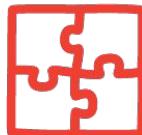




databricks



Compound AI Systems & DSPy



Omar Khattab

Research Scientist, Databricks

Incoming Assistant Professor, MIT EECS (Fall 2025)

Contributors 223



Includes slides adapted from
Krista Opsahl-Ong & Michael Ryan

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

When was Stanford University founded?

Stanford University was founded in 1891. It was named after Leland Stanford, the California governor who donated the land and funded the university. It is located in Stanford, California, near Palo Alto.

Parallelize this loop for me with 16 threads.

[code here]

To parallelize this loop with 16 threads, you can use the concurrent.futures module in Python, specifically the ThreadPoolExecutor class. Here is an example of how to do it:

css

 Copy code

```
from concurrent.futures import ThreadPoolExecutor  
import tqdm
```

Turning monolithic LMs into reliable AI systems remains challenging.

When was Stanford University founded?

Stanford University was founded in 1891. It was named after Leland Stanford, a California governor who donated the land and funds to establish the university. The university is located in Stanford, California, near Palo Alto.

Parallelize this loop for me with 16 threads.

[code here]

To parallelize this loop with 16 threads, you can use the concurrent.futures module in Python, specifically the ThreadPoolExecutor class. Here is an example of how to do it:

css

 Copy code

```
from concurrent.futures import ThreadPoolExecutor  
import tqdm
```



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

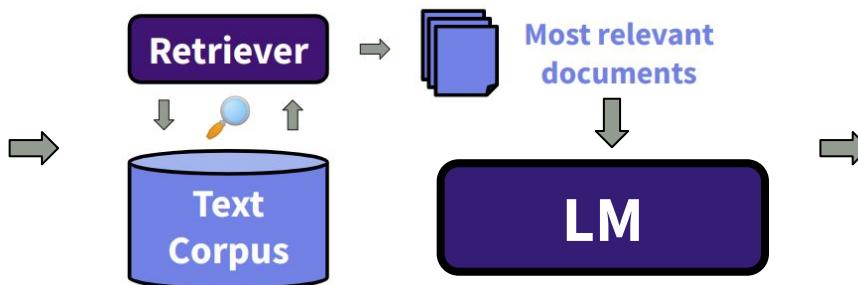


Efficiency: can use smaller LMs, offloading knowledge & control flow

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

Example: Retrieval-Augmented Generation

What compounds protect the digestive system?

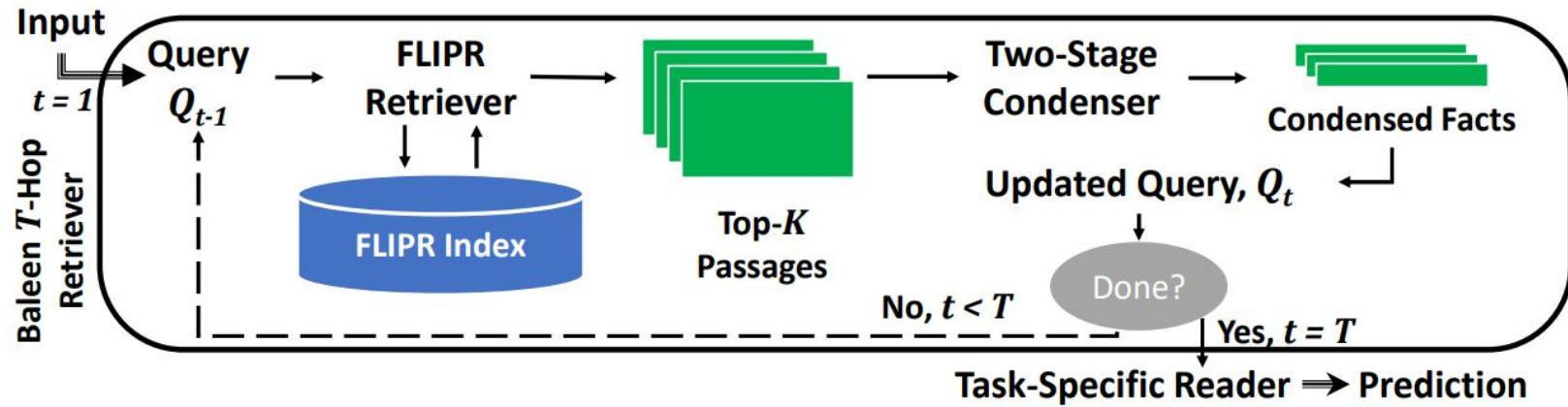


The stomach is protected by gastric acid and proteases.

- ⚡ **Transparency:** can debug traces & offer user-facing attribution
- ⚡ **Efficiency:** can use smaller LMs, offloading knowledge & control flow

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

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”

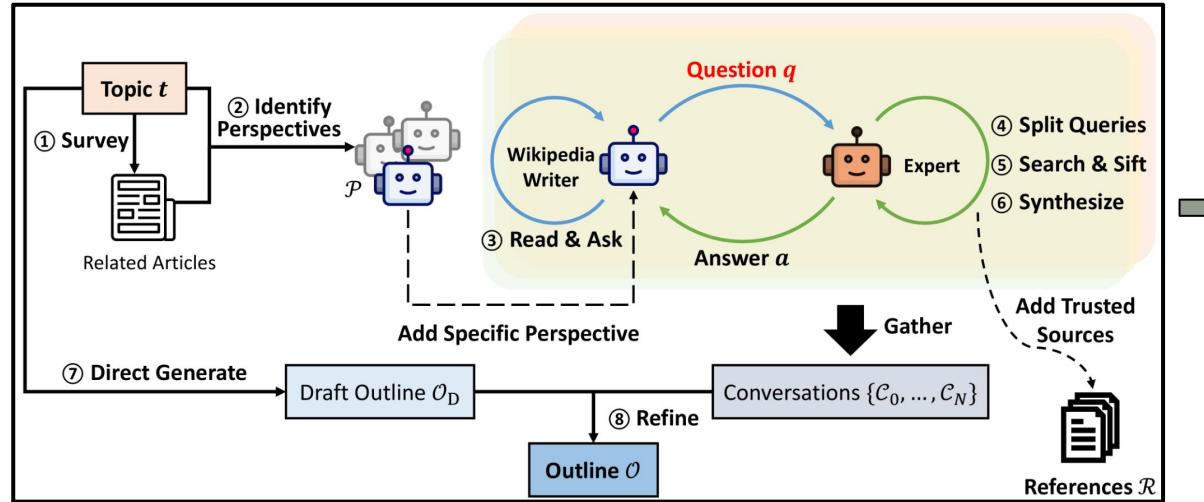


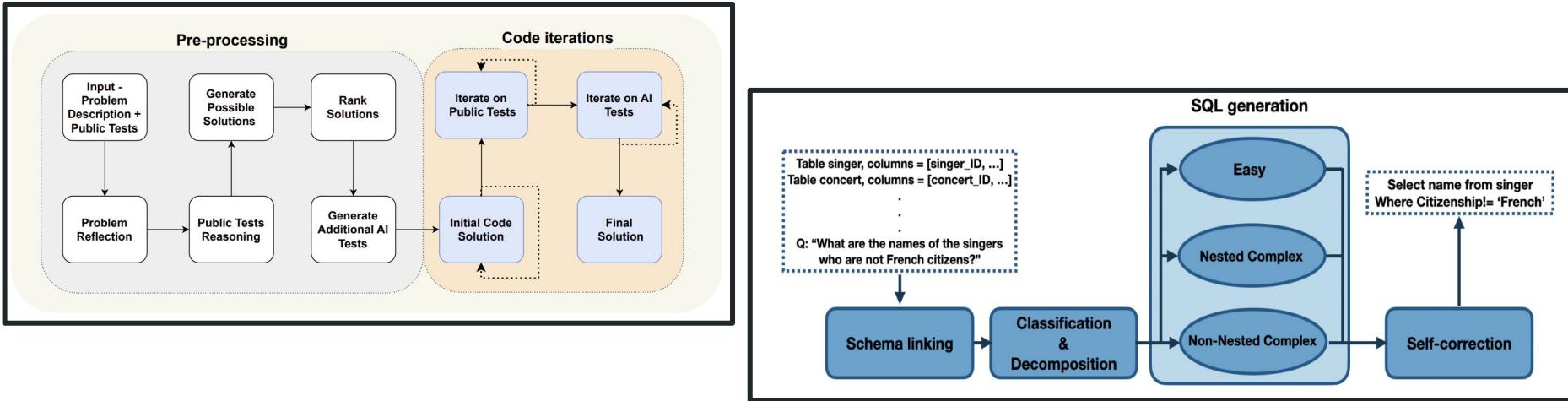
Table of contents

- Background and Motivation
 - Understanding Contextual Embeddings
 - Late Interaction in Information Retrieval
 - Visual and Cognitive Inspiration
 - Influence of BERT
 - Challenges and Innovations
 - Emergence of Alternative Approaches
- ColBERT Architecture
 - Overview
 - Late Interaction Mechanism
 - Model Components and Training
 - Advancements in ColBERTv2
- Training ColBERT
 - Initial Training
 - Retrieval and Ranking
 - Refinement with Naive Retriever
 - Iterative Training
 - Leveraging Cross-Encoders
 - Fine-Tuning and Distillation
- Advancements in Retrieval Efficiency and Accuracy
 - Efficiency in Late Interaction Retrieval



Quality: more reliable composition of better-sscoped LM capabilities

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

(Summary) Why Compound AI Systems?

1. **Quality:** more reliable composition of better-scoped LM capabilities
2. **Control:** can iteratively improve the system & ground it via tools
3. **Transparency:** can debug trajectories & offer user-facing attribution
4. **Efficiency:** can use smaller LMs, offloading knowledge & control flow
5. **Inference-time Scaling:** can systematically search for better outputs

Unfortunately, LMs are **highly sensitive** to how they're instructed to solve tasks, so under the hood we often...

```
1
2
3
4      # Q: I have a chair, two potatoes, a cauliflower, a lettuce head, two tables, a
       cabbage, two onions, and three fridges. How many vegetables do I have?
5
6
7      # note: I'm not counting the chair, tables, or fridges
8      vegetables_to_count = {
9          'potato': 2,
10         'cauliflower': 1,
11
12     }
13
14
15
16      pri
17
18
19      # Q
20
21      mus
22
23         drum : 1,
24         'flute': 1,
25         'clarinet': 1,
26         'violin': 1,
27         'accordion': 4
```

Code Blame 1 lines (1 loc) · 60.9 KB

```
1 {"react_put_0": "You are in the middle of a room. Looking quickly
```

Unfortunately, LMs are **highly sensitive** to how they're instructed to solve tasks, so under the hood we often...

Each “prompt” couples five very different roles:

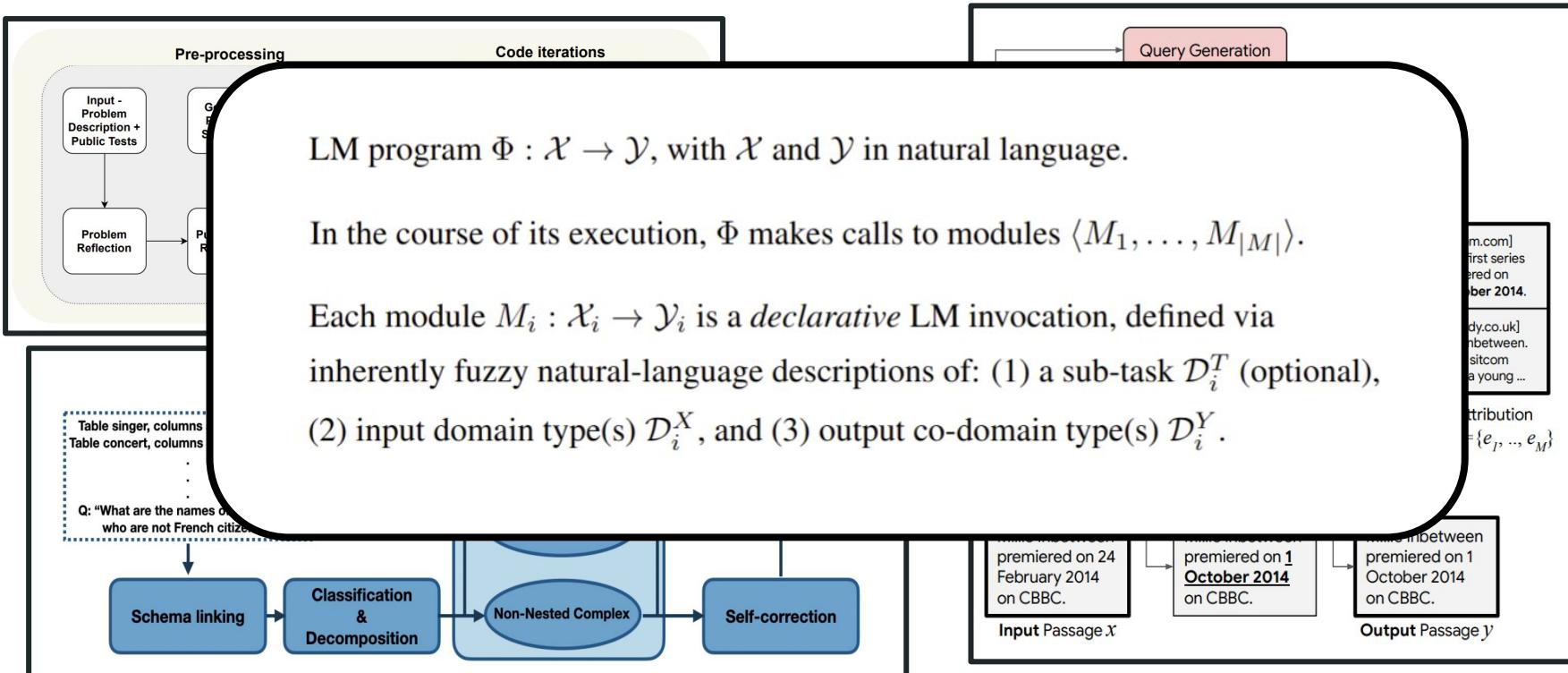
1. The core *input → output* behavior, a **Signature**.
2. The computation specializing an inference-time strategy to the signature, a **Predictor**.
3. The computation formatting the signature’s inputs and parsing its typed outputs, an **Adapter**.
4. The computations defining objectives and constraints on behavior, **Metrics** and **Assertions**.
5. The strings that instruct (or weights that adapt) the LM for 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... programming.

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



```
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:

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

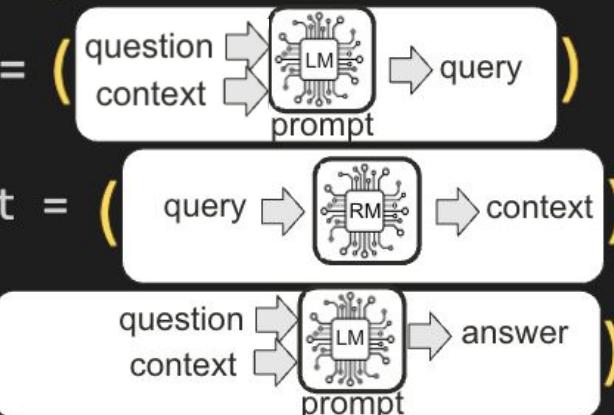
given a small training set $X = \{(x_1, m_1), \dots, (x_{|X|}, m_{|X|})\}$

and a metric $\mu : \mathcal{Y} \times \mathcal{M} \rightarrow \mathbb{R}$ 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*

```
def multihop_qa(question:str) -> str:  
    for i in range(2):  
        query = ( question  
                  context  
                  prompt )  
        context = ( query  
                    LM  
                    context )  
    return ( question  
            context  
            prompt )
```



This can be expressed as the following DS_Py program

```
class MultiHop(dspy.Module):
    def __init__(self):
        self.generate_query = dspy.ChainOfThought("context, question -> query")
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []
        for hop in range(2):
            query = self.generate_query(context, question).query
            context += dspy.Retrieve(k=3)(query).passages
        answer = self.generate_answer(context, question)

    return answer
```

Anatomy of an LM program in DSPy

```
class MultiHop(dspy.Module):
    def __init__(self):
        self.generate_query = dspy.ChainOfThought("context, question -> query")
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []
        for hop in range(2):
            query = self.generate_query(context, question).query
            context += dspy.Retrieve(k=3)(query).passages
        answer = self.generate_answer(context, question)

        return answer
```

Init method defines LM calls

Anatomy of an LM program in DSPy

```
class MultiHop(dspy.Module):
    def __init__(self):
        self.generate_query = dspy.ChainOfThought("context, question")
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []
        for hop in range(2):
            query = self.generate_query(context, question).query
            context += dspy.Retrieve(k=3)(query).passages
        answer = self.generate_answer(context, question)

        return answer
```

Forward method defines
program logic

Anatomy of an LM program in DSPy

```
class MultiHop(dspy.Module):  
    def __init__(self):  
        self.generate_query = dspy.ChainOfThought("context, question -> query")  
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")  
  
    def forward(self, question):  
        context = []  
        for hop in range(2):  
            query = self.generate_query(context, question).query  
            context += dspy.Retrieve(k=3)(query).passages  
        answer = self.generate_answer(context, question)  
  
        return answer
```

Signature: what to do,
not how to prompt!

Anatomy of an LM program in DSPy

Modules define the strategy
for expressing a signature

```
class MultiHop(ds...  
    def __init__(self):  
        self.generate_query = dspy.ChainOfThought("context, question -> query")  
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")  
  
    def forward(self, question):  
        context = []  
        for hop in range(2):  
            query = self.generate_query(context, question).query  
            context += dspy.Retrieve(k=3)(query).passages  
        answer = self.generate_answer(context, question)  
  
        return answer
```

Anatomy of an LM program in DSPy

```
class MultiHop(dspy.Module):
    def __init__(self):
        self.generate_query = dspy.ChainOfThought("context, question -> query")
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []
        for hop in range(2):
            query = self.generate_query(context, question).query
            context += dspy.Retrieve(k=3)(query).passages
        answer = self.generate_answer(context, question)

        return answer
```

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>

DSPy's Optimizers can then tune this prompt! ... jointly along with all other prompts 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 = MIPR0v2()  
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 a string prompt...

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 extends?

Thought 1: I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends, and then find the elevation range of the area.

Action 1: Search[Colorado orogeny]

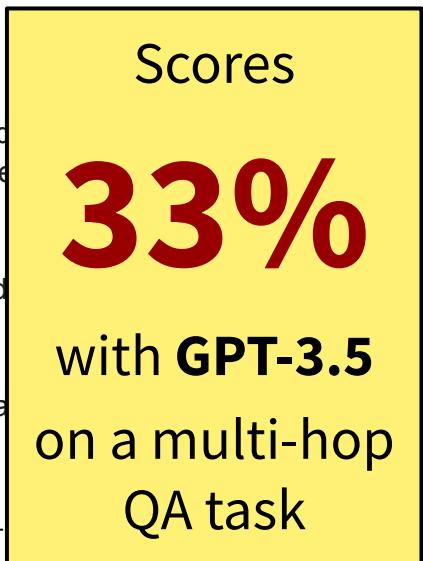
Observation 1: The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and New Mexico.

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 Plains.

[... truncated ...]



```

class MultiHop(dspy.Module):

    def __init__(self):
        self.generate_query = dspy.ChainOfThought("context, question -> query")
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []
        for hop in range(2):
            query = self.generate_query(context, question).query
            Carefully read the provided `context` and `question`. Your task is to formulate a concise and relevant `search_query` that could be used to retrieve information from a search engine to answer the question most effectively. The `search_query` should encapsulate...
            Context: [1] Twilight is a series of four vampire-themed fantasy romance...
            [2] The Harper Connelly Mysteries is a series of fantasy...
            Question: In which year was the first of the vampire-themed fantasy romance novels, for which The Twilight Saga serves as a spin-off encyclopedic reference book, first published?
            Reasoning: Let's determine when that fantasy romance novel was first published.
            Search Query: When was the first of the vampire-themed fantasy romance novels published?

            Context: [1] The Victorians - Their Story In Pictures is a 2009 British documentary ...
            [2] The Caxtons: A Family Picture is an 1849 Victorian novel by Edward ...
            Question: The Victorians is a documentary series written by an author born in what year?
            Reasoning: We know that the documentary series is about Victorian art and culture, and it was written by Jeremy Paxman. We need to find the year in which Jeremy Paxman was born.
            Search Query: Jeremy Paxman birth year

```

Scores

55%

with GPT-3.5
on multi-hop QA

39%

with T5-770M

50%

with Llama2-13B

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's **BootstrapFinetune**, starting from 200 answers, scores **39%**

DSPy Optimizers vary in how they tune the prompts & weights in a program, but at a high level they typically...

1. Construct an **initial prompt from each module** via an *Adapter*
2. **Generate examples of every module** via rejection sampling
3. Use the examples to **update the program's modules**
 - a. Automatic few-shot prompting: `dspy.BootstrapFewShotWithRandomSearch`
 - b. Induction of instructions: `dspy.MIPROv2`
 - c. Multi-stage fine-tuning: `dspy.BootstrapFinetune`

**Optimizing Instructions and Demonstrations
for Multi-Stage Language Model Programs**

Krista Opsahl-Ong^{1*}, Michael J Ryan^{1*}, Josh Purtell²,
David Broman³, Christopher Potts¹, Matei Zaharia⁴, Omar Khattab¹

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

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

Dilara Soylu Christopher Potts Omar Khattab

Stanford University

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 ...

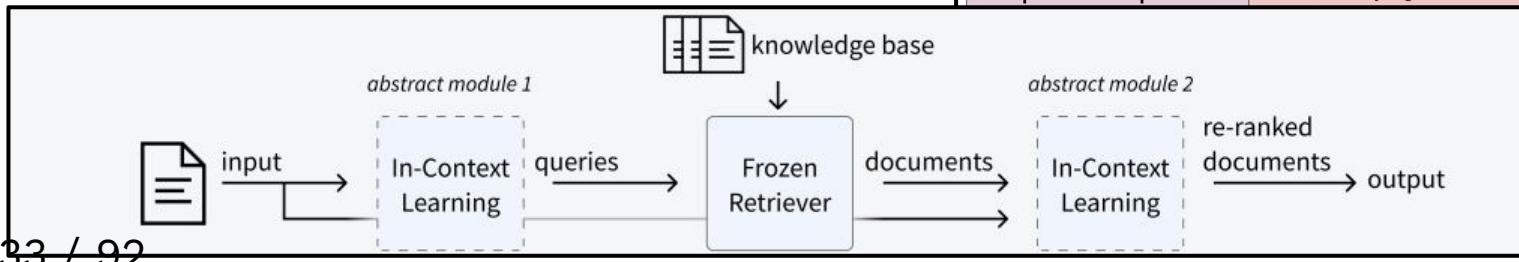
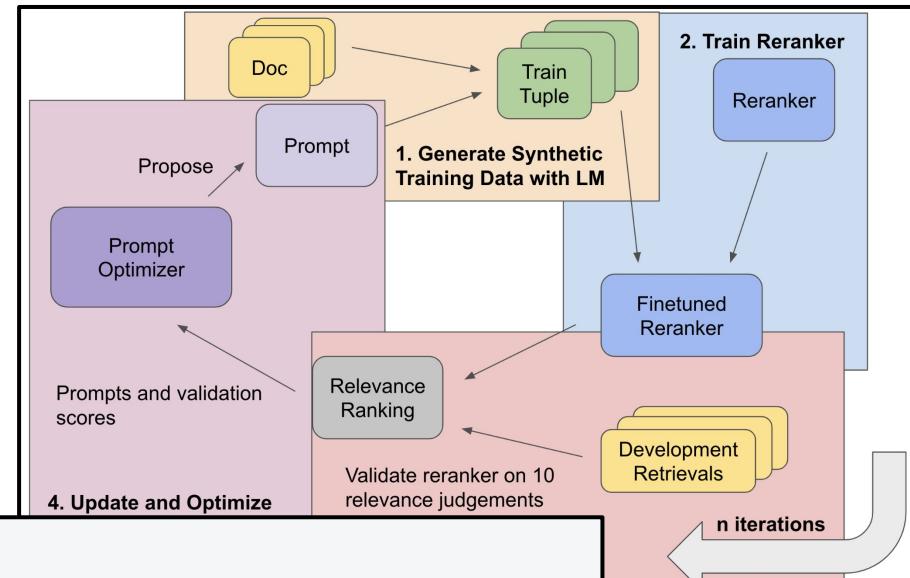
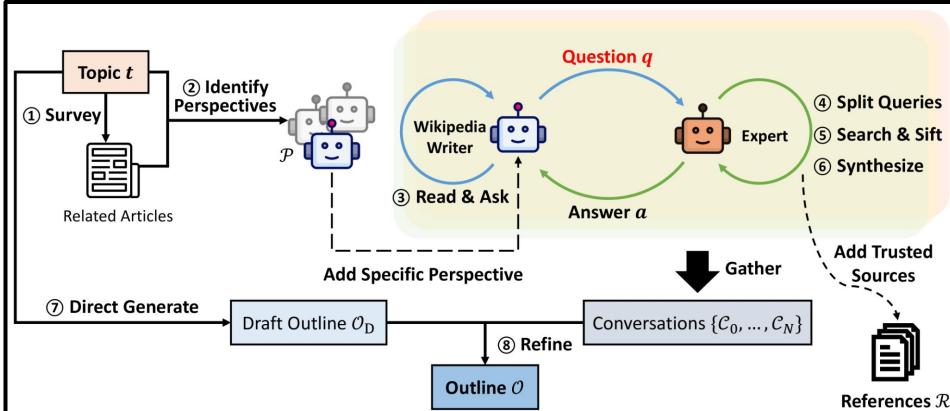
We also put our expert prompt engineer against an AI prompt engineer.

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 (Siyan Li, Columbia), Efficient Agents (Sayash Kapoor, Princeton), ECG-Chat (Yubao Zhao, Beijing Normal U), ...



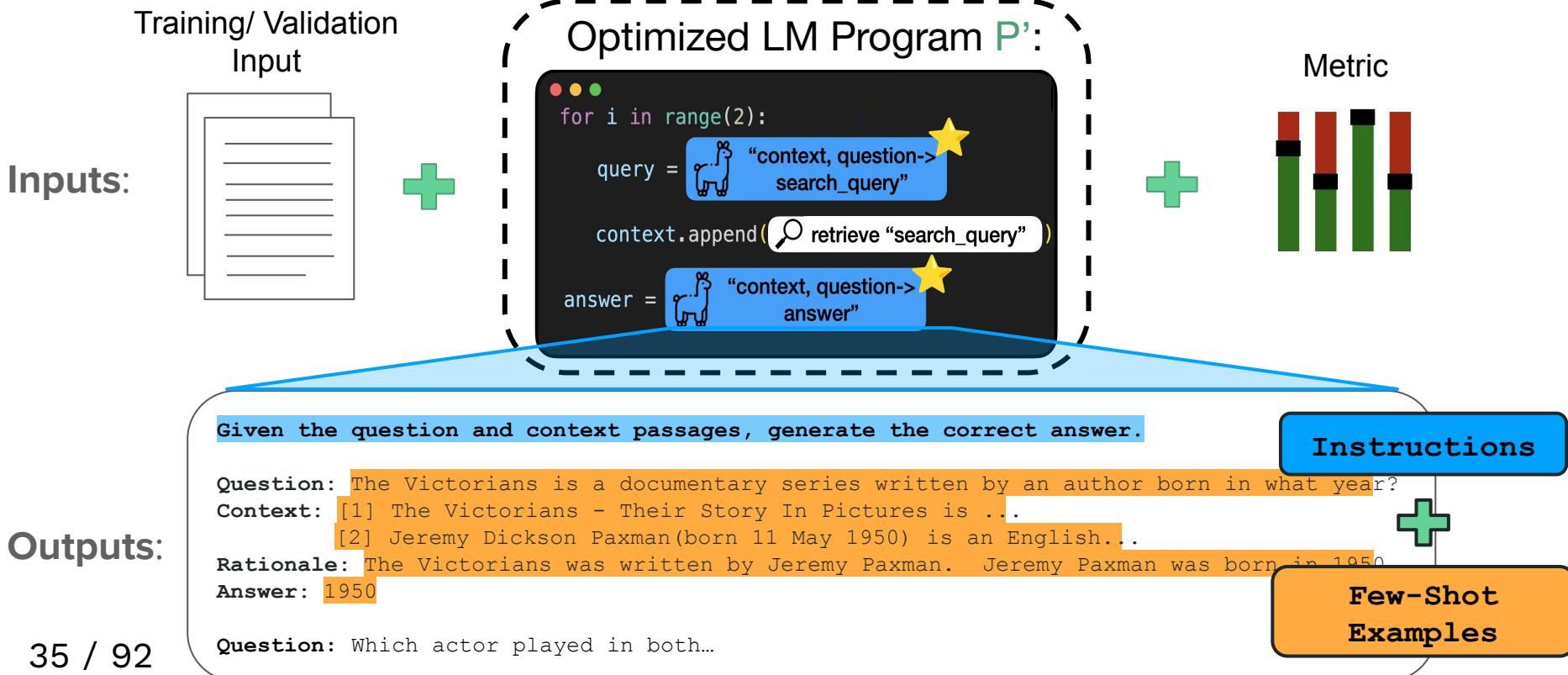
Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs

**Krista Opsahl-Ong^{1*}, Michael J Ryan^{1*}, Josh Purtell²,
David Broman³, Christopher Potts¹, Matei Zaharia⁴, Omar Khattab¹**

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

Slides adapted from
Krista Opsahl-Ong & Michael Ryan

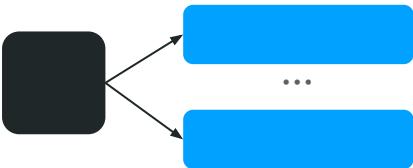
Problem Setting



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.

Key Challenges



Prompt Proposal.

Searching over all possible strings is intractable, especially as we add in multiple modules we need to optimize. Instead, we need to propose a *small set of high quality* options.

Credit Assignment.

We need efficient ways of inferring how each prompt variable contributes to performance, so that we can find the best set for our program.

Methods

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

Methods

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

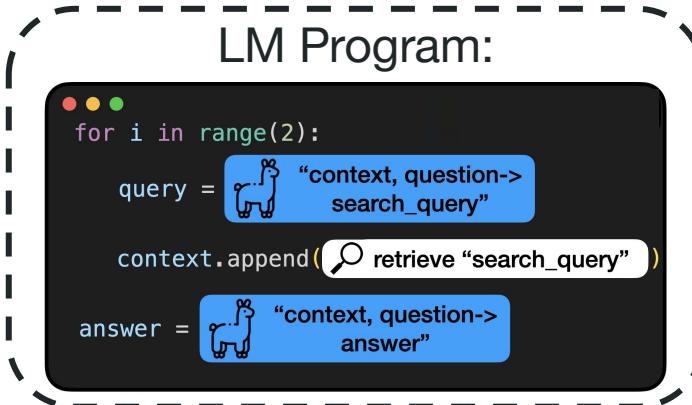


Bootstrap Few-shot examples with simple rejection sampling

Bootstrap Few-Shot Examples

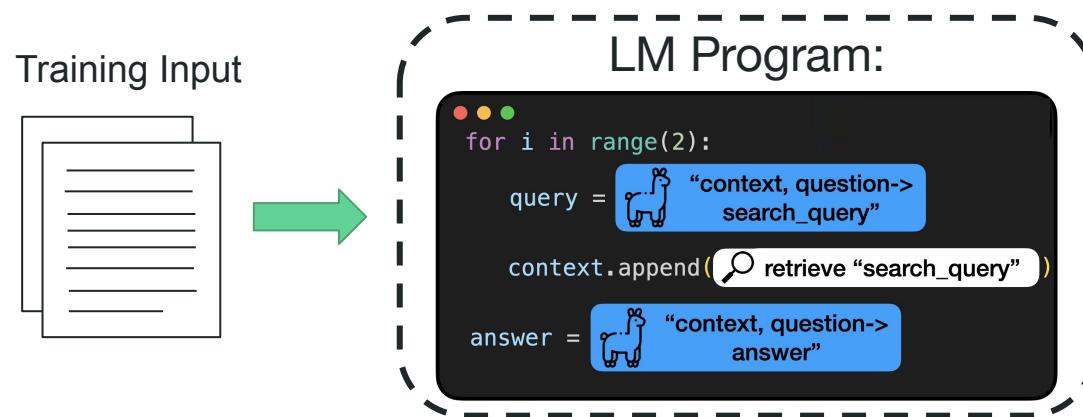
Bootstrap Few-Shot Examples

LM Program:

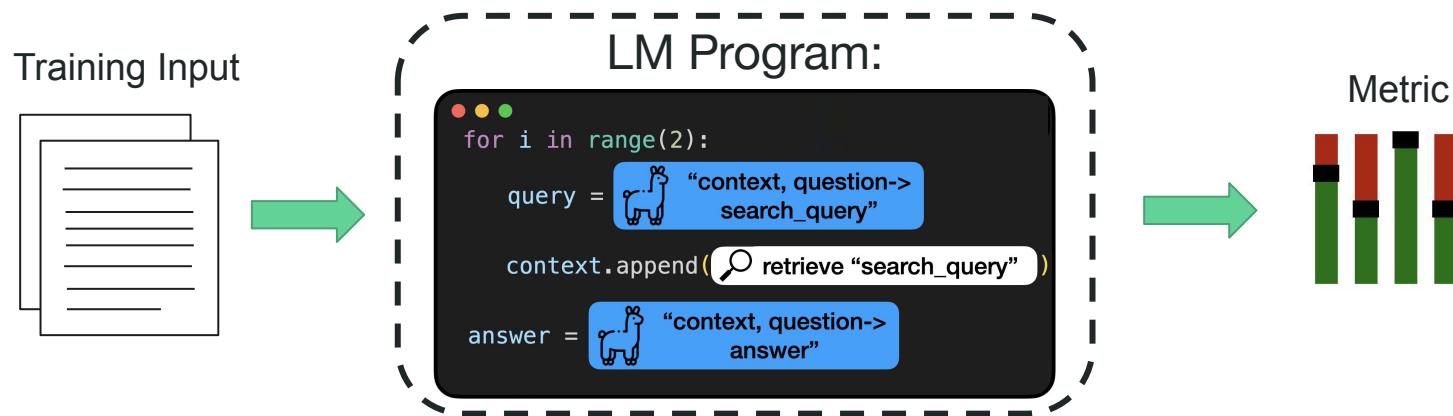


```
for i in range(2):
    query = "context, question->
              search_query"
    context.append(retrieve "search_query")
    answer = "context, question->
              answer"
```

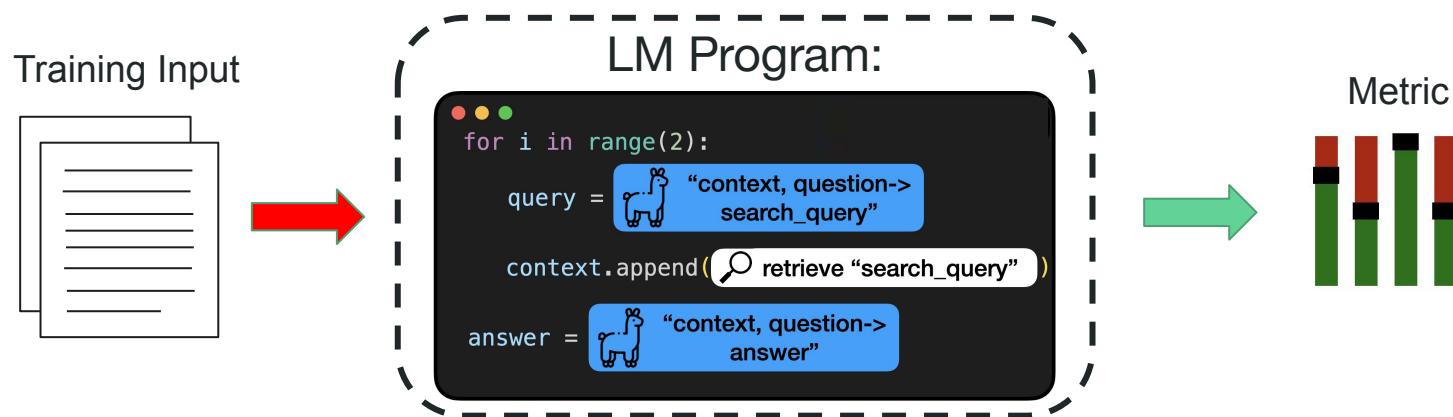
Bootstrap Few-Shot Examples



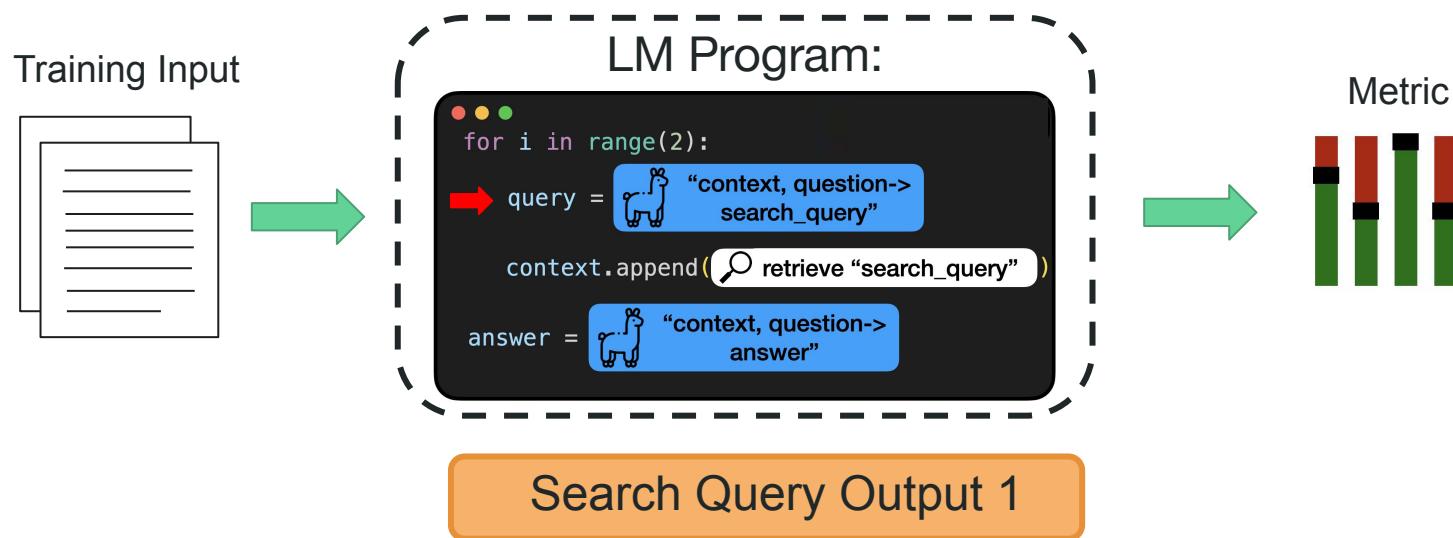
Bootstrap Few-Shot Examples



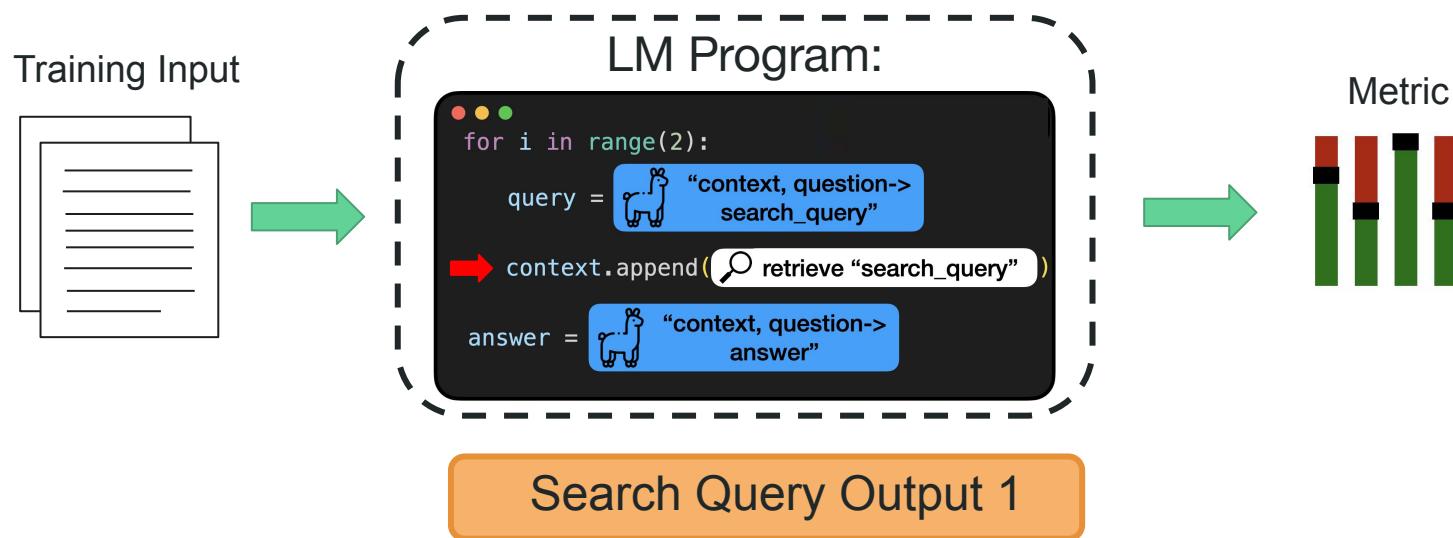
Bootstrap Few-Shot Examples



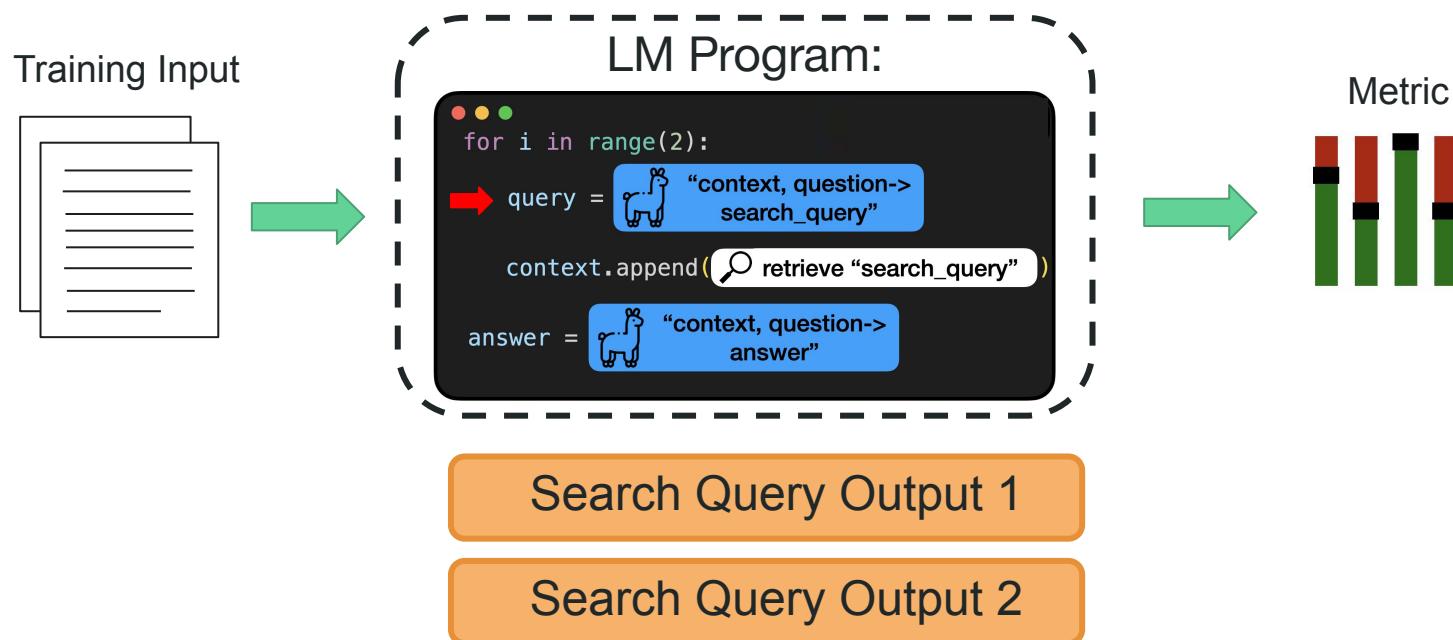
Bootstrap Few-Shot Examples



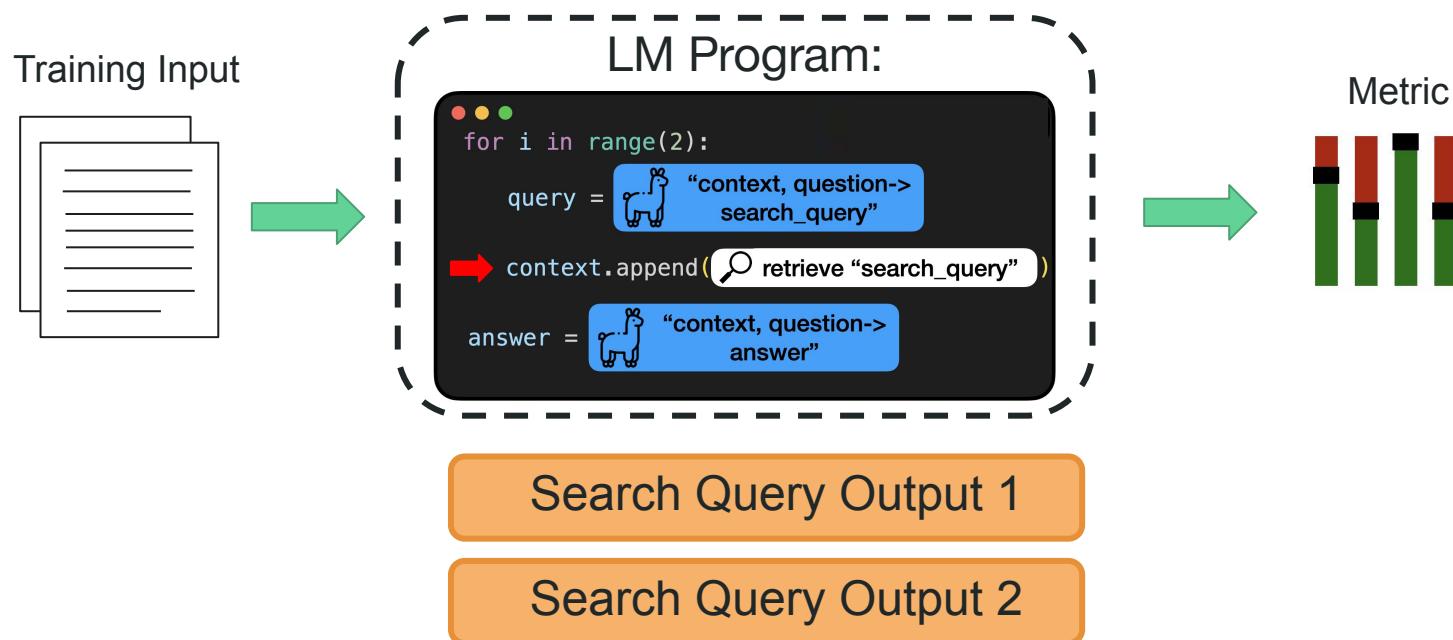
Bootstrap Few-Shot Examples



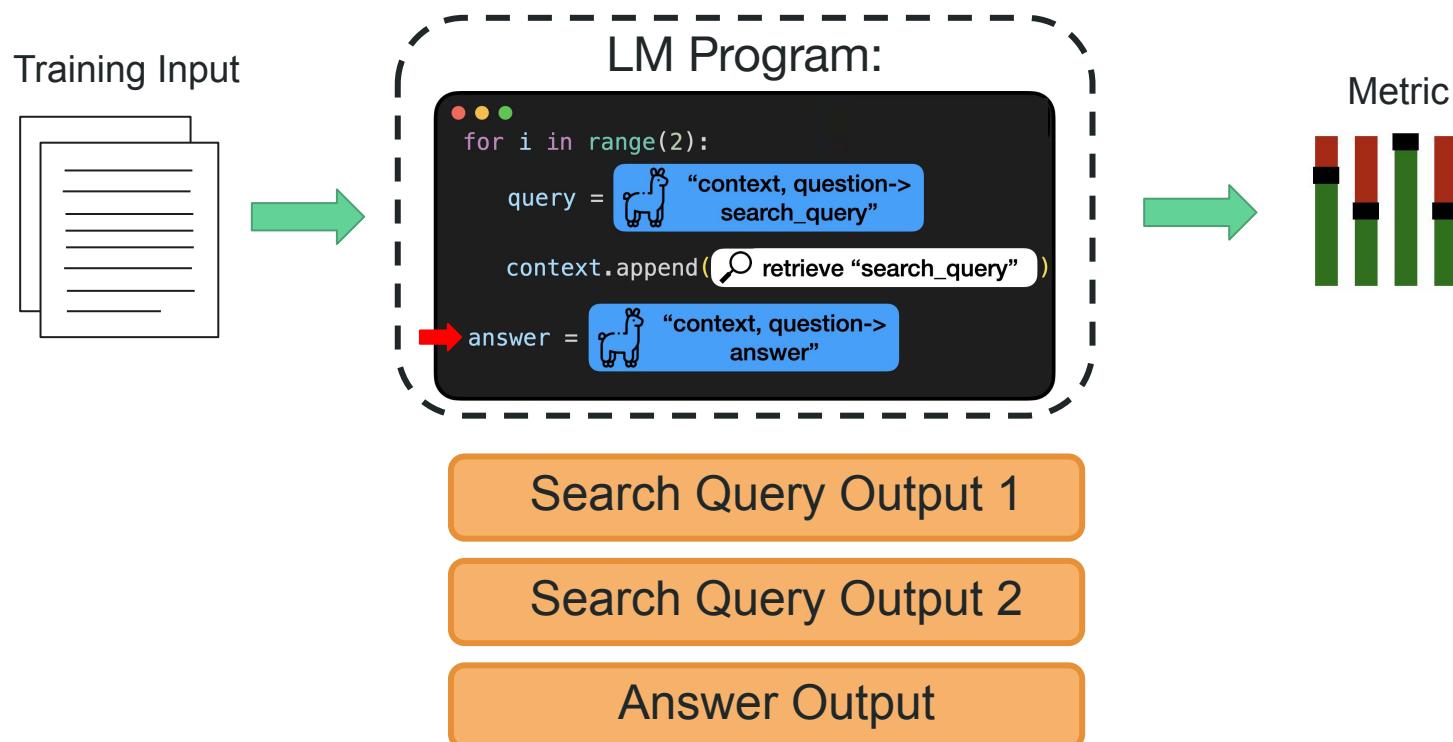
Bootstrap Few-Shot Examples



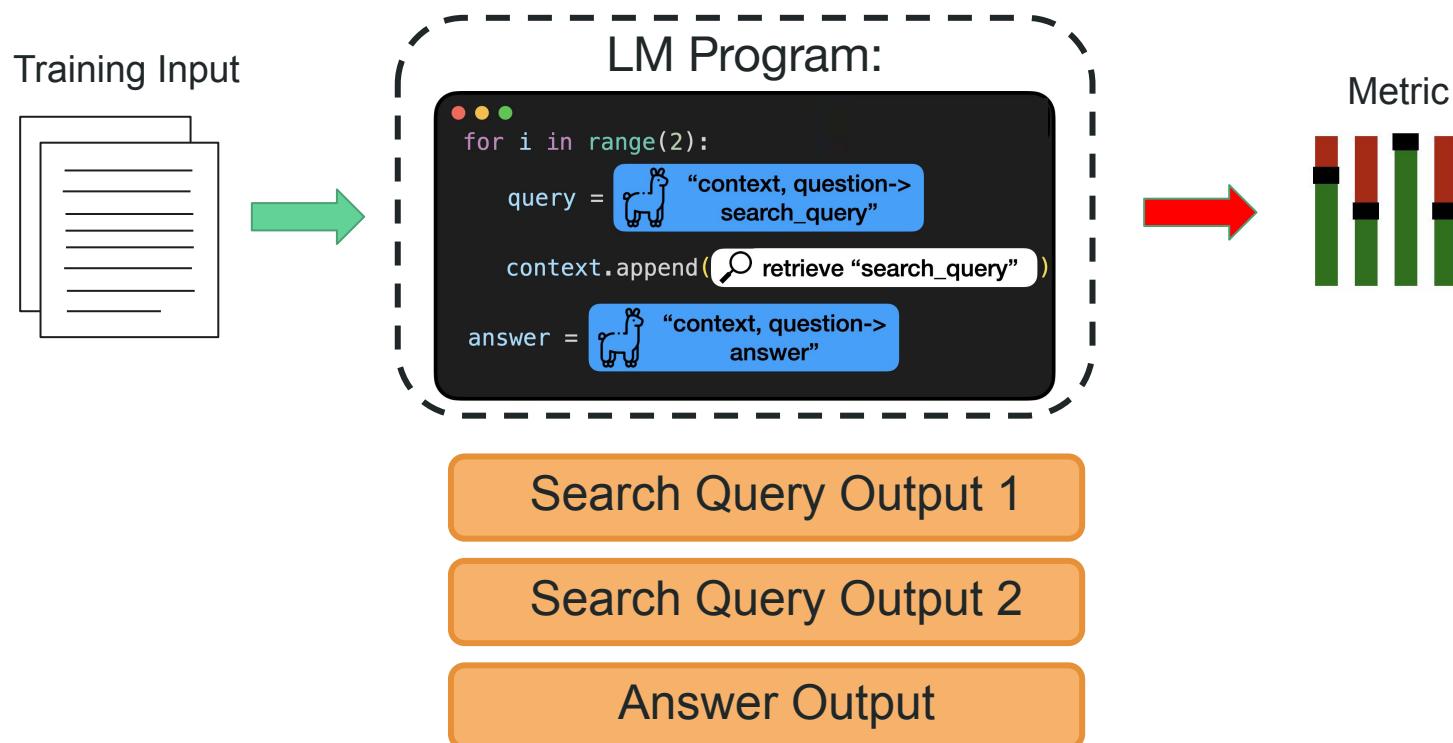
Bootstrap Few-Shot Examples



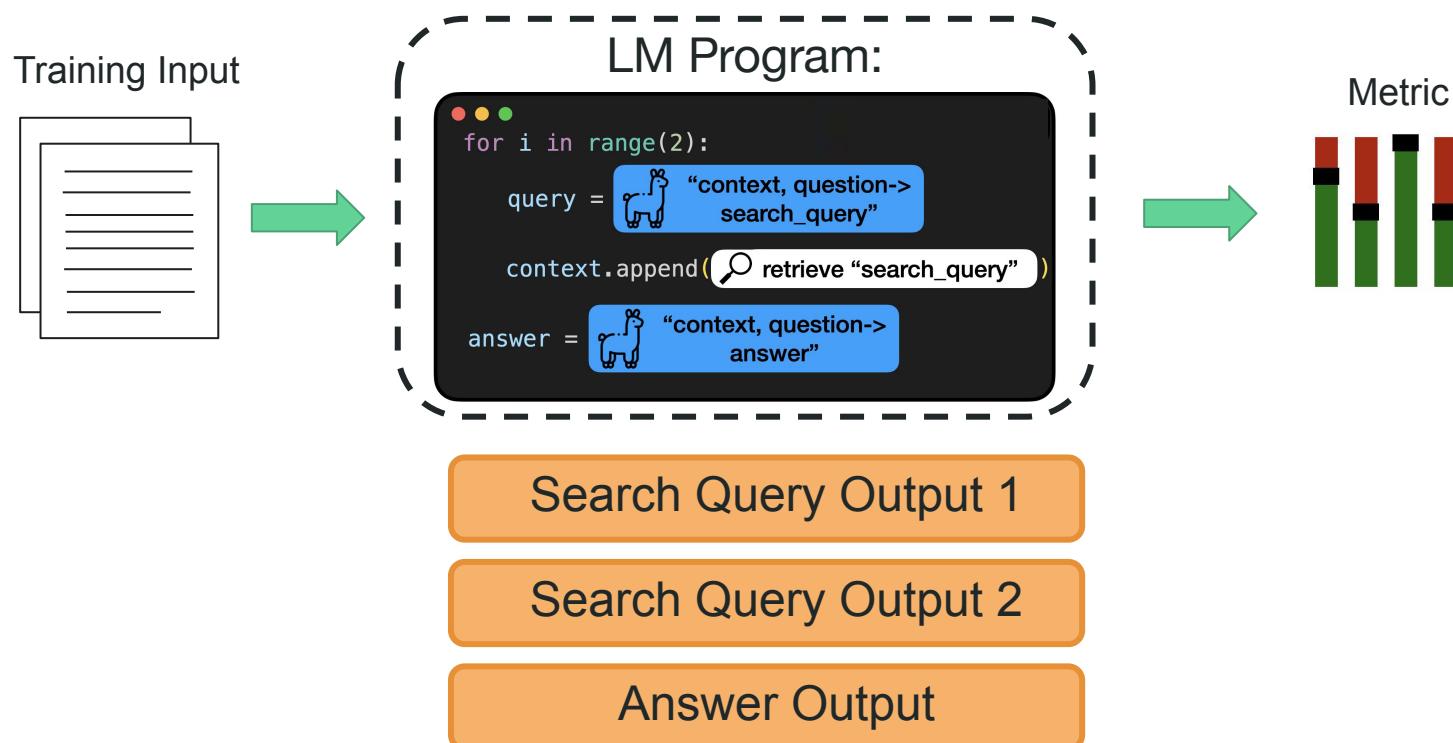
Bootstrap Few-Shot Examples



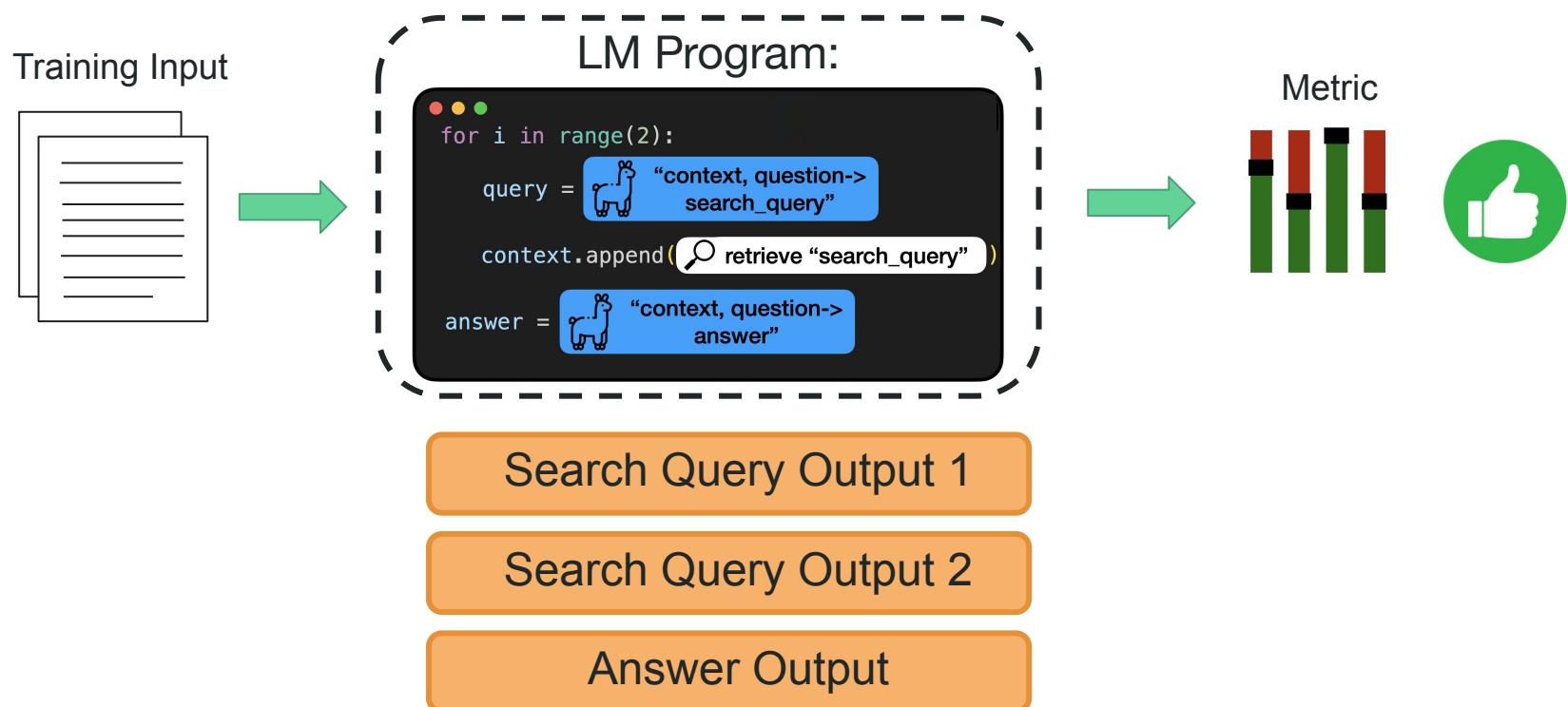
Bootstrap Few-Shot Examples



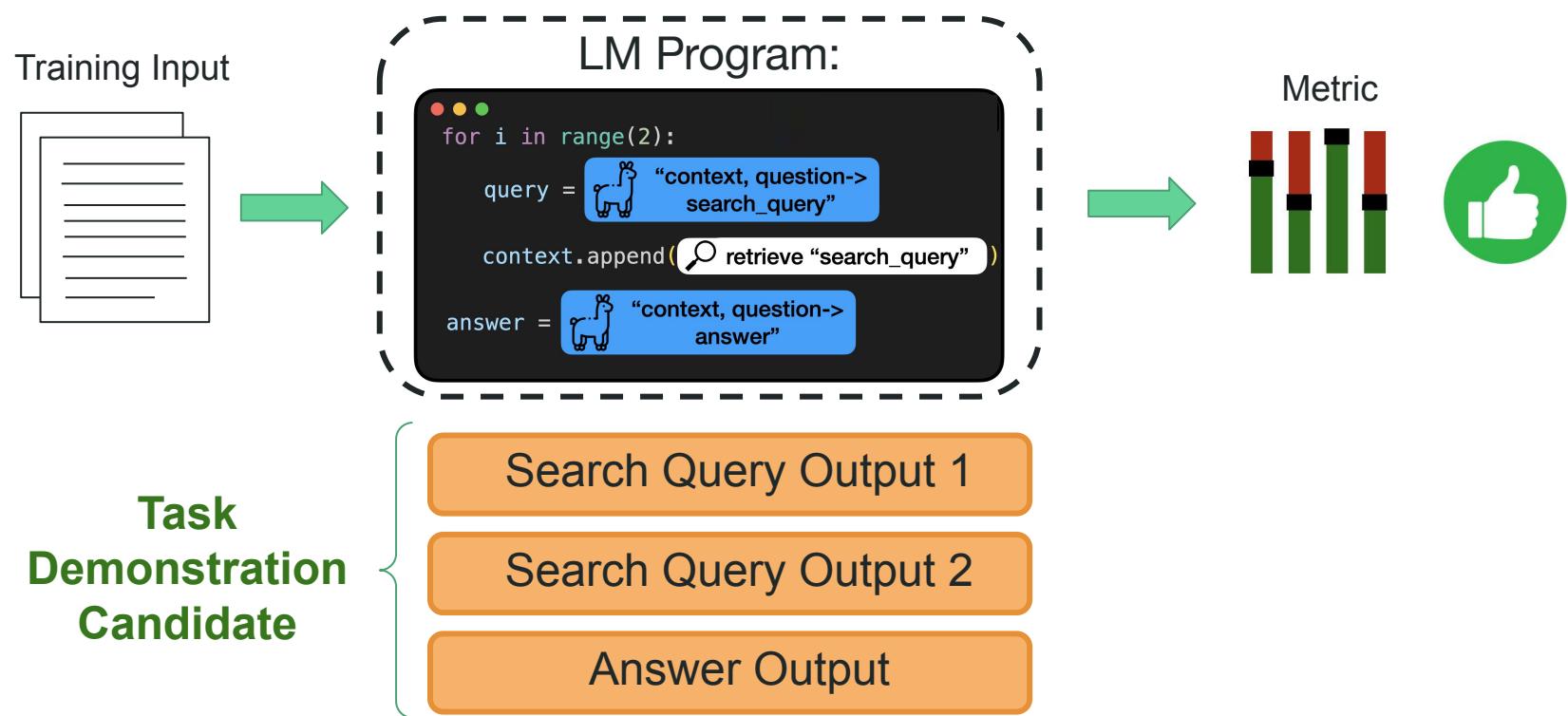
Bootstrap Few-Shot Examples



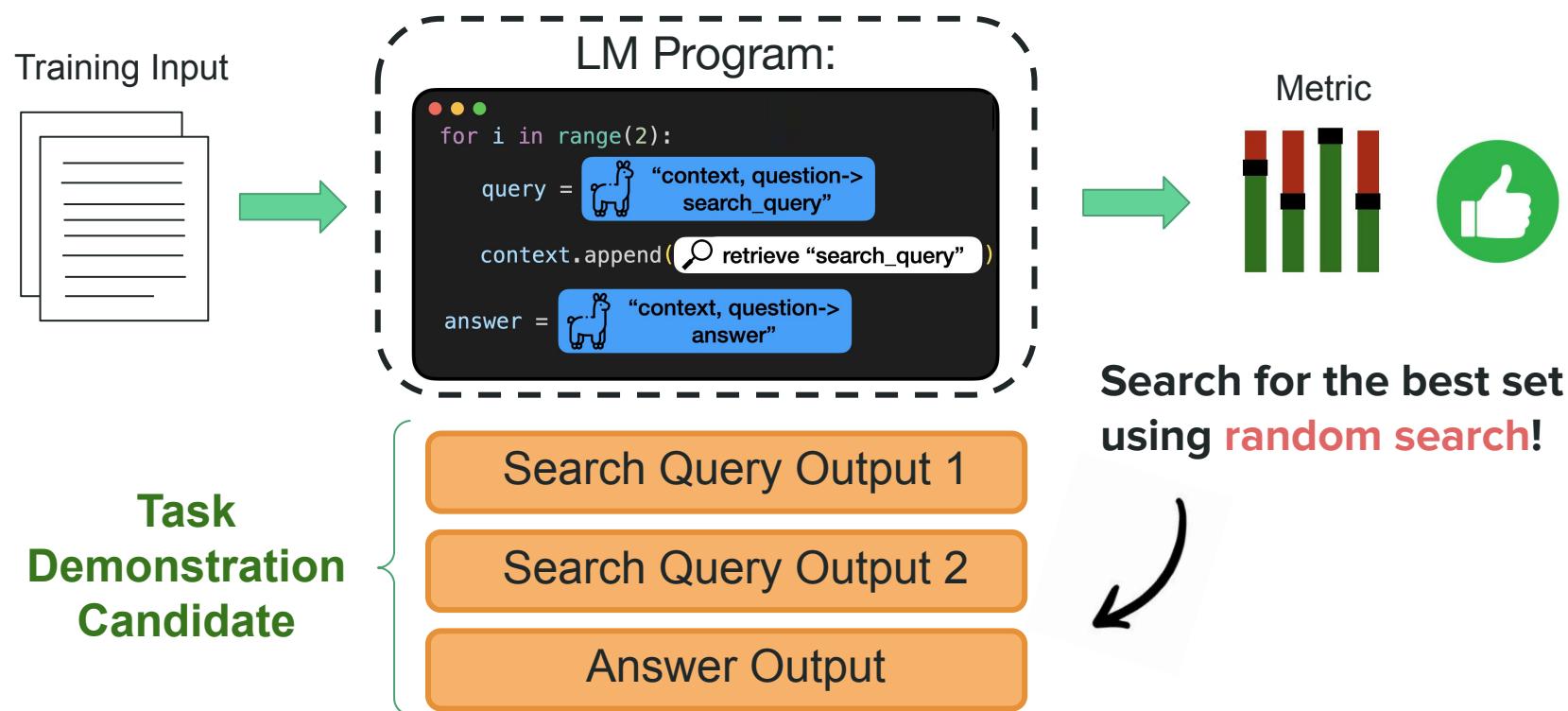
Bootstrap Few-Shot Examples



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

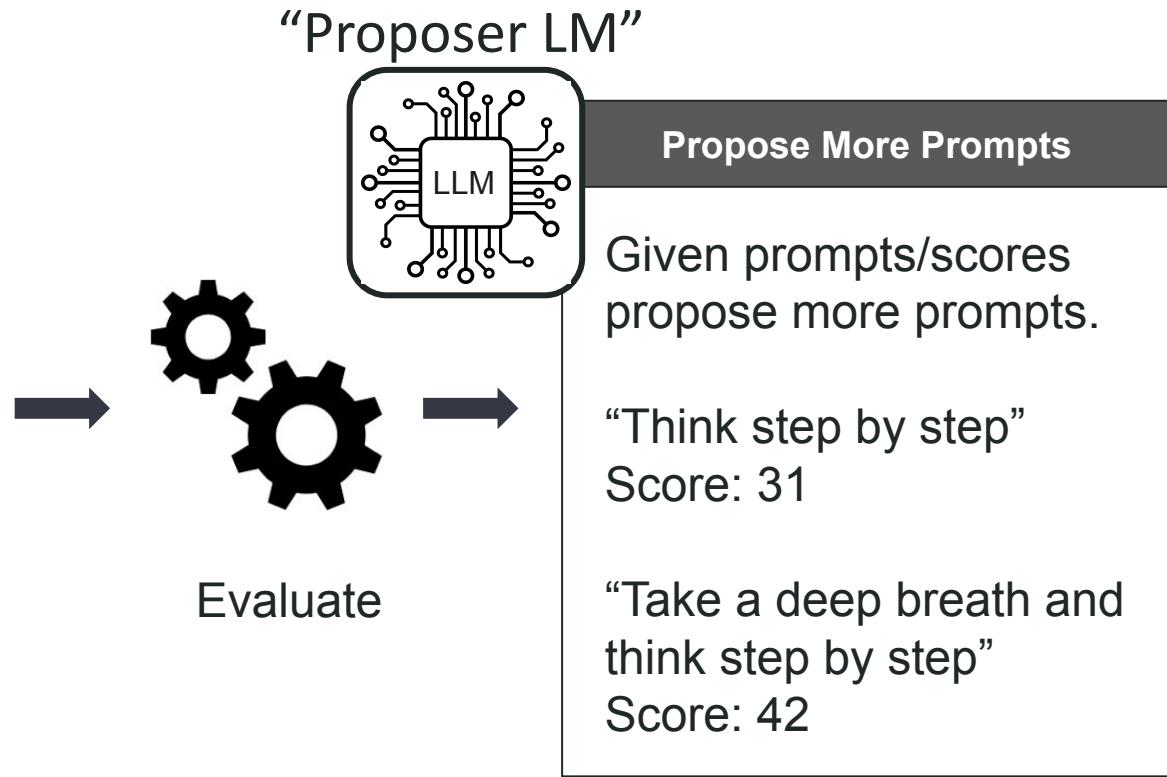
1. Bootstrap Few-shot
2. Extending OPRO
3. MIPRO



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

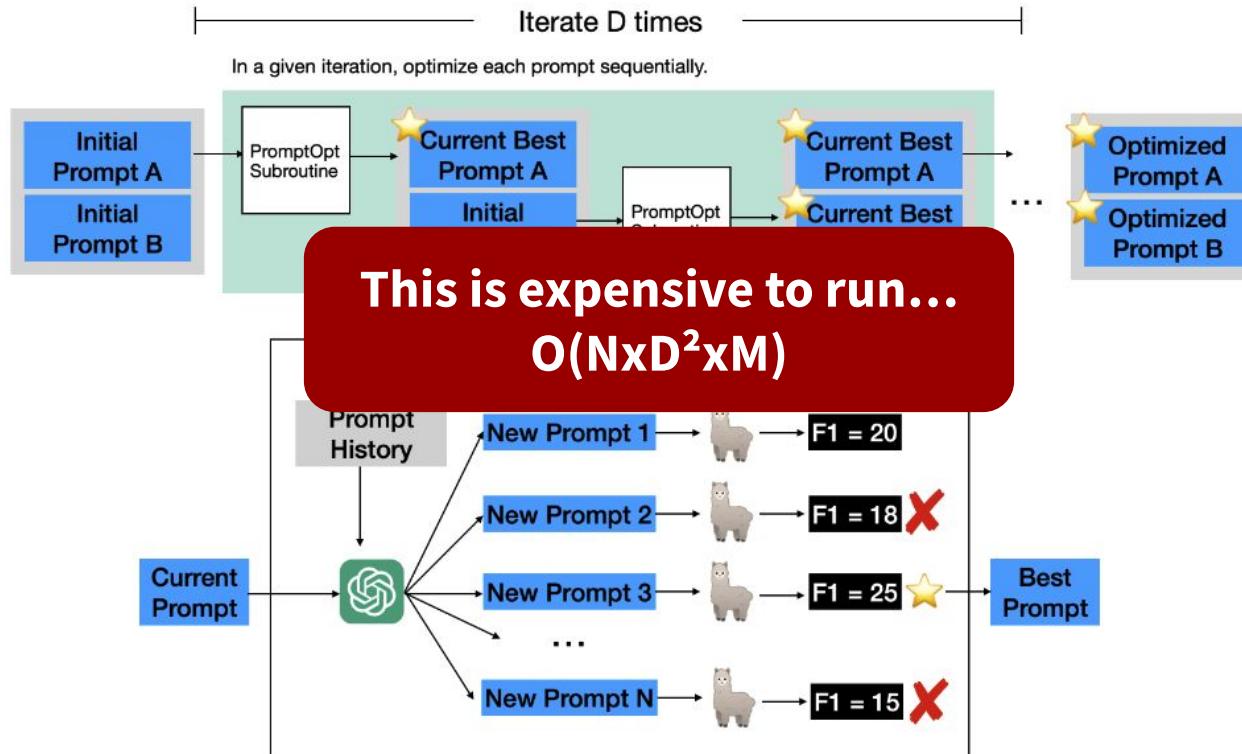
What is OPRO? Optimization through Prompting

| Prompt Proposals |
|---|
| “Think step by step” |
| “Take a deep breath and think step by step” |
| “Carefully solve the problem” |
| “Let’s do the math” |



Initial extension to multi-stage: CA-OPRO

Coordinate-Ascent OPRO



Module-Level OPRO

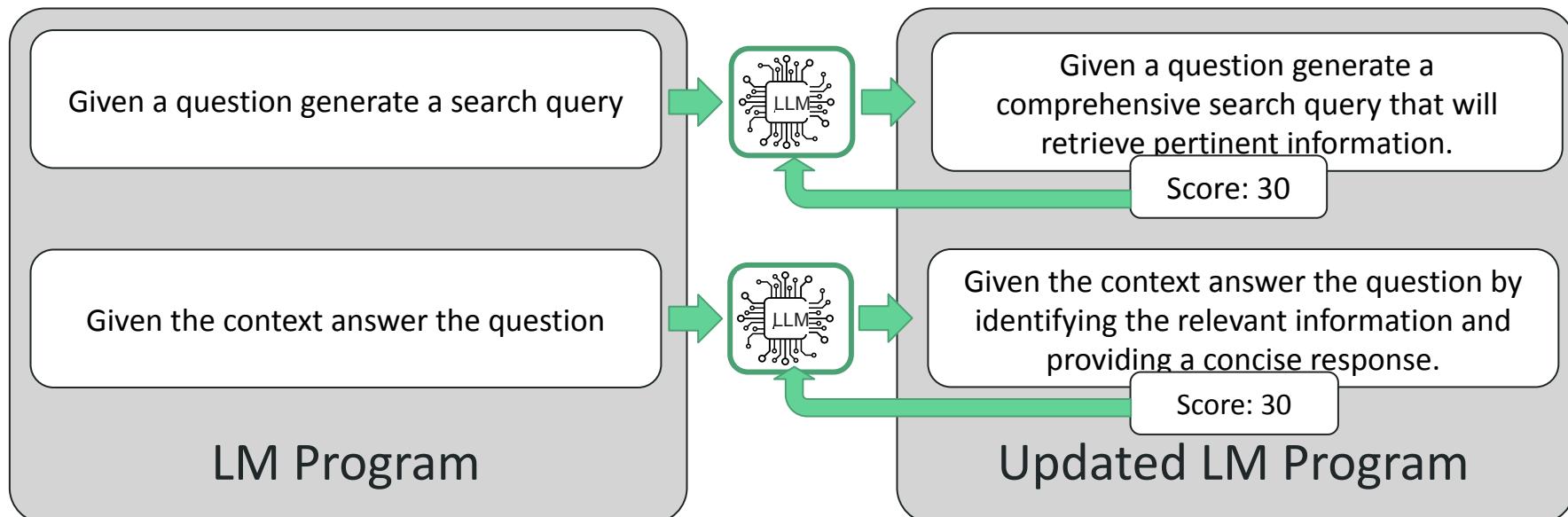


Key Idea: Coordinate-Ascent was expensive, maybe we don't need explicit credit assignment? Let's just change both prompts at a time in parallel!

Module-Level OPRO



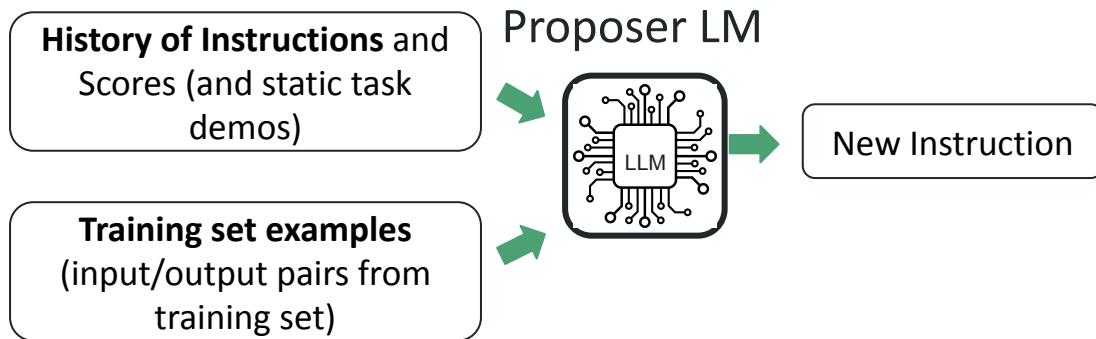
Key Idea: Coordinate-Ascent was expensive, maybe we don't need explicit credit assignment? Let's just change both prompts at a time in parallel!



Finally, Grounding!



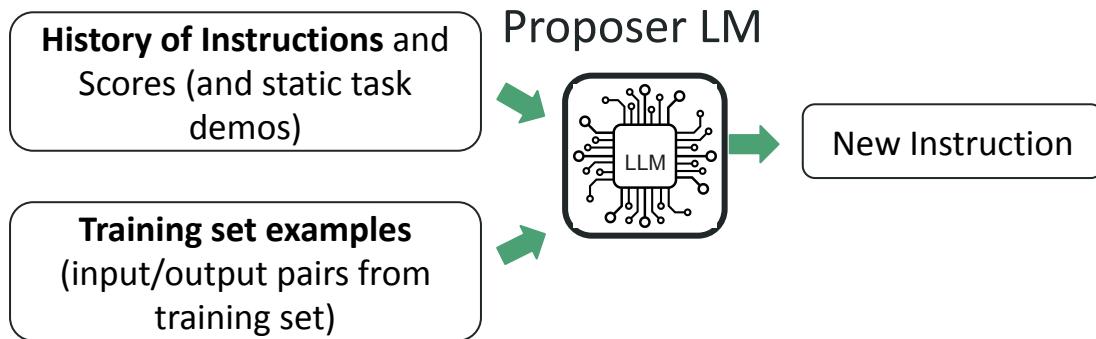
Hypothesis: Providing our proposer LM with more information relevant to the task can help us propose better instructions.



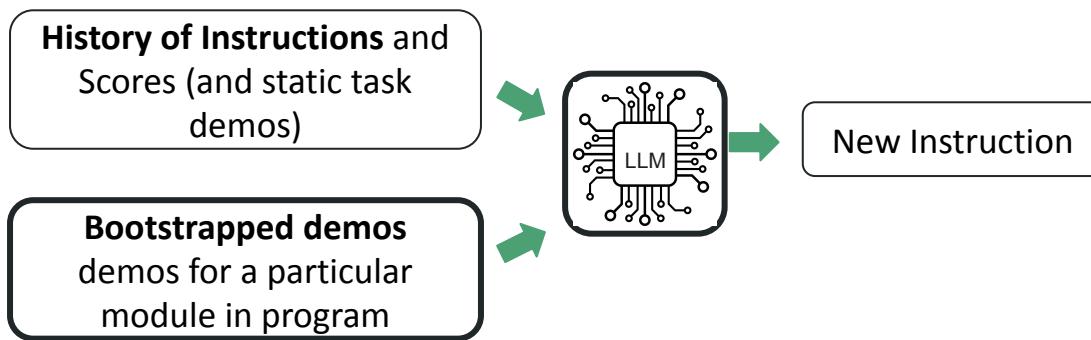
Finally, Grounding!



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



Finally, Grounding!



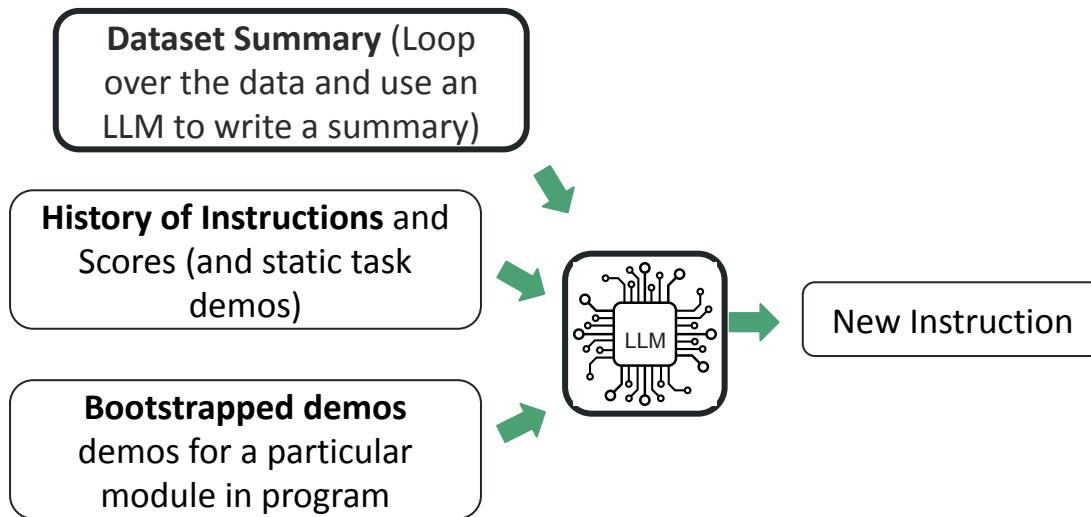
Bootstrapped demo example:

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"

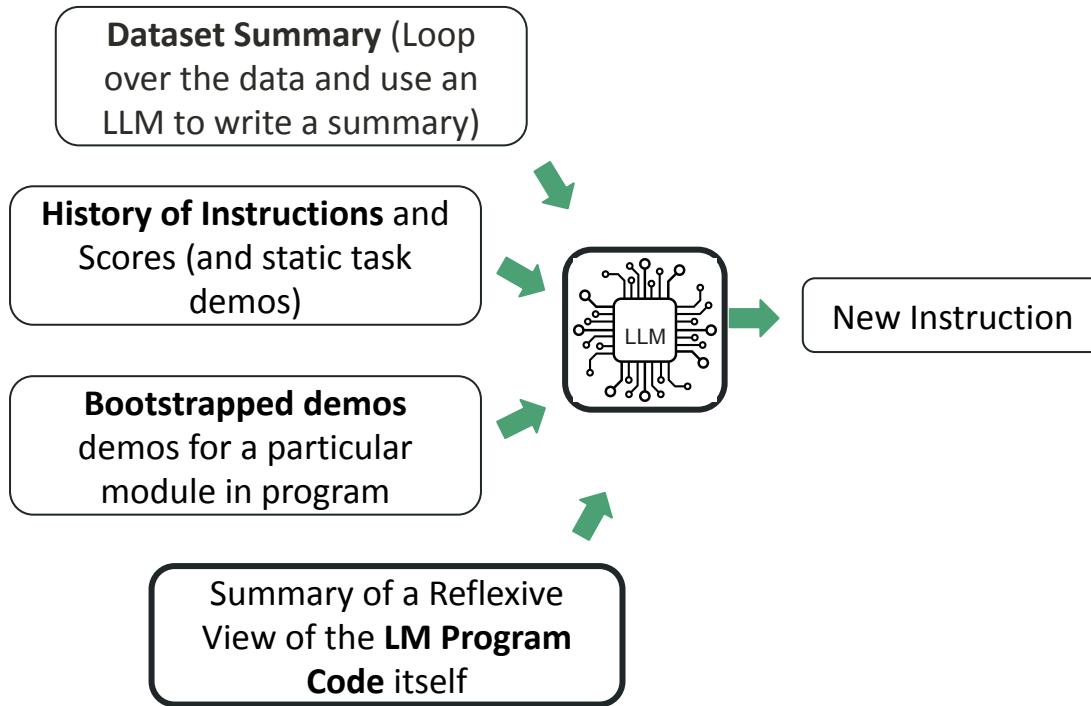
Finally, Grounding!



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.."

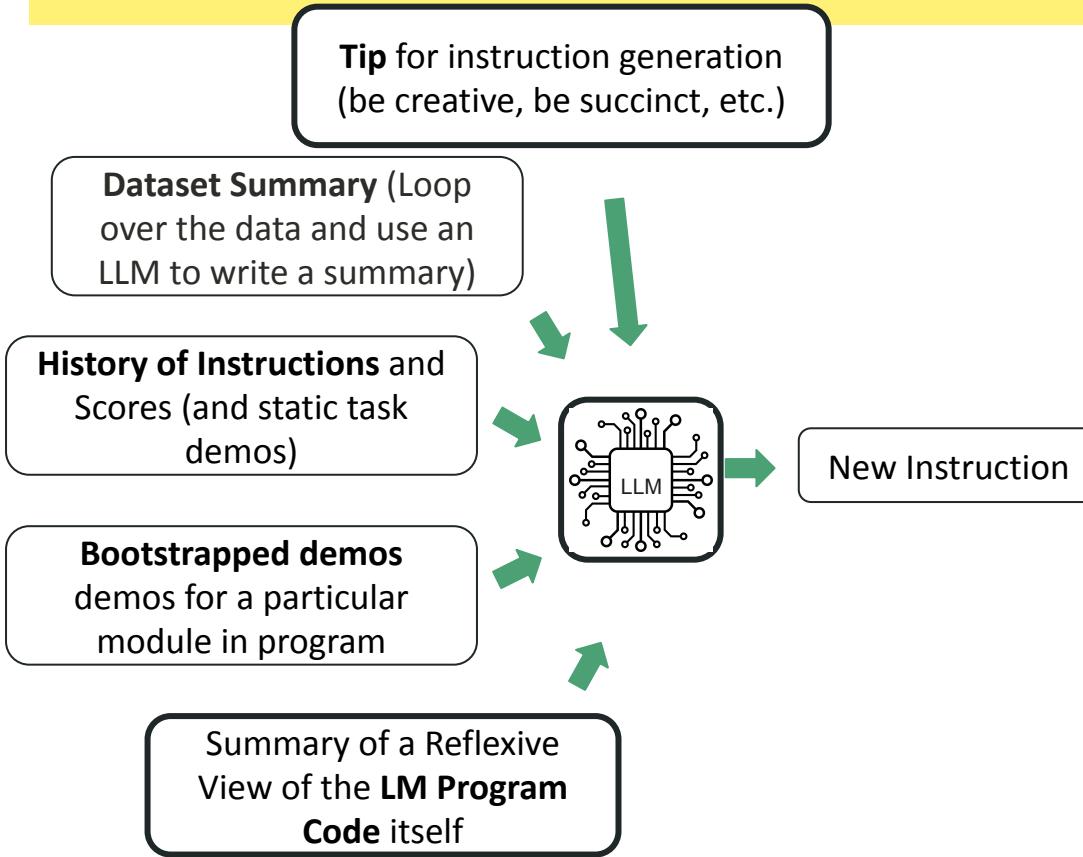
Finally, Grounding!



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**

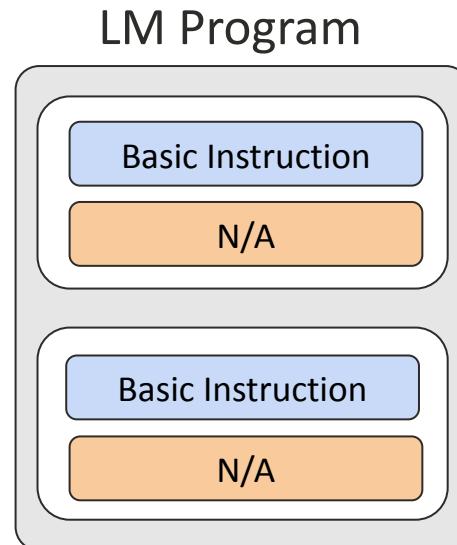
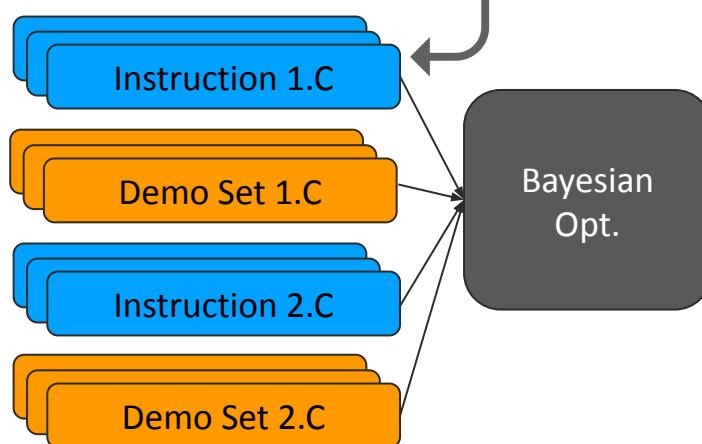
3. Jointly tune with a Bayesian
hyperparameter optimizer

Step 3: Optimize with Bayesian Learning



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

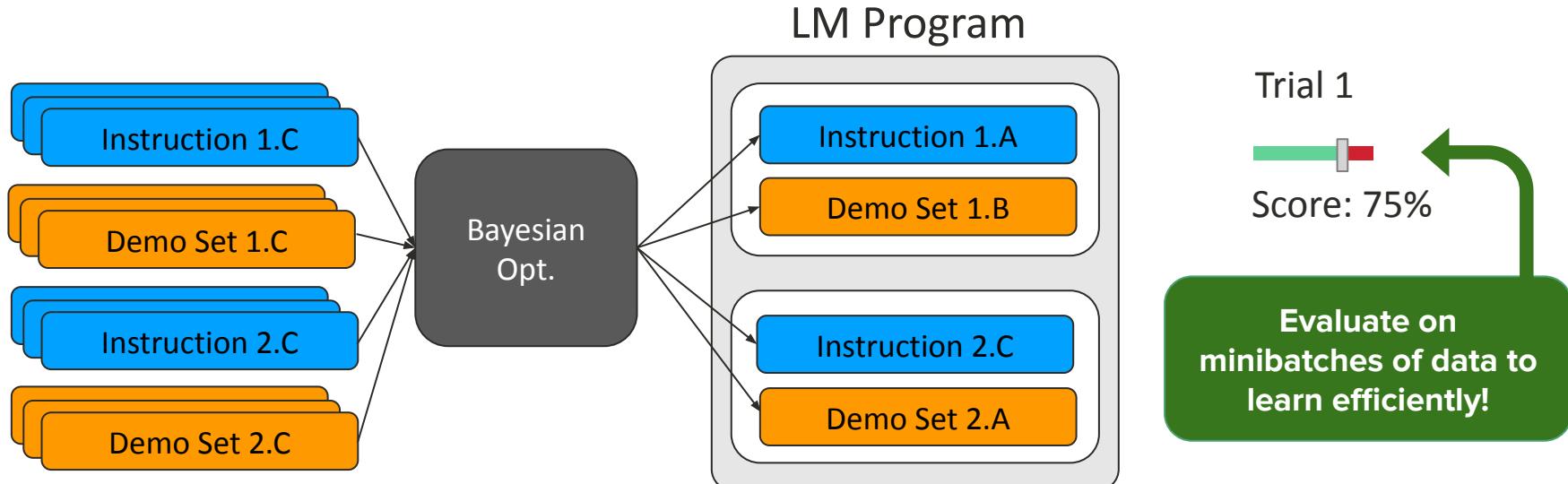
Set of instructions / fewshot candidates for each module:



Step 3: Optimize with Bayesian Learning



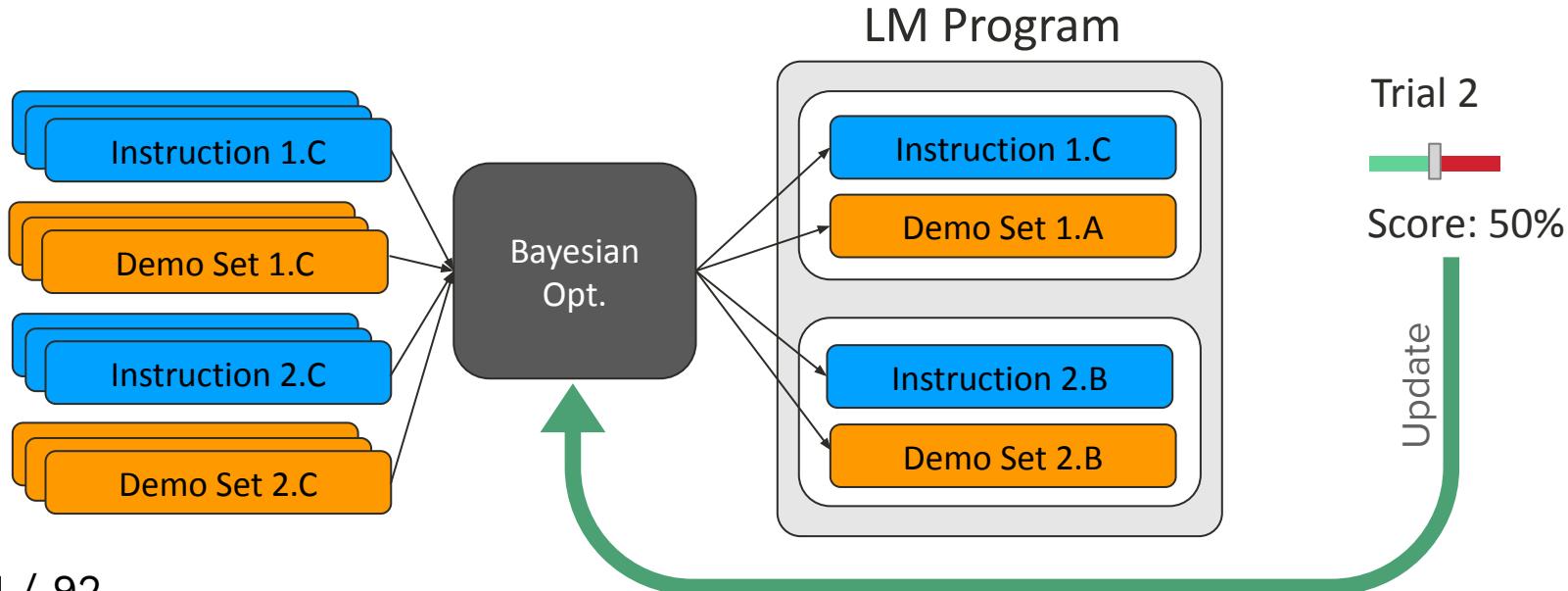
Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Step 3: Optimize with Bayesian Learning



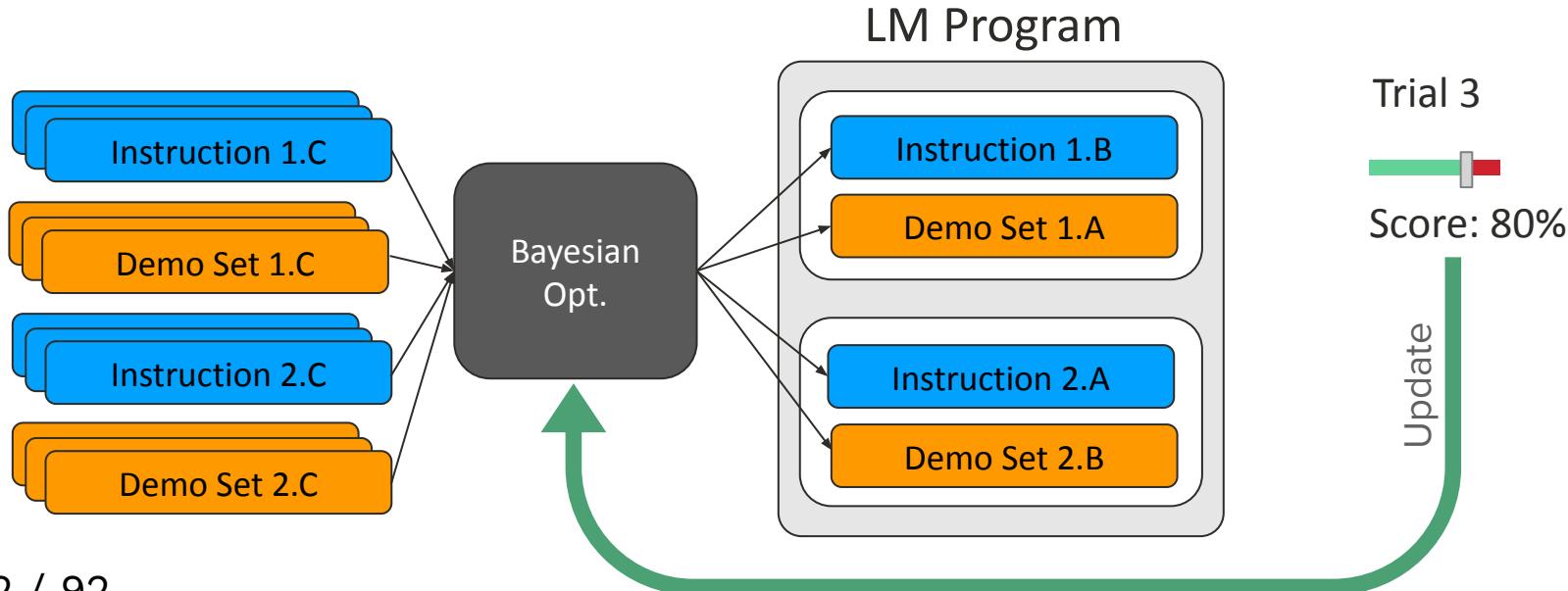
Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Step 3: Optimize with Bayesian Learning



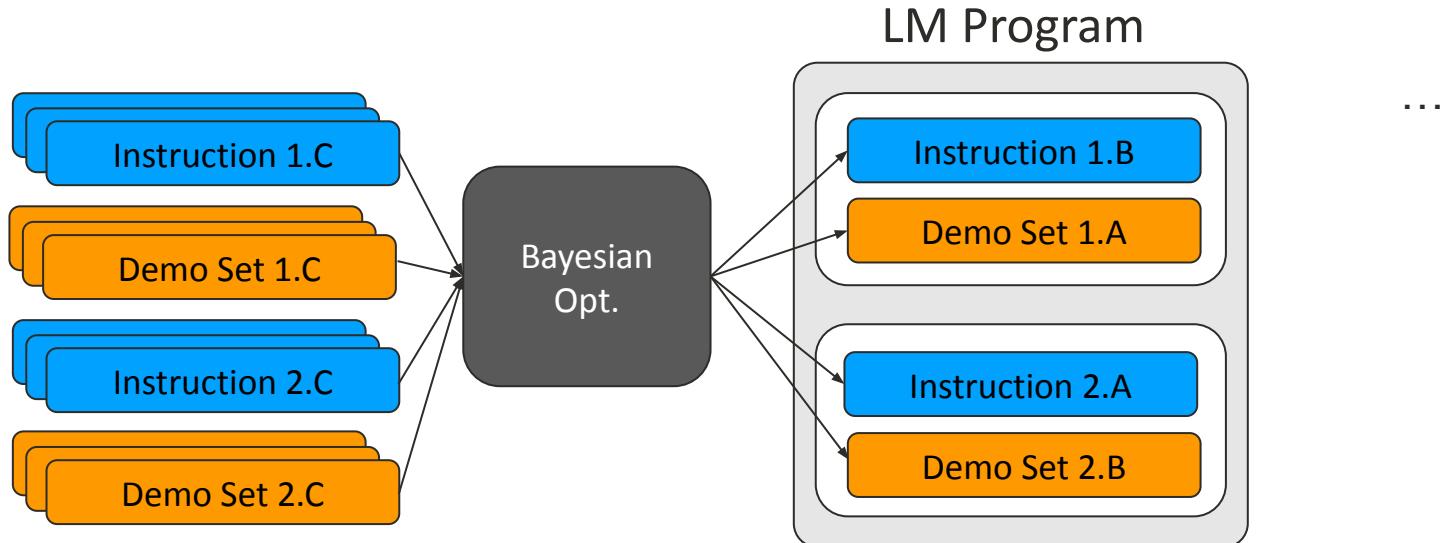
Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Step 3: Optimize with Bayesian Learning



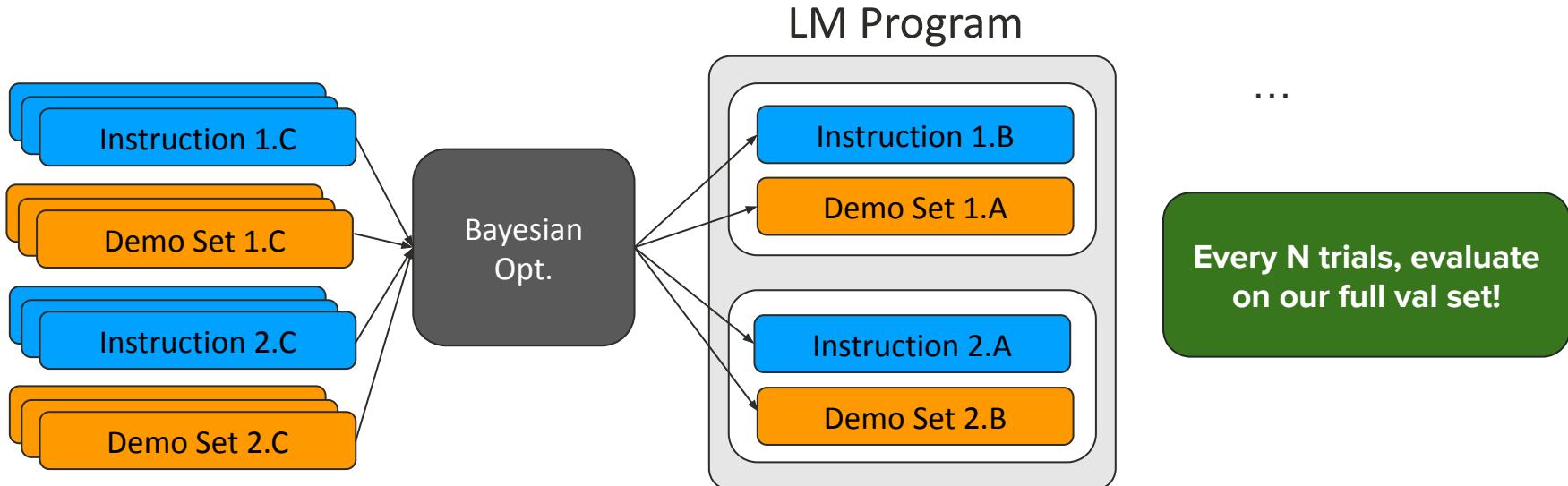
Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Step 3: Optimize with Bayesian Learning



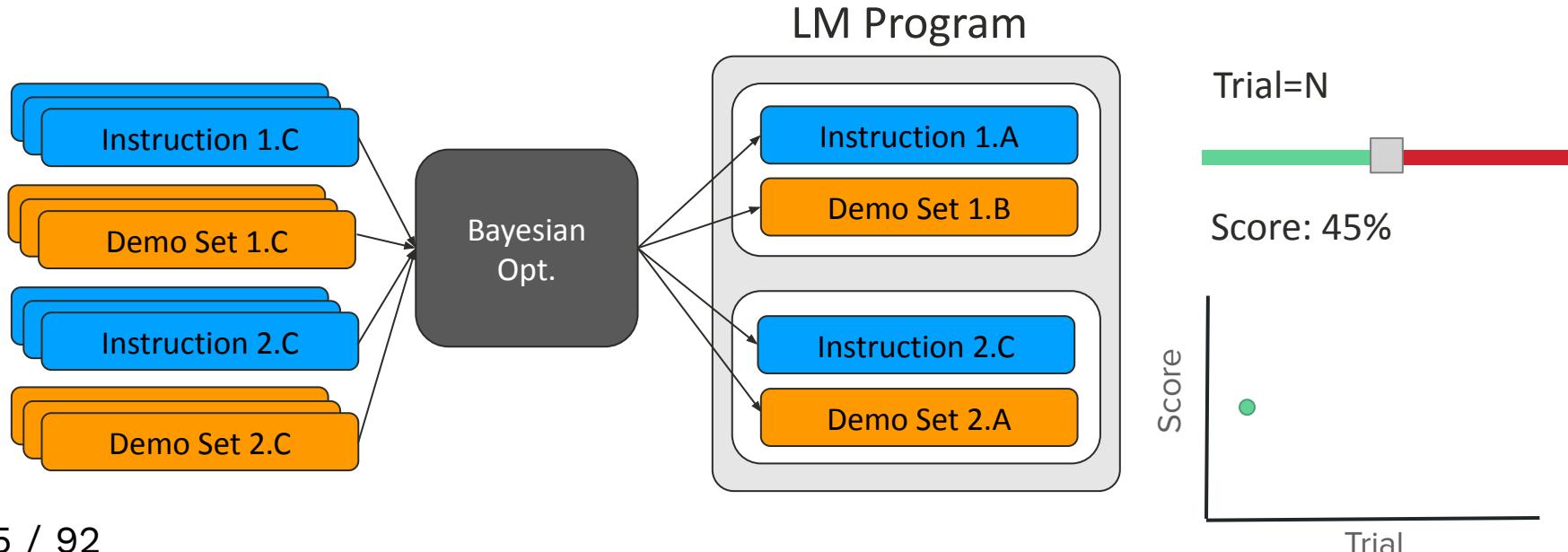
Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Step 3: Optimize with Bayesian Learning



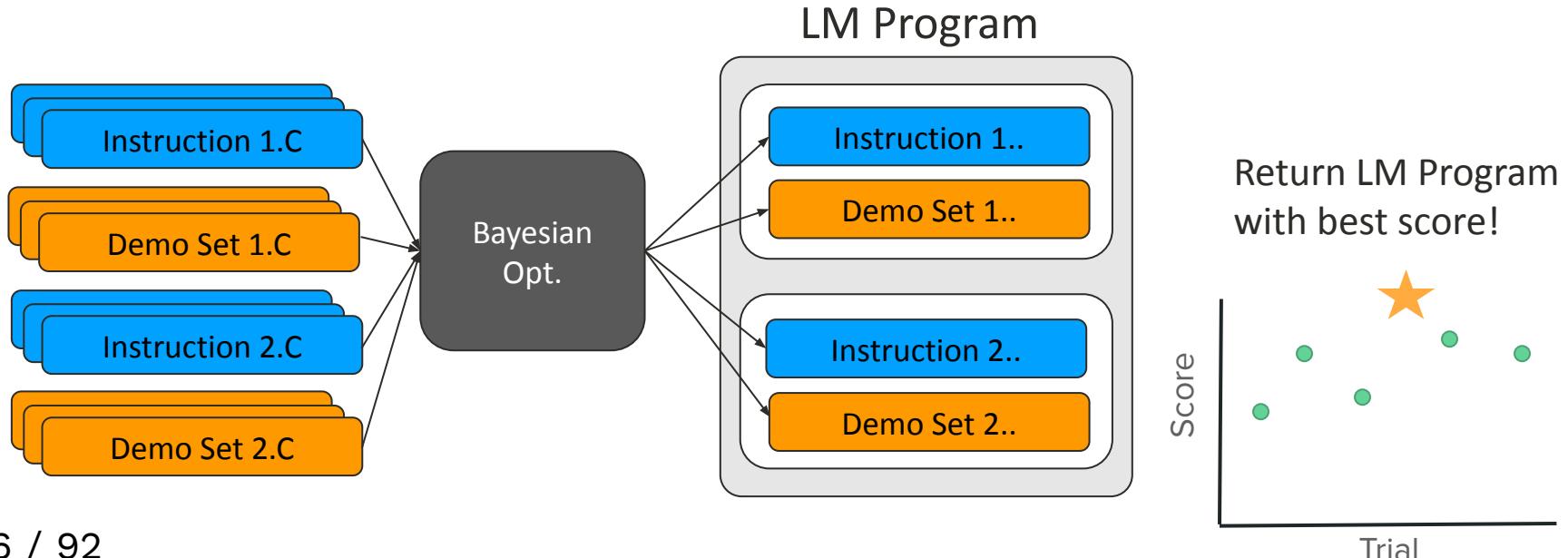
Key Idea: MIPRO uses a Bayesian Surrogate Model for Credit Assignment



Step 3: Optimize with Bayesian Learning



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



Experiments & Results

So how do these optimization methods compare?

Enter LangProBe, the Language Model Program Benchmark

| Benchmark | Task Type | Program | Modules | LM Calls | Metric |
|----------------------|----------------------------|---------------------|---------|----------|-------------|
| HotPotQA | Multi-Hop QA | Multi-Hop Retrieval | 2 | 3 | Exact Match |
| HotPotQA Conditional | Multi-Hop QA | Multi-Hop Retrieval | 2 | 3 | Custom |
| Iris | Classification | Chain of Thought | 1 | 1 | Accuracy |
| Heart Disease | Classification | Answer Ensemble | 2 | 4 | Accuracy |
| ScoNe | Natural Language Inference | Chain of Thought | 1 | 1 | Exact Match |
| HoVer | Multi-Hop Claim Verify | Multi-Hop Retrieval | 4 | 4 | Recall@21 |

So how do these optimization methods compare?

Enter LangProBe, the Language Model Program Benchmark

| Benchmark | Task Type | Program | Modules | LM Calls | Metric |
|----------------------|----------------------------|---------------------|---------|----------|-------------|
| HotPotQA | Multi-Hop QA | Multi-Hop Retrieval | 2 | 3 | Exact Match |
| HotPotQA Conditional | Multi-Hop QA | Multi-Hop Retrieval | 2 | 3 | Custom |
| Iris | Classification | Chain of Thought | 1 | 1 | Accuracy |
| Heart Disease | Classification | Answer Ensemble | 2 | 4 | Accuracy |
| ScoNe | Natural Language Inference | Chain of Thought | 1 | 1 | Exact Match |
| HoVer | Multi-Hop Claim Verify | Multi-Hop Retrieval | 4 | 4 | Recall@21 |

Hypothesis: Instructions become more important in tasks with multiple conditional rules, which cannot be fully expressed with a set # of few-shot ex.

| Optimizer | ScoNe | HotPotQA | HoVer | HotPotQA Cond. | Iris | Heart Disease |
|-----------------------------------|-------|----------|-------|-------------------|------|------------------|
| <i>Instructions only (0-shot)</i> | | | | | | |
| N/A | 69.1 | 36.1 | 25.3 | 6 | 32 | 26.8 |
| Module-Level OPRO –G | 76.1 | 36.0 | 25.7 | – | – | – |
| Module-Level OPRO | 73.5 | 39.0 | 32.5 | – | – | – |
| 0-Shot MIPRO | 71.5 | 36.8 | 33.1 | 14.6 | 56.7 | 25.8 |

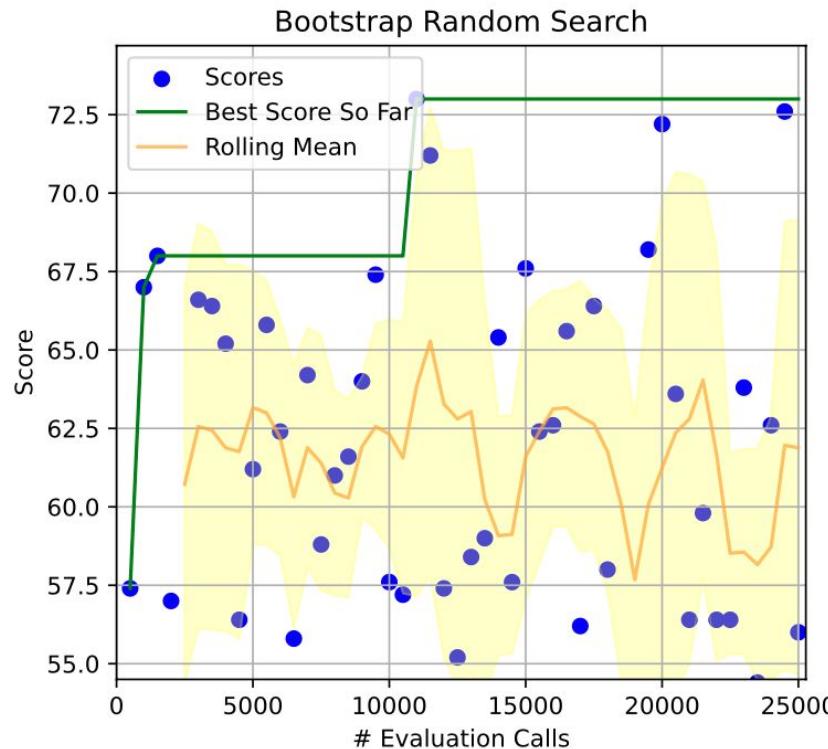
Optimizing instructions can deliver gains over baseline signatures.

| Optimizer | ScoNe | HotPotQA | HoVer | HotPotQA Cond. | Iris | Heart Disease |
|-----------------------------------|-------|----------|-------|-------------------|------|------------------|
| <i>Instructions only (0-shot)</i> | | | | | | |
| N/A | 69.1 | 36.1 | 25.3 | 6 | 32 | 26.8 |
| Module-Level OPRO –G | 76.1 | 36.0 | 25.7 | – | – | – |
| Module-Level OPRO | 73.5 | 39.0 | 32.5 | – | – | – |
| 0-Shot MIPRO | 71.5 | 36.8 | 33.1 | 14.6 | 56.7 | 25.8 |

However, there's no obvious best approach to instruction proposal yet.

| Optimizer | ScoNe | HotPotQA | HoVer | HotPotQA Cond. | Iris | Heart Disease |
|---------------------------------------|-------|-------------|-------|-------------------|------|------------------|
| <i>Instructions only (0-shot)</i> | | | | | | |
| N/A | 69.1 | 36.1 | 25.3 | 6 | 32 | 26.8 |
| Module-Level OPRO –G | 76.1 | 36.0 | 25.7 | – | – | – |
| Module-Level OPRO | 73.5 | 39.0 | 32.5 | – | – | – |
| 0-Shot MIPRO | 71.5 | 36.8 | 33.1 | 14.6 | 56.7 | 25.8 |
| <i>Demonstrations only (Few-shot)</i> | | | | | | |
| Bootstrap RS | 75.4 | 45.8 | 37.2 | 10.4 | 58.7 | 79.2 |
| Bayesian Bootstrap | 77.4 | 46.2 | 37.6 | – | – | – |

Optimizing bootstrapped demonstrations is key!



The bootstrapped demonstrations we choose matters a lot!
Understanding why is an area for future research.

| Optimizer | ScoNe | HotPotQA | HoVer | HotPotQA Cond. | Iris | Heart Disease |
|---------------------------------------|-------------|-------------|-------------|-------------------|-------------|------------------|
| <i>Instructions only (0-shot)</i> | | | | | | |
| N/A | 69.1 | 36.1 | 25.3 | 6 | 32 | 26.8 |
| Module-Level OPRO –G | 76.1 | 36.0 | 25.7 | – | – | – |
| Module-Level OPRO | 73.5 | 39.0 | 32.5 | – | – | – |
| 0-Shot MIPRO | 71.5 | 36.8 | 33.1 | 14.6 | 56.7 | 25.8 |
| <i>Demonstrations only (Few-shot)</i> | | | | | | |
| Bootstrap RS | 75.4 | 45.8 | 37.2 | 10.4 | 58.7 | 79.2 |
| Bayesian Bootstrap | 77.4 | 46.2 | 37.6 | – | – | – |
| <i>Both (Few-shot)</i> | | | | | | |
| MIPRO | 79.4 | 46.4 | 39.0 | 23.3 | 68.7 | 74.2 |

Optimizing both instructions and demonstrations via
MIPRO is often the most effective approach!

| Optimizer | ScoNe | HotPotQA | HoVer | HotPotQA Cond. | Iris | Heart Disease |
|---------------------------------------|-------------|-------------|-------------|-------------------|-------------|------------------|
| <i>Instructions only (0-shot)</i> | | | | | | |
| N/A | 69.1 | 36.1 | 25.3 | 6 | 32 | 26.8 |
| Module-Level OPRO –G | 76.1 | 36.0 | 25.7 | – | – | – |
| Module-Level OPRO | 73.5 | 39.0 | 32.5 | – | – | – |
| 0-Shot MIPRO | 71.5 | 36.8 | 33.1 | 14.6 | 56.7 | 25.8 |
| <i>Demonstrations only (Few-shot)</i> | | | | | | |
| Bootstrap RS | 75.4 | 45.8 | 37.2 | 10.4 | 58.7 | 79.2 |
| Bayesian Bootstrap | 77.4 | 46.2 | 37.6 | – | – | – |
| <i>Both (Few-shot)</i> | | | | | | |
| MIPRO | 79.4 | 46.4 | 39.0 | 23.3 | 68.7 | 74.2 |

The impact of optimizing instructions (rather than demonstrations) is more visible in tasks that have many isolated conditional rules.

Summary & Lessons

Key Lessons I: Natural Language Programming

1. Programs can often be more accurate, controllable, transparent, and even efficient than models.
2. You just need declarative programs, not implementation details. High-level optimizers can bootstrap prompts — or weights, or whatever the next paradigm deals with.

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~~ ⇒ Optimized programs

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

and is being widely used in production & OSS -- [dspy.ai](#)

at JetBlue, Databricks, Walmart, VMware, Replit, Haize Labs, Normal Computing, Sephora, Moody's...

 **replit**

← Back to blog

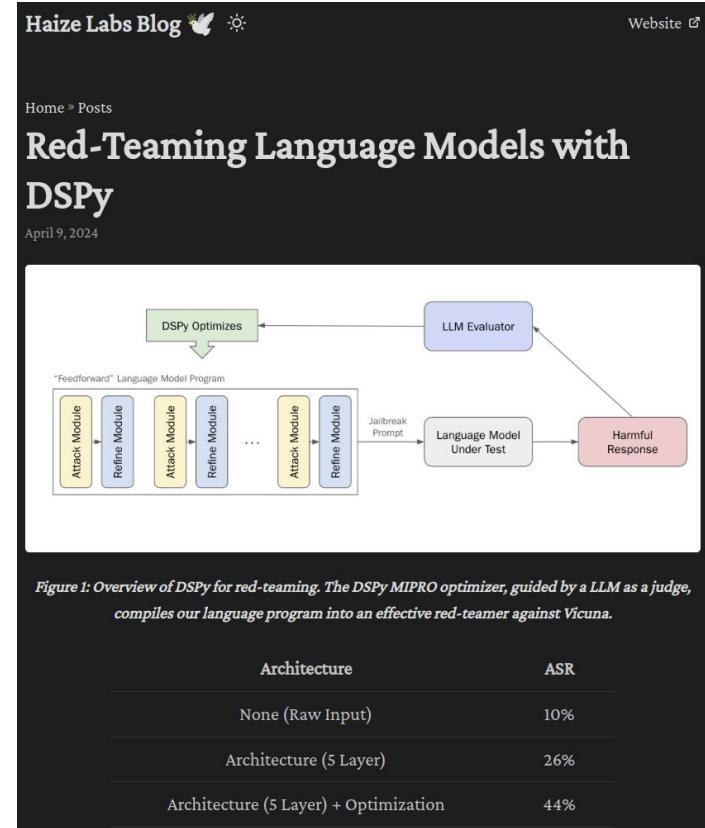
AI

Building LLMs for Code Repair



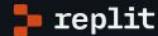
```
def count_replies(self, nums: List[int]) -> dict[int, int]:  
    counter = Counter(nums)  
    index = 0  
    for num, count in counter.items():  
        nums[index] == num  
        index += 1  
        if count >> 1:  
            ...  
    return nums
```

89 / 92



and is being widely used in production & OSS -- [dspy.ai](#)

at JetBlue, Databricks, Walmart, VMware, Replit, Haize Labs, Normal Computing, Sephora, Moody's...



The End of Prompting, The Beginning of Compound Systems

As more and more companies leverage LLMs, the limitations of a generic chatbot interface are increasingly clear. These off-the-shelf platforms are highly dependent on parameters that are outside the control of both end-users and administrators. By [constructing compound systems](#) that leverage a combination of LLM calls and traditional software development, companies can easily adapt and optimize these solutions to fit their use case. **DSPy** is enabling this paradigm shift toward modular, trustworthy LLM systems that can optimize themselves against any metric. With the power of Databricks and **DSPy**, JetBlue is able to deploy better LLM solutions at scale and push the boundaries of what is possible.

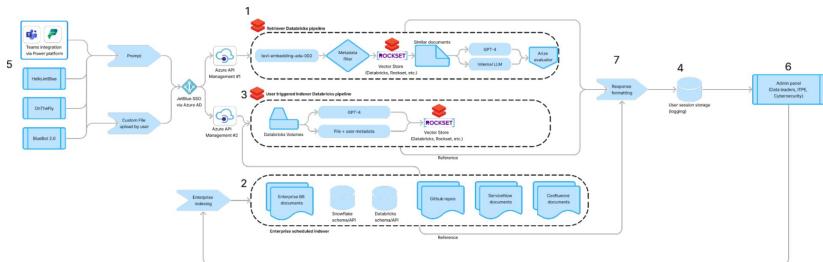
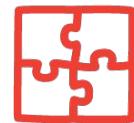


Figure 7: Using Databricks' solutions, JetBlue's complete chatbot architecture makes use of custom document uploads with different user groups



DSPy

Fork 1.2k

Starred 16.1k

downloads/month 160k

Contributors 206



Key Lessons II: Natural Language Optimization

1. In isolation, on many tasks nothing beats bootstrapping good demonstrations. **Show don't tell!**
2. Generating good instructions on top of these is possible, and is especially important for tasks with **conditional rules!**
3. But you will need **effective grounding**, and explicit forms of **credit assignment**.

Can open research again lead AI progress?

 **DSPy** (dspy.ai) aims to show that this lies in modularity.

- ✖ Not ever-larger, opaque LMs in isolation.
- ✖ Not ad-hoc tricks for prompting or synthetic data.

But well-scoped programs, better inference-time strategies, and new ways to optimize how LMs are used to solve tasks.