

DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines

For reliability, we often build pipelines of FMs

Break down problems and **delegate subtasks** to specialized FMs

How many storeys are there in the castle David Gregory inherited?

For reliability, we often build pipelines of FMs

Break down problems and delegate subtasks to specialized FMs

How many storeys are there in the castle David Gregory inherited?



LM

For reliability, we often build pipelines of FMs

Break down problems and delegate subtasks to specialized FMs

How many storeys are there in the castle David Gregory inherited?



LM



*Castle Gregory has **three** storeys.*

For reliability, we often build pipelines of FMs

Break down problems and delegate subtasks to specialized FMs

How many storeys are there in the castle David Gregory inherited?



LM



Castle Gregory has three storeys. ❌

For reliability, we often build pipelines of FMs

How many storeys are there in the castle David Gregory inherited?



LM



*Castle Gregory has **three** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?



Retriever

St. Gregory is a 9-floor boutique hotel...

For reliability, we often build pipelines of FMs

How many storeys are there in the castle David Gregory inherited?



LM



*Castle Gregory has **three** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?



Retriever

St. Gregory is a 9-floor boutique hotel...



LM



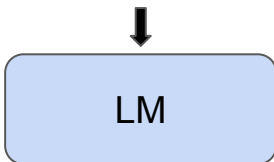
*St. Gregory Hotel has **nine** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?

For reliability, we often build pipelines of FMs

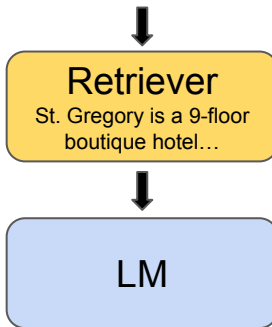
LM
What castle did David Gregory inherit?

How many storeys are there in the castle David Gregory inherited?



*Castle Gregory has **three** storeys.* ❌

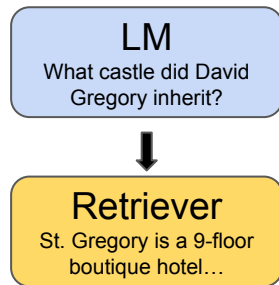
How many storeys are there in the castle David Gregory inherited?



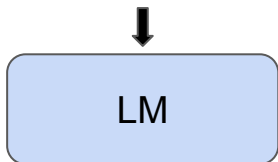
*St. Gregory Hotel has **nine** storeys.* ❌

For reliability, we often build pipelines of FMs

How many storeys are there in the castle David Gregory inherited?

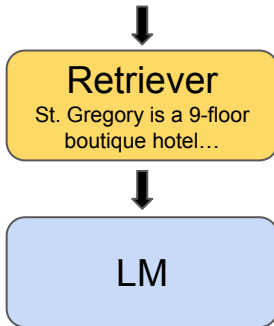


How many storeys are there in the castle David Gregory inherited?



*Castle Gregory has **three** storeys.* ❌

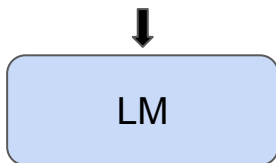
How many storeys are there in the castle David Gregory inherited?



*St. Gregory Hotel has **nine** storeys.* ❌

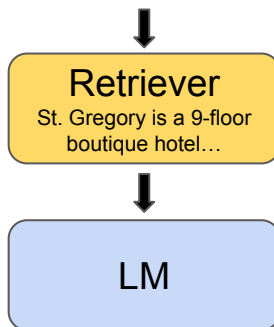
For reliability, we often build pipelines of FMs

How many storeys are there in the castle David Gregory inherited?



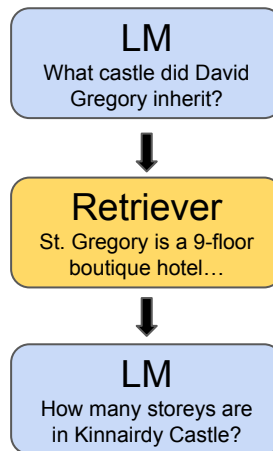
*Castle Gregory has **three** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?



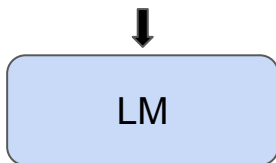
*St. Gregory Hotel has **nine** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?



For reliability, we often build pipelines of FMs

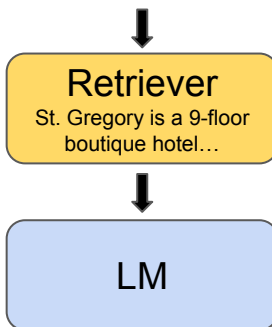
How many storeys are there in the castle David Gregory inherited?



*Castle Gregory has **three** storeys.*



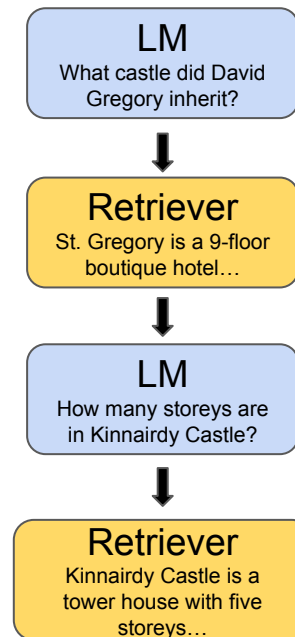
How many storeys are there in the castle David Gregory inherited?



*St. Gregory Hotel has **nine** storeys.*

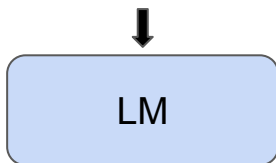


How many storeys are there in the castle David Gregory inherited?



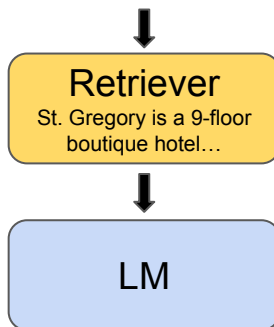
For reliability, we often build pipelines of FMs

How many storeys are there in the castle David Gregory inherited?



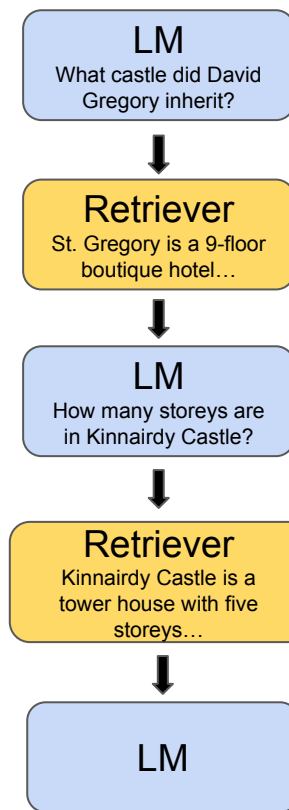
*Castle Gregory has **three** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?



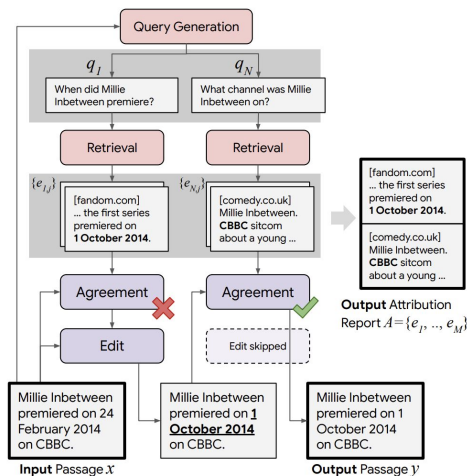
*St. Gregory Hotel has **nine** storeys.* ❌

How many storeys are there in the castle David Gregory inherited?



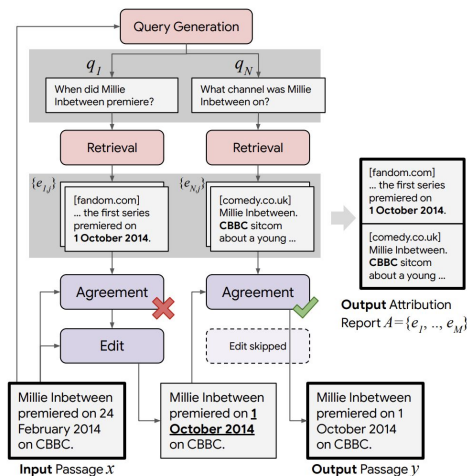
*Castle Gregory has **five** storeys.* ✅

For reliability, we often build pipelines of FMs

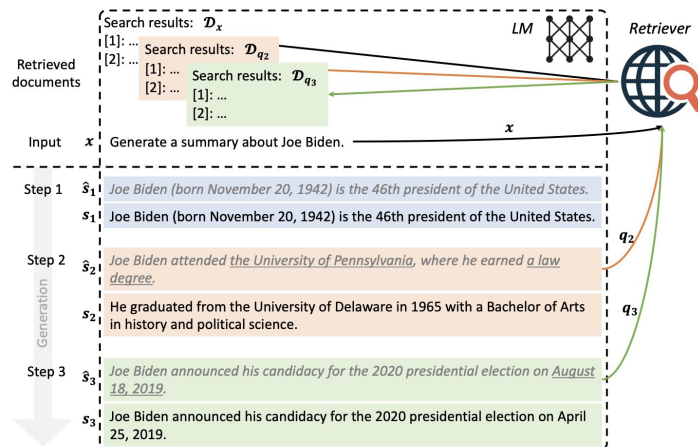


RARR - a pipeline of
hand-prompted LLMs and a
retriever [Gao et al. 2023]

For reliability, we often build pipelines of FMs

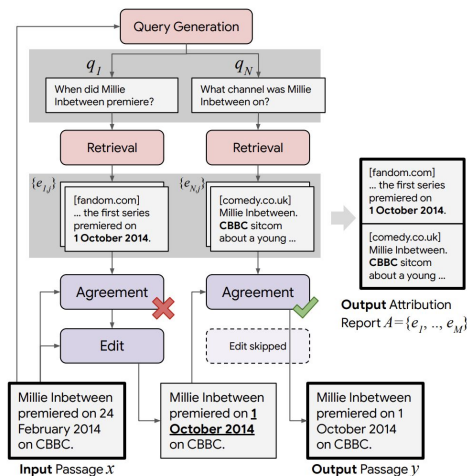


RARR - a pipeline of hand-prompted LLMs and a retriever [Gao et al. 2023]

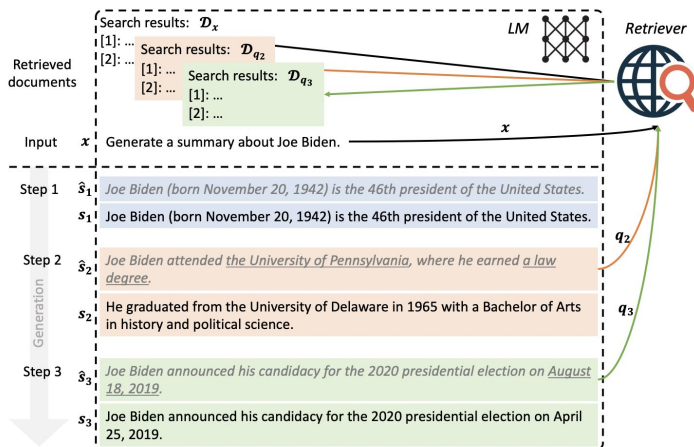


FLARE - a pipeline of hand-prompted LLMs and a retriever [Jiang et al. 2023]

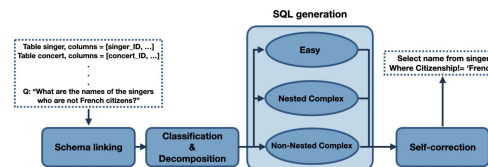
For reliability, we often build pipelines of FMs



RARR - a pipeline of hand-prompted LLMs and a retriever [Gao et al. 2023]

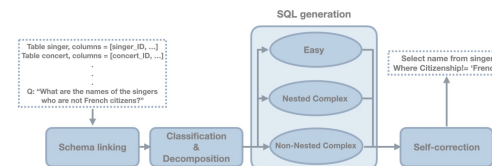
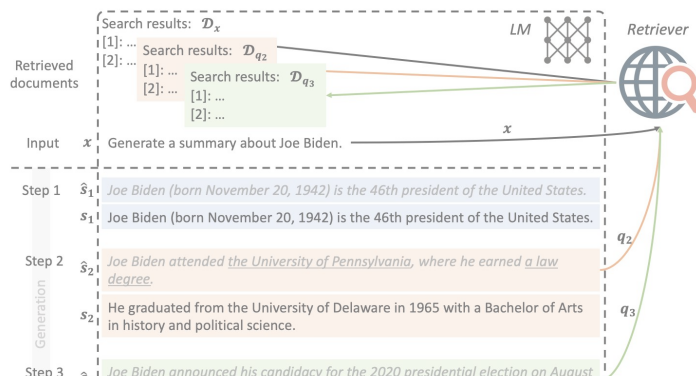
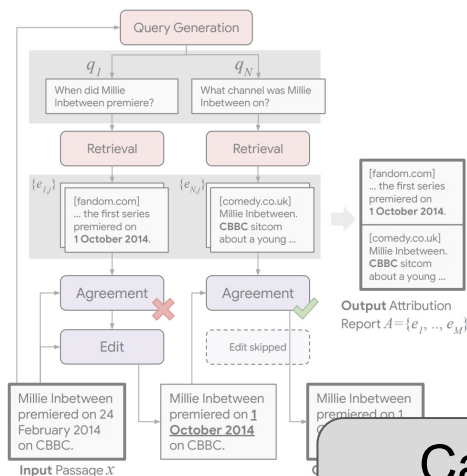


FLARE - a pipeline of hand-prompted LLMs and a retriever [Jiang et al. 2023]



DIN-SQL - a pipeline of hand-prompted LLMs and a SQL DB [Pourreza et al. 2023]

For reliability, we often build pipelines of FMs



Can we develop a **systematic** approach to design **LM pipelines**?

RARR - a pipeline of hand-prompted LLMs and a retriever [Gao et al. 2023]

LLMs and a retriever [Jiang et al. 2023]

line of LMs and a SQL DB [23]

But building **effective pipelines** isn't easy

It is not clear how to **connect and optimize all the pieces** of a pipeline together.

- Pipeline prompts are **extremely fragile** and non-transferable.
- Pipeline finetunes are **very complicated!**

But building **effective pipelines** isn't easy

It is not clear how to **connect and optimize all the pieces** of a pipeline together.

- Pipeline prompts are **extremely fragile** and non-transferable.
- Pipeline finetunes are **very complicated!**

Right now, many NLP researchers are writing demonstrations - and often long free-form prompts by hand!

Create a high-level plan for completing a household task using the allowed actions and visible objects.

Allowed actions: OpenObject, CloseObject, PickupObject, PutObject, ToggleObjectOn, ToggleObjectOff, SliceObject, Navigation

Task description: Put a heated egg in the sink.

Completed plan: Navigation fridge, OpenObject fridge, PickupObject egg, CloseObject fridge

Visible objects are sink, egg, microwave

Next plan: Navigation microwave, OpenObject microwave, PutObject egg microwave, CloseObject microwave, ToggleObjectOn microwave, ToggleObjectOff microwave, OpenObject microwave, PickupObject egg, CloseObject microwave, Navigation sinkbasin, PutObject egg sinkbasin

Task description: Cook a potato and put it into the recycle bin.

Completed plan: Navigation fridge, OpenObject fridge, PickupObject potato, CloseObject fridge, Navigation microwave, OpenObject microwave, PutObject potato microwave, CloseObject microwave, ToggleObjectOn microwave, ToggleObjectOff microwave, OpenObject microwave, PickupObject potato, CloseObject microwave

Visible objects are microwave, fridge, potato, garbagecan

Next plan:

Table 1. Prompt for SEEACTION Action Generation with GPT-4V.

Imagine that you are imitating humans doing web navigation for a task step by step. At each stage, you can see the webpage like humans by a screenshot and know the previous actions before the current step decided by yourself through recorded history. You need to decide on the first following action to take. You can click an element with the mouse, select an option, or type text with the keyboard. (For your understanding, they are like the click(), select_option() and type() functions in playwright respectively) One next step means one operation within the three.

You are asked to complete the following task: {TASK}

Previous Actions:
{PREVIOUS ACTIONS}

The screenshot below shows the webpage you see. Follow the following guidance to think step by step before outlining the next action step at the current stage:

(Current Webpage Identification)
Firstly, think about what the current webpage is.

(Previous Action Analysis)

Secondly, combined with the screenshot, analyze each step of the previous action history and their intention one by one. Particularly, pay more attention to the last step, which may be more related to what you should do now as the next step.

(Screenshot Details Analysis)

Closely examine the screenshot to check the status of every part of the webpage to understand what you can operate with and what has been set or completed. You should closely examine the screenshot details to see what steps have been completed by previous actions even though you are given the textual previous actions. Because the textual history may not clearly and sufficiently record some effects of previous actions, you should closely evaluate the status of every part of the webpage to understand what you have done.

(Next Action Based on Webpage and Analysis)

Then, based on your analysis, in conjunction with human web browsing habits and the logic of web design, decide on the following action. And clearly outline which element in the webpage users will operate with as the first next target element, its detailed location, and the corresponding operation.

To be successful, it is important to follow the following rules:

1. You should only issue a valid action given the current observation.
2. You should only issue one action at a time

But building **effective pipelines** isn't easy

It is not clear how to **connect and optimize all the pieces** of a pipeline together.

- Pipeline prompts are **extremely fragile** and non-transferable.
- Pipeline finetunes are **very complicated!**

Right now, many NLP researchers are writing demonstrations - and often long free-form prompts by hand!



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

You are an intelligent shopping assistant that can help users find the right item. You are given an observation of the current web navigation session, in the following format:

Current Observation:

WebShop

Instruction:

{the user instruction}

[button] Search [button_] (generate a search query based on the user instruction and select this button to find relevant items)

Every button in the observation represents a possible action you can take. Based on the current observation, your task is to generate a rationale about the next action you should take. Note that if an history of past rationales and actions is provided, you should also consider the history when generating the rationale.

The problem

- Talking to ChatGPT requires lengthy instructions and lots of mistranslations
 - Sounds imperative?
 - Reminiscent of Siri



- Use declarative as cleaner solution
 - Like programming, needs a compiler to generate complex instructions from simpler commands

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)
2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)
2. **Modules** generalize prompting techniques (and compose into multi-stage systems)
3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)
2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

dspy.ChainOfThought

dspy.ReAct

dspy.MultiChainComparison

dspy.ProgramOfThought

3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)

question -> answer long_document -> summary context, question -> rationale, response passage -> main entities

2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

dspy.ChainOfThought dspy.ReAct dspy.MultiChainComparison dspy.ProgramOfThought

3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)

question -> answer long_document -> summary context, question -> rationale, response passage -> main entities

2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

dspy.ChainOfThought dspy.ReAct dspy.MultiChainComparison dspy.ProgramOfThought

3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

BootstrapFewShot BootstrapFinetune BootstrapFewShotWithRandomSearch BootstrapFewShotWithOptuna

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)

question -> answer long_document -> summary context, question -> rationale, response passage -> main entities

2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

dspy.ChainOfThought dspy.ReAct dspy.MultiChainComparison dspy.ProgramOfThought

3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

BootstrapFewShot BootstrapFinetune BootstrapFewShotWithRandomSearch BootstrapFewShotWithOptuna

When in doubt, assume it's PyTorch and LM is your device

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)
2. **Modules** generalize prompting techniques (and compose into multi-stage systems)
Modules are parameterized layers (or compositions).
3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

When in doubt, assume it's PyTorch and LM is your device

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)

Signatures are the in/out dimensions (function signature) of a module.

2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

Modules are parameterized layers (or compositions).

3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

When in doubt, assume it's PyTorch and LM is your device

The **DSPy** paradigm: Let's *program* - not prompt - LMs

Shift focus from tweaking the LM or prompts to good overarching system design.

Declarative **S**elf-improving Language **P**rograms. How?

1. **Signatures** abstract prompts (and fine-tuned LM weights)

Signatures are the in/out dimensions (function signature) of a module.

2. **Modules** generalize prompting techniques (and compose into multi-stage systems)

Modules are parameterized layers (or compositions).

3. **Teleprompters** optimize systems (i.e. complex modules) toward an arbitrary metric

Teleprompters are optimizers.

When in doubt, assume it's PyTorch and LM is your device

Signatures

- A signature consists of three simple elements
 - A minimal **description of the sub-task** that the LM is supposed to solve
 - A description of one of more **input fields** (e.g., input question) that is given as input to the LM
 - A description of one of more **output fields** (e.g., answer) that we expect from the LM.
 - DSPy will parse this notation and expand the field names into meaningful instructions for the LM

Signatures

- A signature consists of three simple elements
 - A minimal **description of the sub-task** that the LM is supposed to solve
 - A description of one of more **input fields** (e.g., input question) that is given as input to the LM
 - A description of one of more **output fields** (e.g., answer) that we expect from the LM.
 - DSPy will parse this notation and expand the field names into meaningful instructions for the LM

```
1 qa = dspy.Predict("question -> answer")
2 qa(question="Where is Guaraní spoken?")
3 # Out: Prediction(answer='Guaraní is spoken mainly in South America.')
```

What about more complex tasks?

For some advanced tasks, you need more verbose signatures. This is typically to:

1. Clarify something about the nature of the task (expressed below as a docstring).
2. Supply hints on the nature of an input field, expressed as a 'desc' keyword argument for 'dspy.InputField'.
3. Supply constraints on an output field, expressed as a 'desc' keyword argument for 'dspy.OutputField'.

What about more complex tasks?

Example D: A metric that evaluates faithfulness to citations

```
class CheckCitationFaithfulness(dspy.Signature):  
    """Verify that the text is based on the provided context.""" docstring  
    context = dspy.InputField(desc="facts here are assumed to be true") Input field description  
    text = dspy.InputField() Output field description  
    faithfulness = dspy.OutputField(desc="True/False indicating if text is faithful to context")  
  
context = "The 21-year-old made seven appearances for the Hammers and netted his only goal for them in  
  
text = "Lee scored 3 goals for Colchester United."  
  
faithfulness = dspy.ChainOfThought(CheckCitationFaithfulness)  
faithfulness(context=context, text=text)
```

Teleprompters

- *Tele*-prompters: Abstracting and automating the task of prompting
- Teleprompters are powerful optimizers that can take any program and learn to bootstrap and select effective prompts for its modules.
- Hence the name “*prompting at a distance*”.

```
1 tp = BootstrapFewShotWithRandomSearch(metric=gsm8k_accuracy)
2 bootstrap = tp.compile(program, trainset=trainset, valset=devset)
```

Teleprompter Examples

- **LabeledFewshot**
 - Sample random GT question–answer pairs from train set.

Teleprompter Examples

- **BootstrapFewShot**

- Bootstrapping few-shot examples and their reasoning traces for ICL, optionally using a teacher program.
- Keep those demonstrations which satisfy a given metric.
- Advantages: Store reasoning traces from a teacher, can be used in CoT.

Teleprompter Examples

- **BootstrapFewShotWithRandomSearch**
 - Random search in the dataset for few-shot examples in ICL.
 - The training dataset is shuffled each time.
 - Use the validation set to select the best performing candidate program according to specified metric.

Teleprompter Examples

- **BootstrapFewShotWithOptuna**
 - Hyperparam optimization with Optuna
 - Optimize hyperparameters such as demonstrations, instructions, etc.

Teleprompter Examples

- **BootstrapFinetune**
 - Fine-tune a target LM on demonstrations generated by a teacher program.

Teleprompter Examples

- **Optimizers for instructions**
 - SignatureOptimizer
 - BayesianSignatureOptimizer

Let's get concrete: Math Word Problems with GSM8K

Question: Joseph and his friends watched two movies in his house. The first movie is 1 hour and 30 minutes long while the second movie is 30 minutes longer than the first. Before the movies, they spent 10 minutes making popcorn and twice as long making fries. How long, in hours, did it take Joseph and his friends to cook and watch the movies?

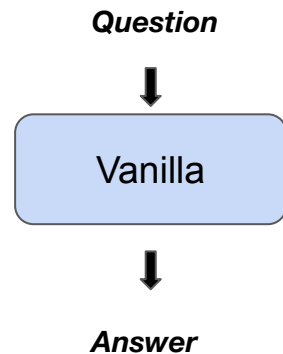
Let's get concrete: Math Word Problems with GSM8K

Question: Joseph and his friends watched two movies in his house. The first movie is 1 hour and 30 minutes long while the second movie is 30 minutes longer than the first. Before the movies, they spent 10 minutes making popcorn and twice as long making fries. How long, in hours, did it take Joseph and his friends to cook and watch the movies?

Answer: 4

Case Study: GSM8K

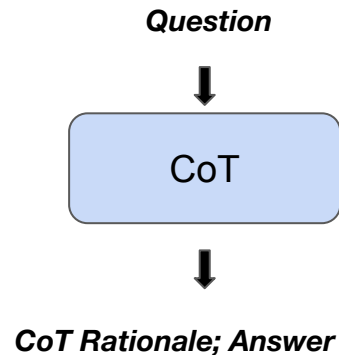
```
vanilla = dsp.Predict("question -> answer")
```



Case Study: GSM8K

```
vanilla = dspy.Predict("question -> answer")
```

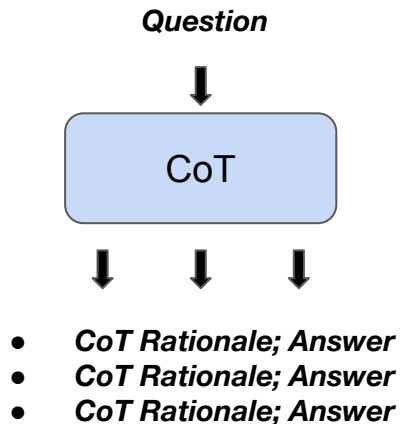
```
CoT = dspy.ChainOfThought("question -> answer")
```



Case Study: GSM8K

```
vanilla = dspy.Predict("question -> answer")
```

```
CoT = dspy.ChainOfThought("question -> answer")
```



Case Study: GSM8K

```
vanilla = dspy.Predict("question -> answer")
```

```
CoT = dspy.ChainOfThought("question -> answer")
```

```
1 vanilla = dspy.Predict("question -> answer") # GSM8K Program 'vanilla'
2
3 CoT = dspy.ChainOfThought("question -> answer") # GSM8K Program 'CoT'
```

```
1 class ThoughtReflection(dspy.Module):
2     def __init__(self, num_attempts):
3         self.predict = dspy.ChainOfThought("question -> answer", n=num_attempts)
4         self.compare = dspy.MultiChainComparison('question -> answer', M=num_attempts)
5
6     def forward(self, question):
7         completions = self.predict(question=question).completions
8         return self.compare(question=question, completions=completions)
9
10 reflection = ThoughtReflection(num_attempts=5) # GSM8K Program 'reflection'
```

Question



CoT



- **CoT Rationale; Answer**
- **CoT Rationale; Answer**
- **CoT Rationale; Answer**



Comparison



CoT Rationale; Answer

Now that we have programs, we can **compile** them.

```
improved_dspy_program = teleprompter.compile(dspy_program, few_examples)
```



Let's compile our program

```
trainset = [ds.py.Example(question="Joseph and his friends watched two movies...", answer="4"), ...]  
CoT = ds.py.ChainOfThought("question -> answer")  
  
teleprompter = ds.py.BootstrapFewShotWithRandomSearch(metric=gsm8k_accuracy)  
bootstrapped_program = teleprompter.compile(CoT, trainset=trainset)
```

```
1  Given the fields 'question', produce the fields 'answer'.  
2  
3  ---  
4  
5  Follow the following format.  
6  
7  Question: ${question}  
8  Reasoning: Let's think step by step in order to produce the answer. We ...  
9  Answer: ${answer}  
10  
11 ---  
12  
13 Question: Mark is baking bread. He has to let it rise for 120 minutes twice. He also needs to spend 10 minutes kneading  
14 it and 30 minutes baking it. How many minutes does it take Mark to finish making the bread?  
15 Reasoning: Let's think step by step in order to find out how many minutes it takes Mark to finish making the bread. We know  
16 that he needs to let it rise for 120 minutes twice, so that's 240 minutes. Then, he needs to spend 10 minutes kneading it  
17 and 30 minutes baking it. So, in total, it will take Mark  $240 + 10 + 30 = 280$  minutes to finish making the bread.  
18 Answer: 280 ---  
19 ---  
20 Question: Ben has $2000 for his business operations costs. He orders goods from his supplier and writes them a cheque for  
21 $600. His debtor pays him $800 from the purchases they had made on credit. Mr. Ben then decides to do equipment maintenance  
22 and spends $1200 on the whole operation. How much money is Mr. Ben remaining with?  
23 Reasoning: Let's think step by step in order to find out how much money Mr. Ben is remaining with. We know that he had  
24 $2000 to begin with, and he spent $600 on goods from his supplier, so he has  $2000 - 600 = 1400$  left. Then, his debtor  
25 paid him $800, so he has  $1400 + 800 = 2200$ . Finally, he spent $1200 on equipment maintenance, so he has  $2200 - 1200$   
26 = 1000 left.  
27 Answer: 1000 ---  
28 ---  
29 ... several other demonstrations here ...  
30  
31 ---  
32 Question:
```

Figure 10: Shortened copy of the prompt automatically generated by DSPy for GSM8K Llama2-13b-chat CoT program compiled with bootstrap.

Let's compile our program

```
1 fewshot = dspy.LabeledFewShot(k=8).compile(program, trainset=trainset)
```

Let's compile our program

```
1 fewshot = dspy.LabeledFewShot(k=8).compile(program, trainset=trainset)
```

```
1 tp = BootstrapFewShotWithRandomSearch(metric=gsm8k_accuracy)  
2 bootstrap = tp.compile(program, trainset=trainset, valset=devset)
```

Let's compile our program

```
1 fewshot = dspy.LabeledFewShot(k=8).compile(program, trainset=trainset)
```

```
1 tp = BootstrapFewShotWithRandomSearch(metric=gsm8k_accuracy)  
2 bootstrap = tp.compile(program, trainset=trainset, valset=devset)
```

```
1 bootstrap2 = tp.compile(program, teacher=bootstrap, trainset=trainset, valset=devset)
```

Let's compile our program

```
1 fewshot = dspy.LabeledFewShot(k=8).compile(program, trainset=trainset)
```

```
1 tp = BootstrapFewShotWithRandomSearch(metric=gsm8k_accuracy)
2 bootstrap = tp.compile(program, trainset=trainset, valset=devset)
```

```
1 bootstrap2 = tp.compile(program, teacher=bootstrap, trainset=trainset, valset=devset)
```

```
1 # A program that ensembles the top-7 candidate programs from a bootstrapping compiler run
  (in particular 'bootstrap' or, when applicable, 'bootstrap2') with majority voting.
2 ensemble = Ensemble(reduce_fn=dspy.majority).compile(bootstrap.programs[:7])
```

GSM8K Results

Program	Compilation	Training	GPT-3.5		Llama2-13b-chat	
			Dev	Test	Dev	Test
vanilla	none	n/a	24.0	25.2	7.0	9.4
	fewshot	trainset	33.1	–	4.3	–
	bootstrap	trainset	44.0	–	28.0	–
	bootstrap×2	trainset	64.7	61.7	37.3	36.5
	+ensemble	trainset	62.7	61.9	39.0	34.6
CoT	none	n/a	50.0	–	26.7	–
	fewshot	trainset	63.0	–	27.3	–
	fewshot	+human_CoT	78.6	72.4	34.3	33.7
	bootstrap	trainset	80.3	72.9	43.3	–
	+ensemble	trainset	88.3	81.6	43.7	–
reflection	none	n/a	65.0	–	36.7	–
	fewshot	trainset	71.7	–	36.3	–
	bootstrap	trainset	83.0	76.0	44.3	40.2
	+ensemble	trainset	86.7	–	49.0	46.9

Model size	Acc (8-shot) (%)
13b	28.7
34b	42.2
70b	56.8

LLaMA2 results (GSM8K)

- DSPy program with LLaMA 13b is competitive with their 34b-based results even though they don't use human reasoning chains in our program.
- Auto-CoT reports 48% for GPT 3 (text-davinci-002), which aligns closely with llama2-13b-chat results.

How many storeys are there in the castle David Gregory inherited?

Case Study: Complex QA

```
1 class BasicMultiHop(dspy.Module):
2     def __init__(self, passages_per_hop):
3         self.retrieve = dspy.Retrieve(k=passages_per_hop)
4         self.generate_query = dspy.ChainOfThought("context, question -> search_query")
5         self.generate_answer = dspy.ChainOfThought("context, question -> answer")
6
7     def forward(self, question):
8         context = []
9
10        for hop in range(2):
11            query = self.generate_query(context=context, question=question).search_query
12            context += self.retrieve(query).passages
13
14        return self.generate_answer(context=context, question=question)
15
16 multihop = BasicMultiHop(passages_per_hop=3)
```

LM
What castle did David Gregory inherit?



Retriever
St. Gregory is a 9-floor boutique hotel...



LM
How many storeys are in Kinnairdy Castle?



Retriever
Kinnairdy Castle is a tower house with five storeys...



LM



Castle Gregory has five storeys. ✓

Why DSPy

- Reliability, predictability, scalability
- Other solutions such as LangChain and LLamaIndex still have manual prompt engineering prevalent in their codebase
- DSPy provides a structured framework that automatically bootstraps prompts.

Key Takeaways

- Don't mess with the weights (prompts) by hand. Define new modules or teleprompters.
- Don't expect "one layer" to do the work. Add inductive biases and "depth".
- Use local models for reproducible research. They work really well when compiled.