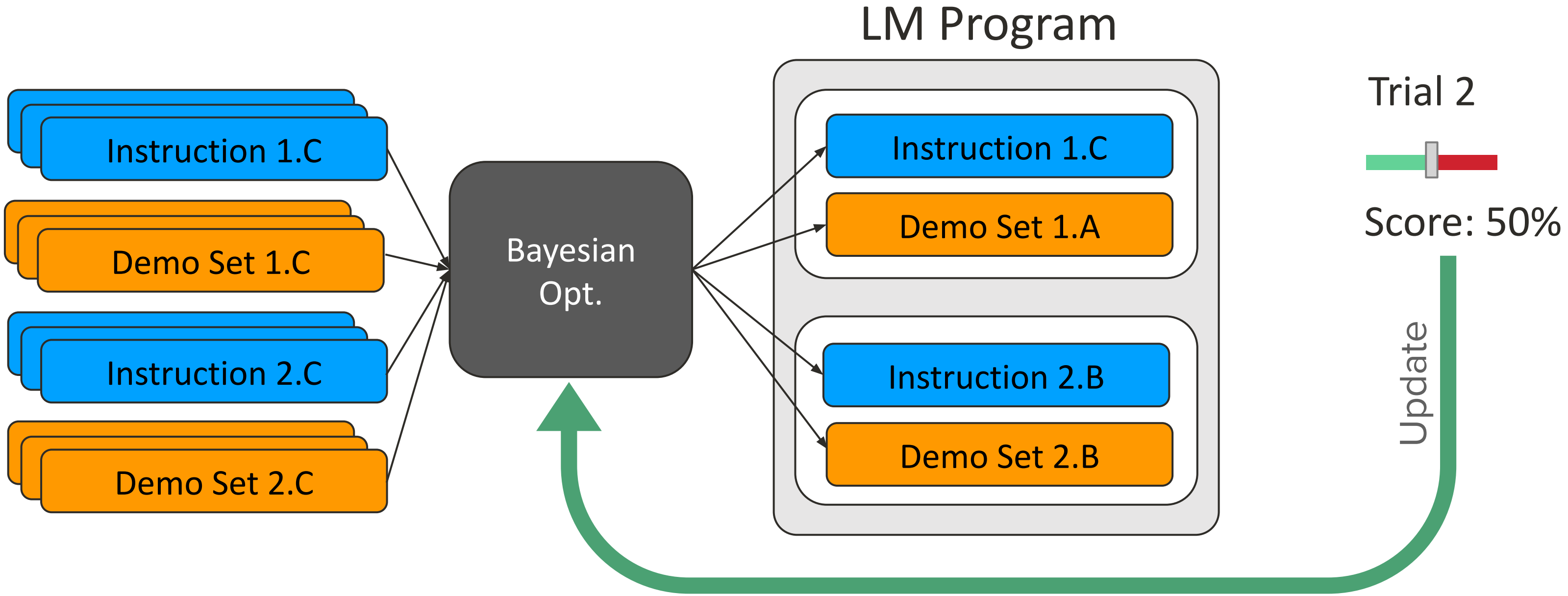
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



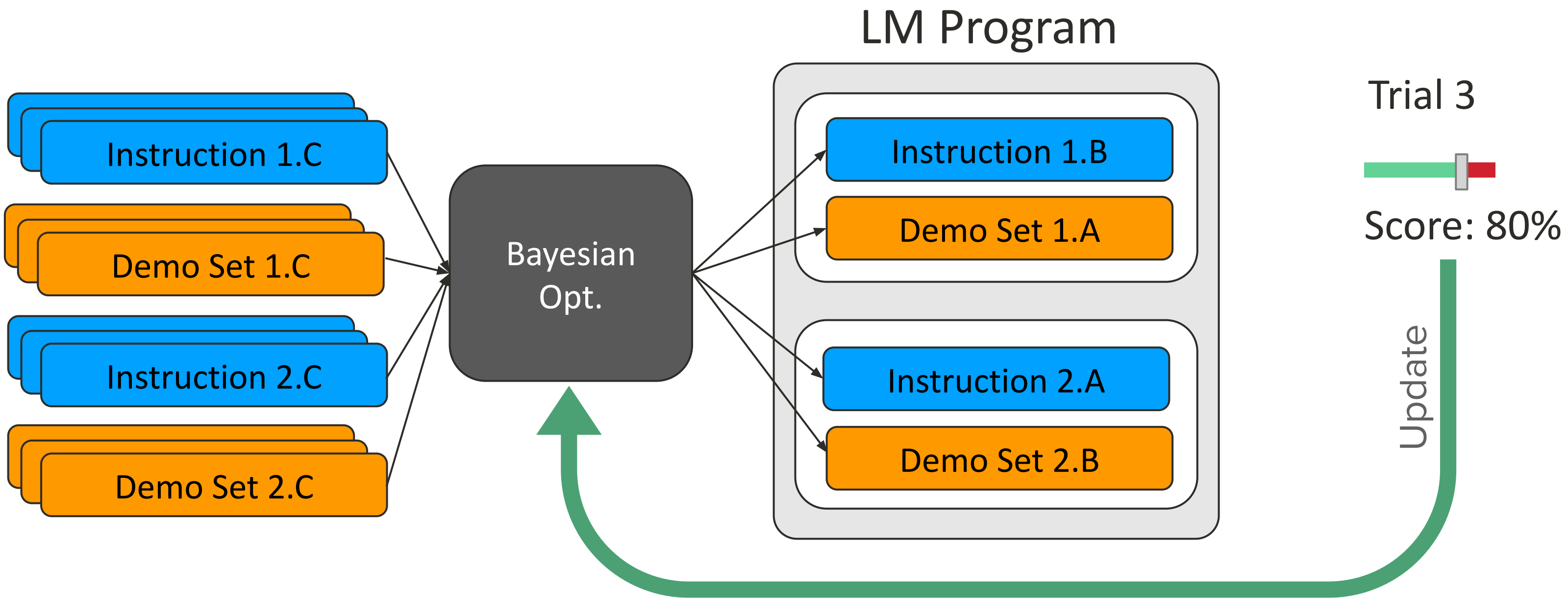
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



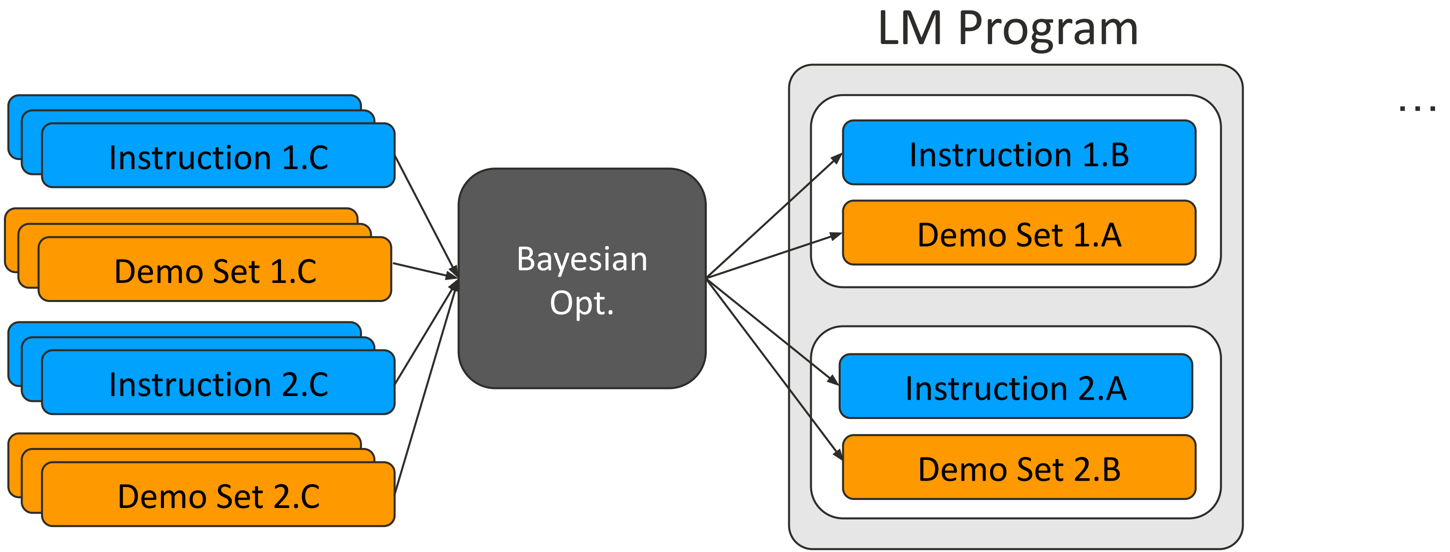
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



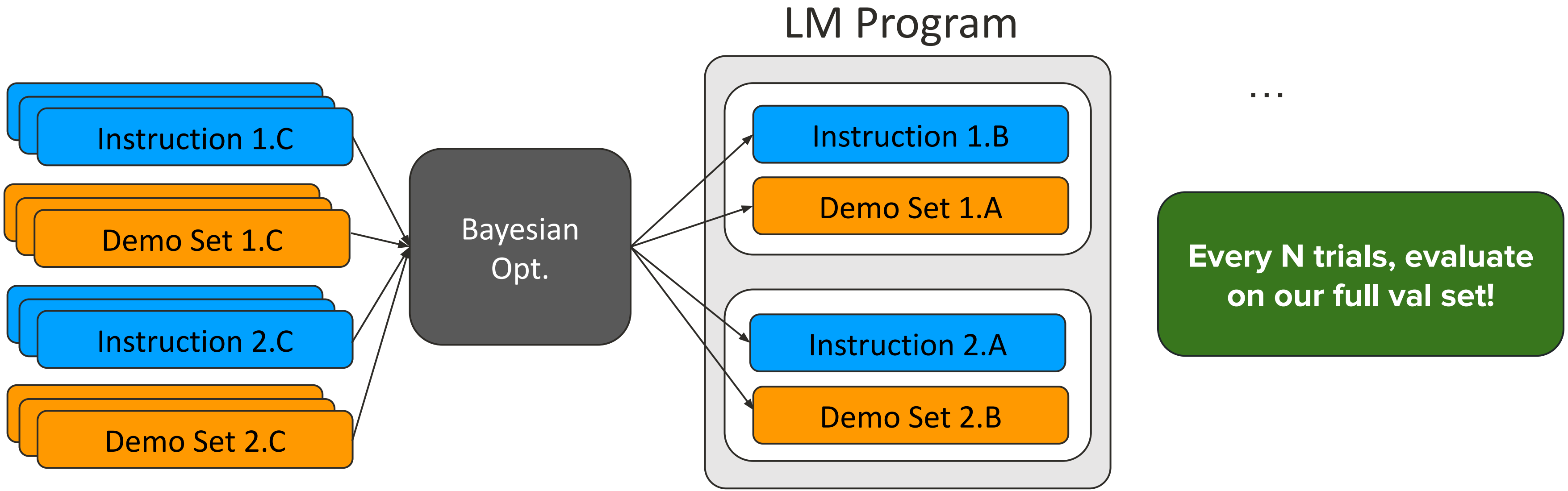
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



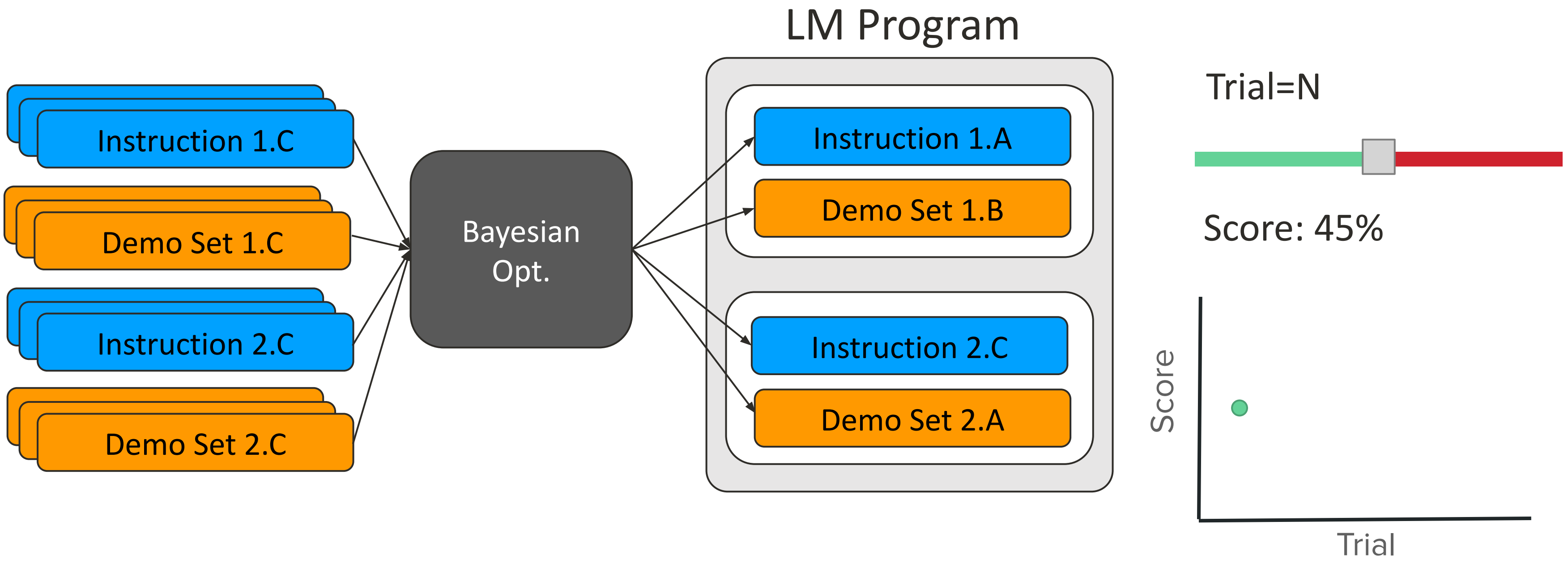
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



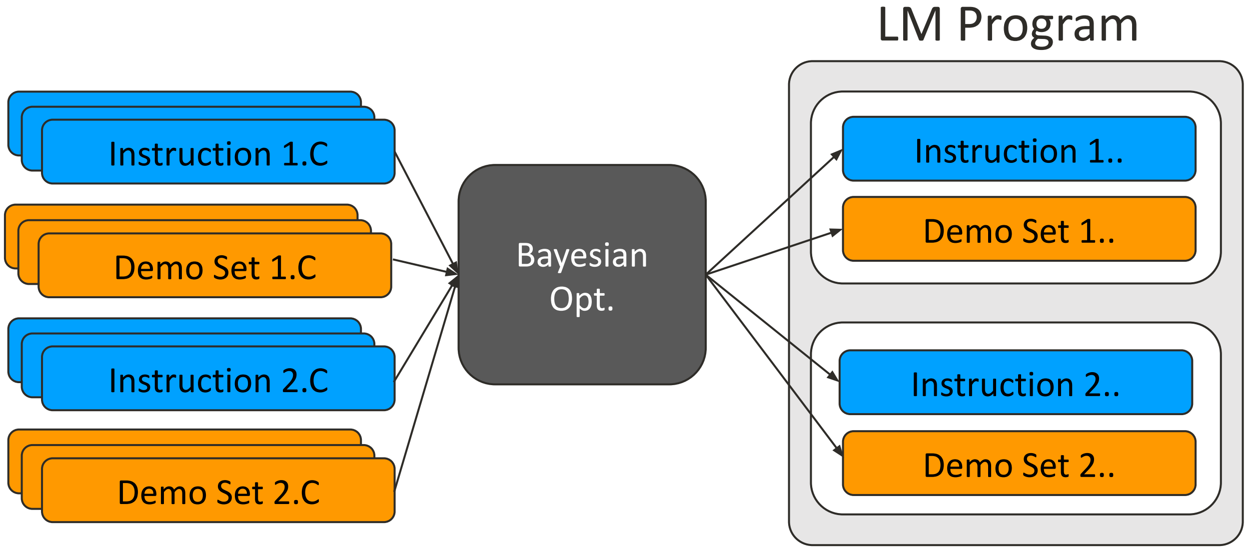
Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment

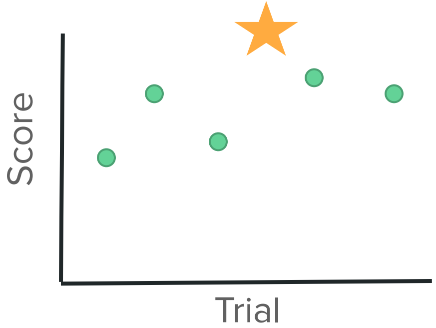


Step 3: Optimize with Bayesian Learning

Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Return LM Program with best score!



That works well in practice…

- **May’24: U of Toronto researchers won the MEDIQA competition via DSPy.**
- **Jun’24: U of Maryland researchers ran a direct case study.**

Rank	Team	Error Sentence Detection Accuracy
1	WangLab	83.6%
2	EM_Mixers	64.0%
3	knowlab_AIMed	61.9%
4	hyeonhwang	61.5%
5	Edinburgh Clinical NLP	61.1%
6	IryoNLP	61.0%
7	PromptMind	60.9%
8	MediFact	60.0%
9	IKIM	59.0%
10	HSE NLP	52.0%

Learn Prompting
@learnprompting

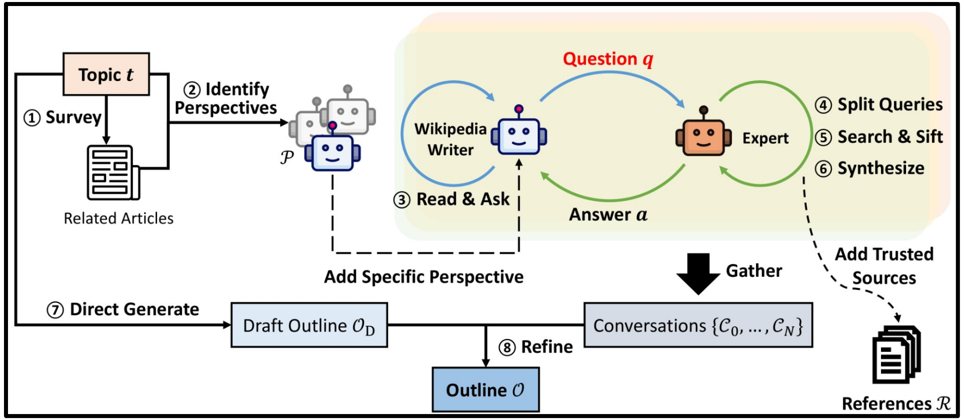
🤖We also put our expert prompt engineer against an AI prompt engineer.

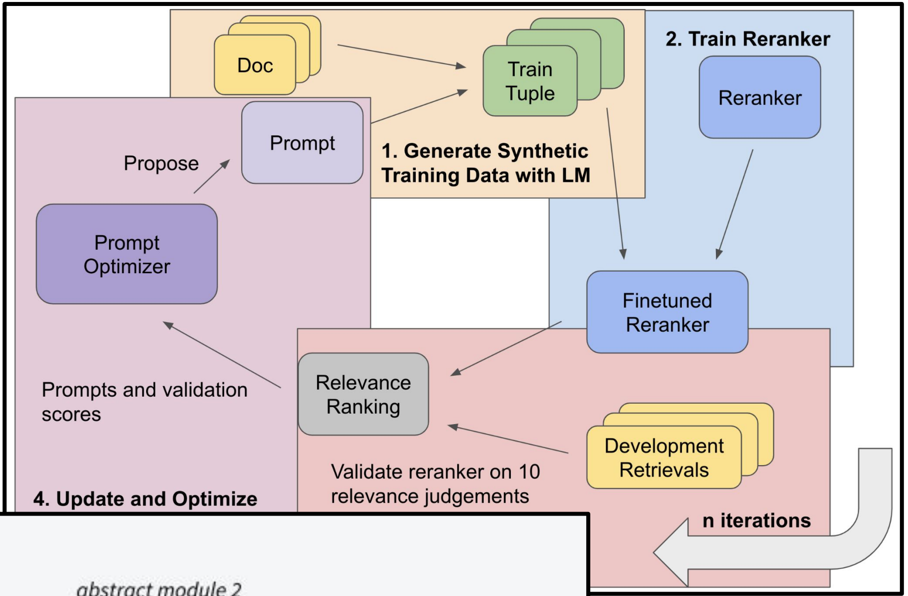
Expert human prompt engineer, @sanderschulhoff faced off against @lateinteraction's DSPy on a labeling task.

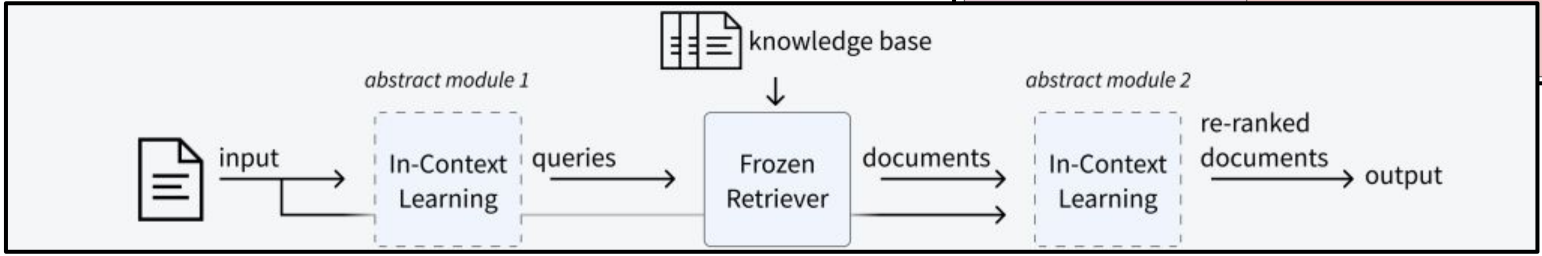
DSPY outperformed our expert Human Prompt Engineer by 50% on our test set and saved over 20 hours!

… and has enabled many SoTA systems

like PATH (Jasper Xia, UWaterloo); IReRa (Karel D'Oosterlink, UGhent), STORM (Yijia Shao, Stanford), EDEN & PAPILLON (Siyan Li, Columbia), Efficient Agents (Sayash Kapoor, Princeton), ECG-Chat (Yubao Zhao, Beijing Normal U), …







DSPy makes it possible to *program* LMs

~~Hand-written prompts~~ ⇒ Signatures

~~Prompting techniques and prompt chains~~ ⇒ Modules

```
qa = dspy.Predict("question -> answer")
mt = dspy.ChainOfThought("english_document -> french_translation")
rc = dspy.ProgramOfThought("contexts, question -> answer_found: bool")
```

~~Manual prompt engineering~~ ⇒ Optimizing program prompts/weights

```
Optimizer(metric).compile(program, dataset)
```