

DSPy: Programming, not Prompting Language Models

Shangyin Tan

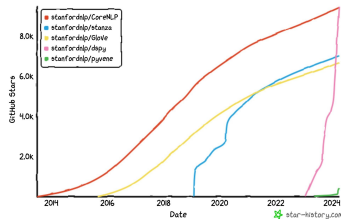


Contributors 164



[+ 150 contributors](#)

Star History



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, a former California governor who donated the land and funds to establish the university. Stanford is located in Stanford, California, near Palo Alto.

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 former California governor who donated the land and funds to establish the university. Stanford is located in Stanford, California, near Palo Alto.

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 former California governor who donated the land and funds to establish the university. Stanford is located in Stanford, California, near Palo Alto.

The **DSPy** paradigm - dspy.ai

Let's program—not prompt—LMs.

Connect declarative modules into a computation graph, and compile it into a chain of optimized prompts (or LM finetunes) automatically. **How?**

1. ~~Hand-written Prompts~~ **Signatures:** *question -> answer long_document -> summary*
2. ~~Prompting Techniques and Chains~~ **Modules:** *dspy.ChainOfThought* *dspy.ReAct*
3. ~~Manual Prompt Engineering~~
 - a. **Optimizers** (given a metric you want to maximize)
 - b. **Assertions** (similar to assertions in programming language)

Let's get concrete: Question Answering with HotPotQA

Question: *How many storeys are in the castle David Gregory inherited?*

Passages: *(1) St. Gregory Hotel is a 9-floor boutique hotel... (2) Kinnairdy Castle is tower house with five storeys...*

Answer: Kinnairdy Castle has five storeys.

Let's build three programs for this task. **Program 1.**

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

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

CoT

*Answer: Castle Gregory has **three** storeys.*



Question



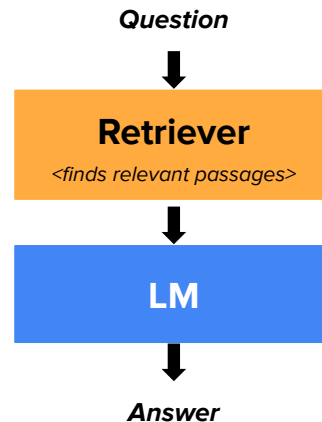
CoT



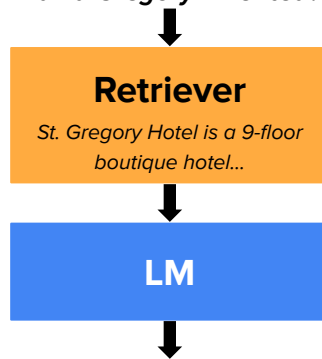
Chain of Thought (Reasoning); Answer

Let's build another program for this task. **Program 2.**

```
class RAG(dspy.Module):  
    def __init__(self, num_passages=3):  
        self.retrieve = dspy.Retrieve(k=num_passages)  
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")  
  
    def forward(self, question):  
        passages = self.retrieve(question).passages  
        return self.generate_answer(context=passages, question=question)
```



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



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



Let's build a solid **Program 3** for this task.

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

How many storeys are in the castle

David Gregory inherited?

LM

What castle did David
Gregory inherit?

Retriever

David Gregory... inherited the
Kinnairdy Castle in 1664.

LM

How many storeys are
in Kinnairdy Castle?

Retriever

Kinnairdy Castle is tower
house with five storeys...

LM

Kinnairdy Castle has five storeys.



What does the **DSPy Compiler** do?

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



Let's compile our multi-hop program.

```
1 Given the fields 'context', 'question', produce the fields 'search.query'.
2
3 ---
4 Follow the following format.
5
6 Context: ${context}
7 Question: ${question}
8 Reasoning: Let's think step by step in order to ${produce the search.query}. We ...
9 Search Query: ${search.query}
10
11 ---
12
13 Context:
14 [1] Twilight (novel series) | Twilight is a series of four vampire-themed fantasy romance novels by American author
15 Stephenie Meyer. ...
16 [2] Harper Connelly Mysteries | The Harper Connelly Mysteries is a series of fantasy mystery novels written by Charlaine
17 Harris, and first published in 2005. ...
18 [3] The Dark Heroine | The Dark Heroine is a series of vampire-themed fantasy romance novels written by English author
19 Abigail Gibbs, published by HarperCollins in 2012. ...
20
21 Question: In which year was the first of the vampire-themed fantasy romance novels for which The Twilight Saga: The
22 Official Illustrated Guide serves as a spin-off encyclopedic reference book first published?
23
24 Reasoning: Let's think step by step in order to determine the year the first of the vampire-themed fantasy romance novels
25 was first published. ...
26
27 Search Query: When was the first of the vampire-themed fantasy romance novels published?
28
29 ---
30
31 Context:
32 [1] The Victorians | The Victorians - Their Story In Pictures is a 2009 British documentary series which focuses on
33 Victorian art and culture. ...
34 [2] The Caxtons | The Caxtons: A Family Picture is an 1849 Victorian novel by Edward Bulwer-Lytton that was popular in its
35 time.
36 [3] Victorian (comics) | The Victorian is a 25-issue comic book series published by Penny-Farthing Press and starting in
37 1999. ...
38
39 Question: The Victorians - Their Story In Pictures is a documentary series written by an author born in what year?
40
41 Reasoning: Let's think step by step in order to produce the search query. We know that the documentary series is about
42 Victorian art and culture, and it was written and presented by Jeremy Paxman. Therefore, we need to find the year in which
43 Jeremy Paxman was born.
44
45 Search Query: Jeremy Paxman birth year
46
47 ---
48
49 Context:
```

```
class MultiHop(dspy.Module):
    def __init__(self, passages_per_hop=3):
        self.generate_query = dspy.ChainOfThought("context, question -> query")
        self.retrieve = dspy.Retrieve(k=passages_per_hop)
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []

        for _ in range(2):
            query = self.generate_query(context, question).query
            context += self.retrieve(query).passages

        return self.generate_answer(context, question)
```

Figure 11: Shortened copy of the prompt automatically generated by DSPy for HotPotQA Llama2-13b-chat multi-hop program (generating second hop query) compiled with bootstrap.

Basket of **optimizers** (and growing!)

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

```
teleprompter = dspy.BootstrapFewShotWithRandomSearch(metric=gsm8k_accuracy)  
bootstrapped_program = teleprompter.compile(program, trainset=trainset)
```

```
bootstrappedx2_program = teleprompter.compile(program, teacher=bootstrapped_program, trainset=trainset)
```

```
ensemble = dspy.Ensemble(reduce_fn=dspy.majority).compile(bootstrapped_program.programs[:7])
```

GSM8K - grade school math dataset

20 birds migrate on a seasonal basis from one lake to another, searching for food. If they fly from lake Jim to lake Disney in one season, which is 50 miles apart, then the next season they fly from lake Disney to lake London, 60 miles apart, calculate the combined distance all of the birds have traveled in the two seasons.

Answer: $20 * (50 + 60) = 2200$ miles

First DSPy program

```
class SimpleMathSolver(dspy.Module):  
    def __init__(self):  
        self.prog = dspy.ChainOfThought("question -> answer")  
  
    def forward(self, question):  
        return self.prog(question=question)  
  
simple_math_solver = SimpleMathSolver()
```

Given the fields `question`, produce the fields `answer`.

Follow the following format.

Question: \${question}

Reasoning: Let's think step by step in order to \${produce the answer}. We ...

Answer: \${answer}

Question: 20 birds migrate on a seasonal basis from one lake to another, searching for food. If they fly from lake Jim to lake Disney in one season, which is 50 miles apart, then the next season they fly from lake Disney to lake London, 60 miles apart, calculate the combined distance all of the birds have traveled in the two seasons.

Reasoning: **Let's think step by step in order to produce the answer. We know that the birds fly 50 miles from lake Jim to lake Disney and then 60 miles from lake Disney to lake London. To find the combined distance, we simply add the two distances together.**

Answer: The combined distance all of the birds have traveled in the two seasons is 50 miles + 60 miles = 110 miles.

Two problems:

1. Reasoning does not include all the information needed
2. Answer in complicated sentences, not a single number

LMs are not following what we expect.

And, **there's no way to even specify the constraints** except manual prompt tuning.

Goal: Enable DSPy programmers to define constraints on LM behavior.

Can we make a robust, extensible programming construct?

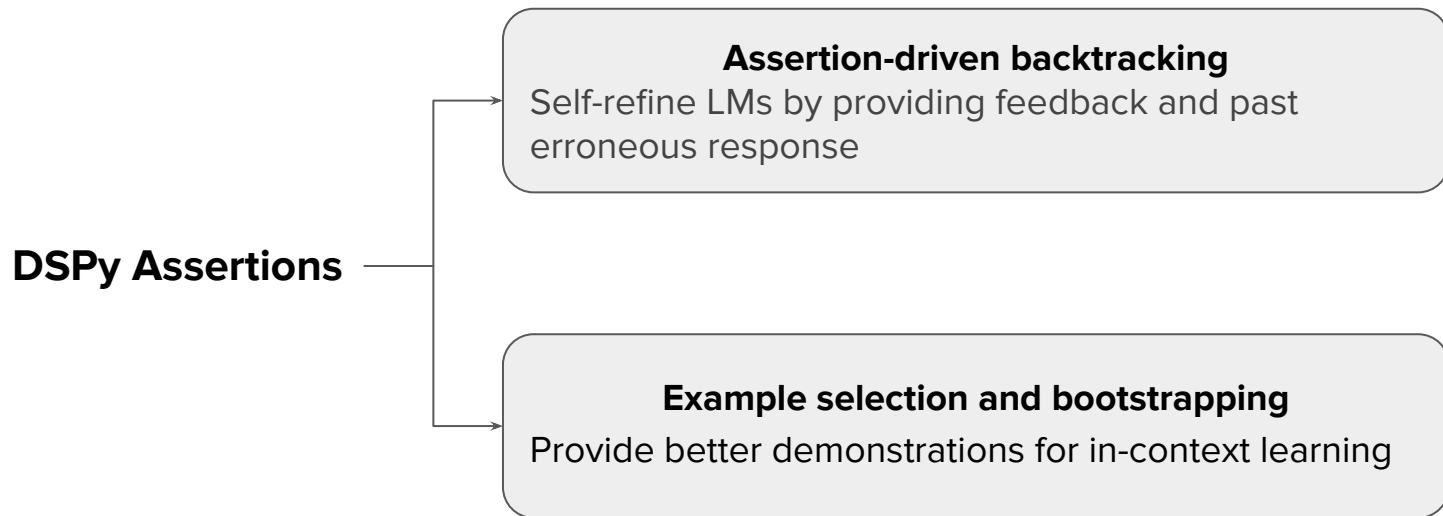
Including Assertions in DSPy

dspy.Assert - DSPy must either pass the assertion or raise an Exception

dspy.Suggest - DSPy should try to pass the assertion, but permit the code to continue otherwise

```
dspy.Suggest(constraint: bool, instruction_message: str)
```

Introducing ***DSPy Assertions*** — to guide LM behavior in DSPy programs.



DSPy program with Assertions

```
def extract_number(question):  
    """extract numbers from a question"""  
  
def has_numbers(rationale, numbers):  
    """whether rationale has all the numbers, if not, return the missing number"""  
  
class SimpleMathSolverWithSuggest(dspy.Module):  
    def __init__(self):  
        self.prog = dspy.ChainOfThought("question -> answer")  
  
    def forward(self, question):  
        pred = self.prog(question)  
        rationale_has_numbers = has_numbers(pred.rationale, numbers)  
        dspy.Suggest(rationale_has_numbers, f"Your Reasoning should contain {missing_number}.")  
        dspy.Suggest(len(pred.answer) < 10, "Your Answer should be a number.")  
        return pred  
  
simple_math_solver_suggest = SimpleMathSolverWithSuggest().activate_assertions()
```

Assertion-driven backtracking

Self-refine LMs by providing feedback and past erroneous response

Given the fields `question`, produce the fields `answer`.

Follow the following format.

Question: \${question}

Reasoning: Let's think step by step in order to \${produce the answer}. We ...

Answer: \${answer}

Question: 20 birds migrate on a seasonal basis from one lake to another, searching for food. If they fly from lake Jim to lake Disney in one season, which is 50 miles apart, then the next season they fly from lake Disney to lake London, 60 miles apart, calculate the combined distance all of the birds have traveled in the two seasons.

Reasoning: **Let's think step by step in order to produce the answer. We know that the birds fly 50 miles from lake Jim to lake Disney and then 60 miles from lake Disney to lake London. To find the combined distance, we simply add the two distances together.**

Answer: The combined distance all of the birds have traveled in the two seasons is 50 miles + 60 miles = 110 miles.

```
def extract_number(question):  
    """extract numbers from a question"""  
  
def has_numbers(rationale, numbers):  
    """whether rationale has all the numbers, if not, return the missing number"""  
  
class SimpleMathSolverWithSuggest(dspy.Module):  
    def __init__(self):  
        self.prog = dspy.ChainOfThought("question -> answer")  
  
    def forward(self, question):  
        pred = self.prog(question=question)  
        rationale_has_numbers, missing_number = has_numbers(pred.rationale, extract_number(question))  
        dspy.Suggest(rationale_has_numbers, f"Your Reasoning should contain {missing_number}.")  
        dspy.Suggest(len(pred.answer) < 10, "Your Answer should be a number.")  
        return pred  
  
simple_math_solver_suggest = SimpleMathSolverWithSuggest().activate_assertions()
```

Given the fields `question`, produce the fields `answer`.

Follow the following format.

Question: $\${question}$

Previous Reasoning: past Reasoning with errors

Instructions: Some instructions you must satisfy

Reasoning: Let's think step by step in order to $\${produce\ the\ answer}$. We ...

Answer: $\${answer}$

Question: 20 birds migrate on a seasonal basis from one lake to another, searching for food. If they fly from lake Jim to lake Disney in one season, which is 50 miles apart, then the next season they fly from lake Disney to lake London, 60 miles apart, calculate the combined distance all of the birds have traveled in the two seasons.

Previous Reasoning: produce the answer. We know that the birds fly 50 miles from lake Jim to lake Disney and then 60 miles from lake Disney to lake London. To find the combined distance, we simply add the two distances together.

Instructions: Your Reasoning should contain 20.

Reasoning: Let's think step by step in order to **calculate the combined distance all of the birds have traveled in the two seasons. In the first season, 20 birds fly 50 miles from lake Jim to lake Disney, which totals to 20 birds * 50 miles = 1000 miles. In the second season, they fly 60 miles from lake Disney to lake London, which totals to 20 birds * 60 miles = 1200 miles. Therefore, the combined distance all of the birds have traveled in the two seasons is 1000 miles + 1200 miles = 2200 miles.**

Answer: The combined distance all of the birds have traveled in the two seasons is 2200 miles.

```
def extract_number(question):
    """extract numbers from a question"""

def has_numbers(rationale, numbers):
    """whether rationale has all the numbers, if not, return the missing number"""

class SimpleMathSolverWithSuggest(dspy.Module):
    def __init__(self):
        self.prog = dspy.ChainOfThought("question -> answer")

    def forward(self, question):
        pred = self.prog(question=question)
        rationale_has_numbers, missing_number = has_numbers(pred.rationale, extract_number(question))
        dspy.Suggest(rationale_has_numbers, f"Your Reasoning should contain {missing_number}.")
        dspy.Suggest(len(pred.answer) < 10, "Your Answer should be a number.")
        return pred

simple_math_solver_suggest = SimpleMathSolverWithSuggest().activate_assertions()
```


Given the fields `question`, produce the fields `answer`.

Follow the following format.

Question: \${question}

Previous Reasoning: past Reasoning with errors

Instructions: Some instructions you must satisfy

Reasoning: Let's think step by step in order to \${produce the answer}. We ...

Answer: \${answer}

Question: 20 birds migrate on a seasonal basis from one lake to another, searching for food. If they fly from lake Jim to lake Disney in one season, which is 50 miles apart, then the next season they fly from lake Disney to lake London, 60 miles apart, calculate the combined distance all of the birds have traveled in the two seasons.

Previous Answer: The combined distance all of the birds have traveled in the two seasons is 2200 miles.

Instructions: Your Answer should be a number.

Reasoning: **Let's think step by step in order to produce the answer. We know that in the first season, 20 birds fly 50 miles from lake Jim to lake Disney, totaling 20 birds * 50 miles = 1000 miles. In the second season, they fly 60 miles from lake Disney to lake London, totaling 20 birds * 60 miles = 1200 miles. Therefore, the combined distance all of the birds have traveled in the two seasons is 1000 miles + 1200 miles = 2200 miles.**

Answer: 2200 miles

Example selection and bootstrapping

Provide better demonstrations for in-context learning

```
optimizer = BootstrapFewShotWithRandomSearch(  
    gsm8k_metric, max_bootstrapped_demos=3, max_labeled_demos=6, num_candidate_programs=6)  
  
compiled_prog = optimizer.compile(student=simple_math_solver)  
  
compiled_prog_suggest = optimizer.compile(student=simple_math_solver_suggest)
```

1. Demonstrations need to pass DSPy assertions, too.
2. Demonstrations can also contain traces with errors and fixes.
3. Optimizer is "assertion-aware" - it also calculates how many assertion failure in addition to "metric"

Results

Simple CoT	CoT w Assertions	Compiled CoT	Compiled w Assertion
61.7	74.3	83.7	84.0

*Compiled w Assertion does not outperform Compiled on this task. Our observations are: the harder assertions are, the better they are at improving in-context learning demonstrations.


HumanEval - code generation

How could we use LM Assertions here?

First DSPy program

```
class NaiveCodeGenerator(dspy.Module):  
    def __init__(self):  
        self.prog = dspy.ChainOfThought("prompt -> code")  
  
    def forward(self, prompt):  
        pred = self.prog(prompt=prompt)  
        return pred
```

Prompt

 Code
Generator

Code

Code

Code Generation Pipeline


UPDATED PROMPT WITH FEEDBACK

Prompt: . . .


Past Code: <previous attempt w/ errors> . . .

Instruction: The generated code failed the test
<test>, please fix it. . .

Prompt

 Code
Generator

Code

 Test
Generator

Test

```
suggest(check_tests(code, test), f"The  
generated code failed the test {test}")
```

Code

Code Generation Pipeline w. Suggestions

DSPy program with Assertions

```
class NaiveCodeGenerator(dspy.Module):  
    def __init__(self):  
        self.prog = dspy.ChainOfThought("prompt -> code")  
        self.generate_test = dspy.ChainOfThought("prompt -> test")  
  
    def forward(self, prompt):  
        pred = self.prog(prompt=prompt)  
        tests = self.generate_test(prompt=prompt)  
        result, test, error = check_tests(pred, tests)  
        dspy.Suggest( result == "passed",  
            f"The generated code failed the test {test}, please fix {error}.",  
            backtrack_module = self.prog )  
        return pred
```

Results

Naive Program Generator	Program Generator w Assertions
70.7	75.6

*Issues with self-consistency:

1. Both tests and code went wrong in the same way
2. Generated tests are wrong, thus not giving useful feedback

Notebook: bit.ly/dspy-gsm8k
bit.ly/dspy-humaneval
bit.ly/dspy-intro

Twitter: @Shangyint

Multi-Hop Question Answering with HotPotQA with Suggestions

```
class MultiHop(dspy.Module):  
    def __init__(self, passages_per_hop=3):  
        self.generate_query = dspy.ChainOfThought("context, question -> query")  
        self.retrieve = dspy.Retrieve(k=passages_per_hop)  
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")  
  
    def forward(self, question):  
        context = []  
        queries = [question]  
        for _ in range(2):  
            query = self.generate_query(context=context, question=question).query  
            dspy.Suggest(len(query) < 100,  
                        "Query should be less than 100 characters")  
  
            dspy.Suggest(is_query_distinct(query, queries),  
                        f"Query should be distinct from {queries}")  
  
            queries += query  
            context += self.retrieve(query).passages  
        return self.generate_answer(context=context, question=question)
```

The query should be
concise

The queries should
be different from
previous ones

Multi-Hop Question Answering with HotPotQA with Suggestions

```
class MultiHop(dspy.Module):  
    def __init__(self, passages_per_hop=3):  
        self.generate_query = dspy.ChainOfThought("context, question -> query")  
        self.retrieve = dspy.Retrieve(k=passages_per_hop)  
        self.generate_answer = dspy.ChainOfThought("context, question -> answer")  
  
    def forward(self, question):  
        context = []  
        queries = [question]  
        for _ in range(2):  
            query = self.generate_query(context=context, question=question).query  
            dspy.Suggest(len(query) < 100,  
                        "Query should be less than 100 characters")  
            dspy.Suggest(is_query_distinct(query, queries),  
                        f"Query should be distinct from {queries}")  
  
            queries += query  
            context += self.retrieve(query).passages  
        return self.generate_answer(context=context, question=question)
```

Fail ✗

backtrack and regenerate **query** with new prompt

update prompt with feedback

Context: ...
Question: ...
Past_Query: {previous attempt w/ errors}
Instructions: Query should be distinct from ...

UPDATED PROMPT WITH FEEDBACK

Context: . . .

Question: . . .

Past Query: <previous attempt w/ errors> . . .

Instruction: Query should be distinct from . . .

{
✓ `suggest(len(query) < 100, "Query should be less than 100 chars")`
✗ `suggest(is_query_distinct(query, prev_queries), f"Query should be distinct from {prev_queries}")`
}

Question

 Query Generator

Query

Retriever

Context

 Answer Generator

Answer

Context + Previous Queries

Multi-hop QA Pipeline