

prog

the 1990s, the number of people in the United States who are 65 years of age or older has increased by 50 percent. The number of people 75 years of age or older has increased by 100 percent. The number of people 85 years of age or older has increased by 200 percent. The number of people 95 years of age or older has increased by 400 percent. The number of people 100 years of age or older has increased by 800 percent. The number of people 105 years of age or older has increased by 1,600 percent. The number of people 110 years of age or older has increased by 3,200 percent. The number of people 115 years of age or older has increased by 6,400 percent. The number of people 120 years of age or older has increased by 12,800 percent. The number of people 125 years of age or older has increased by 25,600 percent. The number of people 130 years of age or older has increased by 51,200 percent. The number of people 135 years of age or older has increased by 102,400 percent. The number of people 140 years of age or older has increased by 204,800 percent. The number of people 145 years of age or older has increased by 409,600 percent. The number of people 150 years of age or older has increased by 819,200 percent. The number of people 155 years of age or older has increased by 1,638,400 percent. The number of people 160 years of age or older has increased by 3,276,800 percent. The number of people 165 years of age or older has increased by 6,553,600 percent. The number of people 170 years of age or older has increased by 13,107,200 percent. The number of people 175 years of age or older has increased by 26,214,400 percent. The number of people 180 years of age or older has increased by 52,428,800 percent. The number of people 185 years of age or older has increased by 104,857,600 percent. The number of people 190 years of age or older has increased by 209,715,200 percent. The number of people 195 years of age or older has increased by 419,430,400 percent. The number of people 200 years of age or older has increased by 838,860,800 percent. The number of people 205 years of age or older has increased by 1,677,721,600 percent. The number of people 210 years of age or older has increased by 3,355,443,200 percent. The number of people 215 years of age or older has increased by 6,710,886,400 percent. The number of people 220 years of age or older has increased by 13,421,772,800 percent. The number of people 225 years of age or older has increased by 26,843,545,600 percent. The number of people 230 years of age or older has increased by 53,687,091,200 percent. The number of people 235 years of age or older has increased by 107,374,182,400 percent. The number of people 240 years of age or older has increased by 214,748,364,800 percent. The number of people 245 years of age or older has increased by 429,496,729,600 percent. The number of people 250 years of age or older has increased by 858,993,459,200 percent. The number of people 255 years of age or older has increased by 1,717,986,918,400 percent. The number of people 260 years of age or older has increased by 3,435,973,836,800 percent. The number of people 265 years of age or older has increased by 6,871,947,673,600 percent. The number of people 270 years of age or older has increased by 13,743,895,347,200 percent. The number of people 275 years of age or older has increased by 27,487,790,694,400 percent. The number of people 280 years of age or older has increased by 54,975,581,388,800 percent. The number of people 285 years of age or older has increased by 109,951,162,777,600 percent. The number of people 290 years of age or older has increased by 219,902,325,555,200 percent. The number of people 295 years of age or older has increased by 439,804,651,110,400 percent. The number of people 300 years of age or older has increased by 879,609,302,220,800 percent. The number of people 305 years of age or older has increased by 1,759,218,604,441,600 percent. The number of people 310 years of age or older has increased by 3,518,437,208,883,200 percent. The number of people 315 years of age or older has increased by 7,036,874,417,766,400 percent. The number of people 320 years of age or older has increased by 14,073,748,835,532,800 percent. The number of people 325 years of age or older has increased by 28,147,497,671,065,600 percent. The number of people 330 years of age or older has increased by 56,294,995,342,131,200 percent. The number of people 335 years of age or older has increased by 112,589,990,684,262,400 percent. The number of people 340 years of age or older has increased by 225,179,981,368,524,800 percent. The number of people 345 years of age or older has increased by 450,359,962,737,049,600 percent. The number of people 350 years of age or older has increased by 900,719,925,474,099,200 percent. The number of people 355 years of age or older has increased by 1,801,439,850,948,198,400 percent. The number of people 360 years of age or older has increased by 3,602,879,701,896,396,800 percent. The number of people 365 years of age or older has increased by 7,205,759,403,792,793,600 percent. The number of people 370 years of age or older has increased by 14,411,518,807,585,587,200 percent. The number of people 375 years of age or older has increased by 28,823,037,615,171,174,400 percent. The number of people 380 years of age or older has increased by 57,646,075,230,342,348,800 percent. The number of people 385 years of age or older has increased by 115,292,150,460,684,697,600 percent. The number of people 390 years of age or older has increased by 230,584,300,921,369,395,200 percent. The number of people 395 years of age or older has increased by 461,168,601,842,738,790,400 percent. The number of people 400 years of age or older has increased by 922,337,203,685,477,580,800 percent. The number of people 405 years of age or older has increased by 1,844,674,407,370,955,161,600 percent. The number of people 410 years of age or older has increased by 3,689,348,814,741,910,323,200 percent. The number of people 415 years of age or older has increased by 7,378,697,629,483,820,646,400 percent. The number of people 420 years of age or older has increased by 14,757,395,258,967,641,292,800 percent. The number of people 425 years of age or older has increased by 29,514,790,517,935,282,585,600 percent. The number of people 430 years of age or older has increased by 59,029,581,035,870,565,171,200 percent. The number of people 435 years of age or older has increased by 118,059,162,071,741,130,342,400 percent. The number of people 440 years of age or older has increased by 236,118,324,143,482,260,684,800 percent. The number of people 445 years of age or older has increased by 472,236,648,286,964,521,369,600 percent. The number of people 450 years of age or older has increased by 944,473,296,573,929,042,739,200 percent. The number of people 455 years of age or older has increased by 1,888,946,593,147,858,085,478,400 percent. The number of people 460 years of age or older has increased by 3,777,893,186,295,716,170,956,800 percent. The number of people 465 years of age or older has increased by 7,555,786,372,591,432,341,913,600 percent. The number of people 470 years of age or older has increased by 15,111,572,745,182,864,683,827,200 percent. The number of people 475 years of age or older has increased by 30,223,145,490,365,729,367,654,400 percent. The number of people 480 years of age or older has increased by 60,446,290,980,731,458,735,308,800 percent. The number of people 485 years of age or older has increased by 120,892,581,961,462,917,470,617,600 percent. The number of people 490 years of age or older has increased by 241,785,163,922,925,834,941,235,200 percent. The number of people 495 years of age or older has increased by 483,570,327,845,851,669,882,470,400 percent. The number of people 500 years of age or older has increased by 967,140,655,691,703,339,764,940,800 percent. The number of people 505 years of age or older has increased by 1,934,281,311,383,406,679,529,881,600 percent. The number of people 510 years of age or older has increased by 3,868,562,622,766,813,359,059,763,200 percent. The number of people 515 years of age or older has increased by 7,737,125,245,533,626,718,119,526,400 percent. The number of people 520 years of age or older has increased by 15,474,250,491,067,253,436,239,052,800 percent. The number of people 525 years of age or older has increased by 30,948,500,982,134,506,872,478,105,600 percent. The number of people 530 years of age or older has increased by 61,897,001,964,269,013,744,956,211,200 percent. The number of people 535 years of age or older has increased by 123,794,003,928,538,027,489,912,422,400 percent. The number of people 540 years of age or older has increased by 247,588,007,857,076,054,979,824,844,800 percent. The number of people 545 years of age or older has increased by 495,176,015,714,152,109,959,649,689,600 percent. The number of people 550 years of age or older has increased by 990,352,031,428,304,219,919,299,379,200 percent. The number of people 555 years of age or older has increased by 1,980,704,062,856,608,439,838,598,758,400 percent. The number of people 560 years of age or older has increased by 3,961,408,125,713,216,879,677,197,516,800 percent. The number of people 565 years of age or older has increased by 7,922,816,251,426,433,759,354,395,033,600 percent. The number of people 570 years of age or older has increased by 15,845,632,502,852,867,518,708,790,067,200 percent. The number of people 575



Follow along with the notebook provided for this tutorial

https://github.com/AdoHaha/dspy_fun/



Large Language Models are super capable and available

- Text: free form text, code, classification
- Images: image understanding, OCR, ...

Ever nicer features:

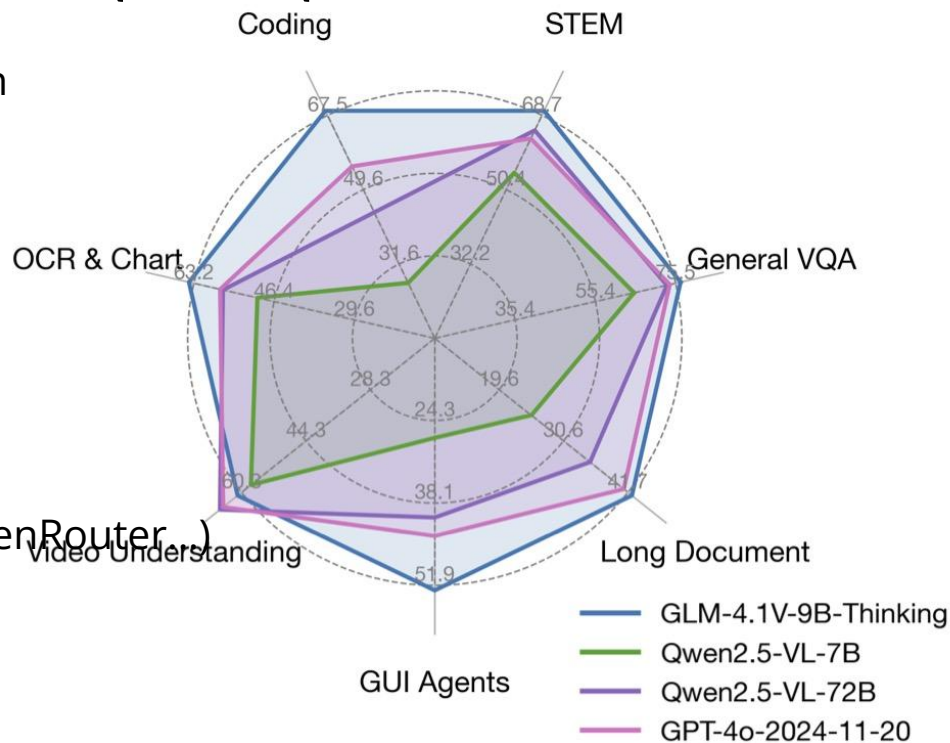
- becoming smart/ specialized
- more and more are open source

Available:

- run locally
- choose from many cloud provides (OpenRouter, ...)

Cheap!

- single image call in range of 0.001 EUR



How to get all those goodies? OPEN AI API

```
from openai import OpenAI

client = OpenAI(
    api_key="GEMINI_API_KEY",base_url="https://generativelanguage.googleapis.com/v1beta/openai/")

completion = client.chat.completions.create(

    model="gemini-2.5-flash",

    messages=[

        {"role": "system", "content": "You are a helpful assistant."},

        {"role": "user", "content": "what is 2+2"}

    ])

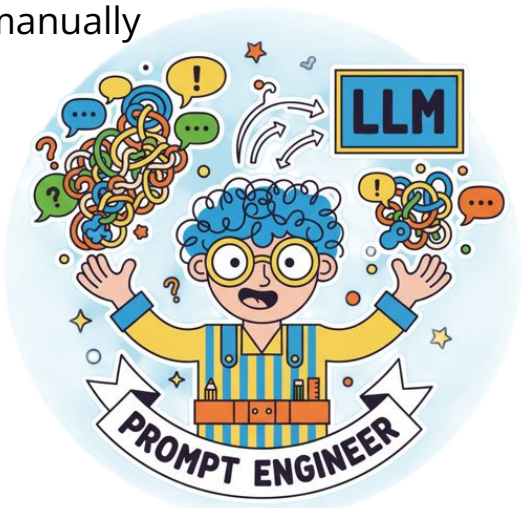
print(completion.choices[0].message)
```

Structured output

```
response_mime_type="application/json",  
    response_schema=genai.types.Schema(  
        type = genai.types.Type.OBJECT,  
        required = ["sum_of_numbers"],  
        properties = {  
            "sum_of_numbers": genai.types.Schema(  
                type = genai.types.Type.NUMBER,  
            ), }, )
```

Prompts

- how to ensure that you get what you want
 - correct answer
 - correct format
- necessarily context
- difficult, time consuming job if doing this manually



How to prompt engineer

The prompt engineering pages in this section have been organized from most broadly effective techniques to more specialized techniques. When troubleshooting performance, we suggest you try these techniques in order, although the actual impact of each technique will depend on your use case.
















1. [Prompt generator](#)
2. [Be clear and direct](#)
3. [Use examples \(multishot\)](#)
4. [Let Claude think \(chain of thought\)](#)
5. [Use XML tags](#)
6. [Give Claude a role \(system prompts\)](#)
7. [Prefill Claude's response](#)
8. [Chain complex prompts](#)
9. [Long context tips](#)

We want models to do particular tasks

- > maybe we want to give required knowledge?
- > maybe we want to limit the output or re-verify?
- > obviously we want to connect these models to various inputs, outputs

For that we need to use more complex interactions

There are strategies to use models to get what we want frequently requiring more than a single tool call. **Bonus complexity if we want multimodality**

 Zero-shot Prompting	 Few-shot Prompting	 Chain-of-Thought Prompting
 Meta Prompting	 Self-Consistency	 Generate Knowledge Prompting
 Prompt Chaining	 Tree of Thoughts	 Retrieval Augmented Generation
 Automatic Reasoning and Tool-use	 Automatic Prompt Engineer	 Active-Prompt
 Directional Stimulus Prompting	 Program-Aided Language Models	 ReAct

In the ideal world, we would focus on:

- focus on information flow
- decouple from underlying technology, modularize
- make it understandable, modifiable
- align the models to the system

So essentially we want to **program these things**

Ideally program in **higher level language**, i.e. be able to say what we want

Let's use DSPy

```
import dspy
```

```
small_model = dspy.LM("gemini/gemini-2.5-flash-lite", api_key=GEMINI_API_KEY)
```

```
dspy.configure(lm=small_model)
```

```
sum_of_numbers = dspy.Predict('numbers -> sum_of_numbers')
```

```
result = sum_of_numbers(numbers = (12,13,15))
```

```
print(result)
```

```
Prediction(
```

```
    sum_of_numbers='40'
```

```
)
```



but we did not specify what numbers are, right?

```
numbers_image = dspy.Image.from_file("./numbers.png")
```

```
result_image = sum_of_numbers(numbers = numbers_image)
```

```
print(result_image)
```

```
Prediction(
```

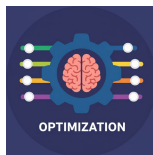
```
    sum_of_numbers='33'
```

```
)
```



DSPy - Declarative Self-improving Python

- Set of components to build an AI system
- Define expected behavior from fuzzy, natural language-defined pieces
- (Usually) invisible tools to adapt to a particular model, good defaults
- Specify information flow/ strategies (what happens to output / how to proceed)
- Improving the model behaviour automatically



signature

- we define what should be the input, output, provide instructions and descriptions
- supports typing

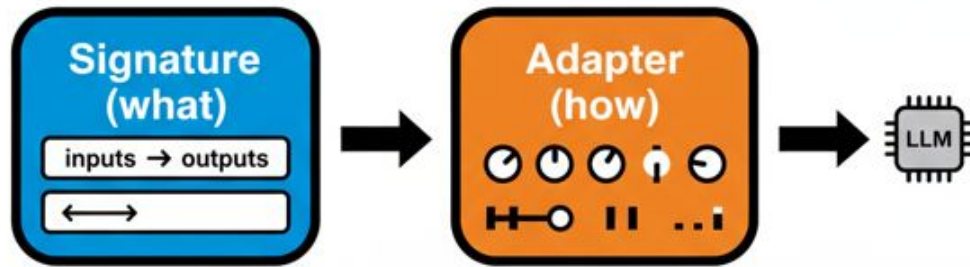
```
from typing import Optional
class NumberAdd(dspy.Signature):
    """Please add numbers provided in a various ways together. Numbers can also be symbolic or require computation.
    Only if there are no numbers in input, write a sad haiku using the contents of input"""
    numbers = dspy.InputField(description="numbers to add")
    sum_of_numbers: float = dspy.OutputField(description="resulting sum")

sum_of_numbers(numbers = ["one, 2"])
Prediction(
    sum_of_numbers=3,
    haiku='Empty bowl waits now,\nNo kibble, just silent air,\nSadness fills the room.'
)
```

Adapter

- Allows using different models (json enabled or not), better or worse at formatting
- Formats input, suggests output and formats the outcoming data (with fallbacks)

JsonAdapters, XML Adapters, chat adapters, two step adapters ...



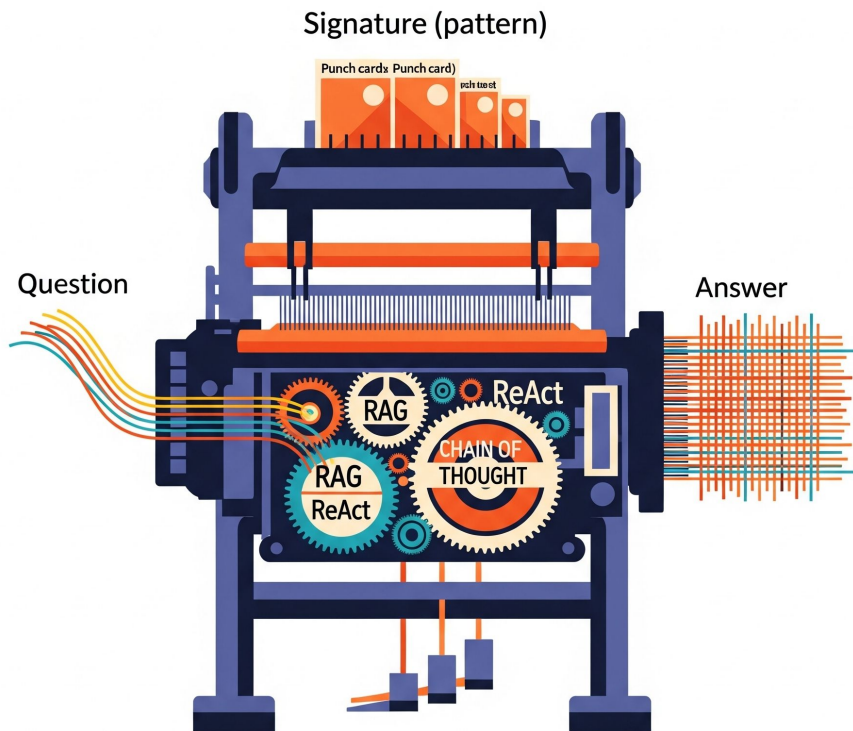
Adapters: translators for LLMs

```
sum_of_numbers =  
dspy.Predict('numbers  
-> sum_of_numbers')
```

```
result_image =  
sum_of_numbers(numbers =  
numbers_image)
```

```
sum_of_numbers.history[-1]  
{  
  "prompt": null,  
  "messages": [{  
    "role": "system",  
    "content": "Your input fields are:\n1. `numbers` (str)\nYour output fields are:\n1. `sum_of_numbers` (str)\n\n[[ ## numbers ## ]]\n{numbers}\n\n[[ ##  
sum_of_numbers ## ]]\n{sum_of_numbers}\n\n[[ ## completed ## ]]\n\nObjective: Given `numbers`, produce `sum_of_numbers`."  
  }, {  
    "role": "user",  
    "content": [  
      {"type": "text", "text": "[[ ## numbers ## ]]\n"},  
      {"type": "image_url", "image_url": {"url": "https://raw.githubusercontent.com/AdoHaha/dspy_fun/refs/heads/main/example_files/image_numbers.png"}},  
      {"type": "text", "text": "\n\nRespond with the output fields, starting with `[[ ## sum_of_numbers ## ]]', then end with `[[ ## completed ## ]]'."}  
    ]  
  }  
}]}
```

Modules: use a strategy to be able to realise the signature



Patterns of use are available, advanced multistep prompting

Predict: take defined input, get defined output

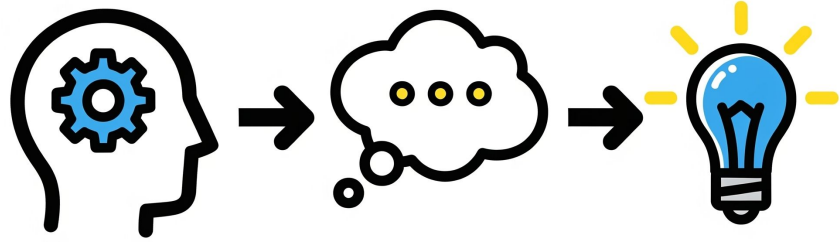
Chain of Thought: improve output through thinking

ReAct: use a set of tools

Retreive (RAG): search for information based on prompt to build context (RAG)

...

Chain of Thought



```
sum_of_numbers_haiku = dspy.ChainOfThought(NumberAdd)
```

```
sum_of_numbers_haiku(numbers = "dragon,siete, enterprise")
```

```
Prediction(
```

```
    reasoning='The input contains words that do not represent numerical values. "dragon",  
    "siete" (Spanish for seven), and "enterprise" are not directly convertible to numbers.
```

```
    Therefore, no numerical sum can be calculated.',
```

```
    sum_of_numbers=0.0,
```

```
    haiku='Words float like lost dreams,\n    Numbers hide, a silent plea,\n    Sum remains unseen.'
```

```
)
```

Still not super smart ;-(lets use a more costly model:

```
with dspy.context(lm = dspy.LM("gemini/gemini-2.5-flash",api_key=GEMINI_API_KEY)):  
    sum_of_numbers_haiku = dspy.ChainOfThought(NumberAdd)  
    print(sum_of_numbers_haiku(numbers = "dragon,siete, enterprise"))
```

```
Prediction(  
    reasoning='I identified "siete" as the Spanish word for seven. "Dragon" and "enterprise" were not recognized as numerical values or symbolic  
representations of numbers. Therefore, the only number to sum is 7.',  
    sum_of_numbers=7.0,  
    haiku=None  
)  
big_cost = larger_lm.history[1]["cost"]  
small_cost = small_model.history[1]["cost"]  
  
print(f"Big cost: {big_cost}")  
print(f"Small cost: {small_cost}")  
print(f"Smaller model is {big_cost/small_cost} times cheaper")  
Big cost: 0.0007455000000000001  
Small cost: 7.19e-05  
Smaller model is 10.368567454798333 times cheaper
```

Own modules

- create sophisticated interactions with users, tools
- usually composed of submodules
- form a foundations for expressing information flow for context building
- many goodies build-in: ease of logging, asynchronous operations

```
class BestNumber (dspy.Module):
    """module returns sum of numbers through generating multiple answers, analyzing
    them and
    finally verifying the best answer"""
    def __init__(self, n):
        self.n = n
        # will generate n answers
        self.chain = dspy.ChainOfThought(NumberAdd, n=n)
        signature_possible = NumberAdd.append( "possible_answers" ,
        dspy.InputField(
            desc= "choice of possible answers, with reasoning" ,
        ))
        best_answer = dspy.ChainOfThought(signature_possible)

        self.check_rule = dspy.Refine(best_answer, N= 3, reward_fn=self.check_result,
        threshold=1.0)

    def check_result (self, args, result):
        """when number is not zero, haiku should not be generated"""
        rule_exclusive_or = (result.sum_of_numbers != 0) ^ (result.haiku is not None)
        return rule_exclusive_or

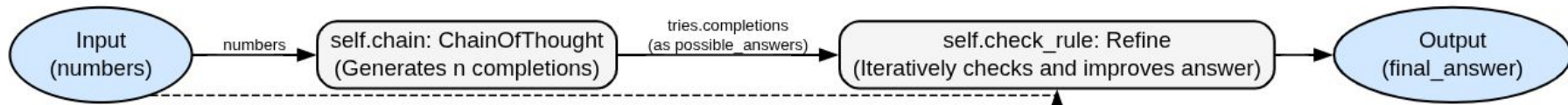
    def forward(self, numbers):
        tries = self.chain(numbers=numbers)

        final_answer = self.check_rule(possible_answers = tries.completions,
        numbers=numbers)

        return final_answer

bestnumber = BestNumber(n= 4)

print (bestnumber(numbers = "dragon,siete,enterprise" ))
```



ReAct: Reasoning and Acting

- Plans steps and uses tools

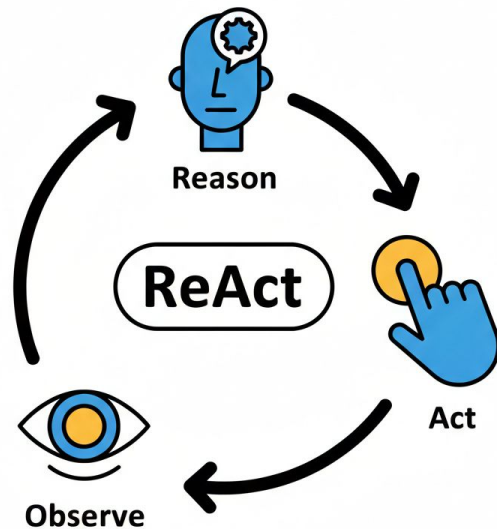
```
import sympy
```

```
def symbolic_expression_sympy(expression: str,  
*Args) -> float:
```

```
    """  
    Takes a symbolic math expression (written as a  
    string) and returns a result as a float,  
    evaluated to 5 significant numbers, using  
    sympy.
```

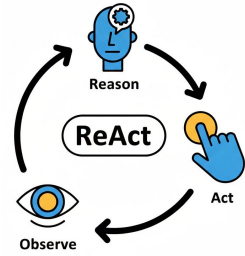
```
    For example  
    symbolic_expression_sympy("2*log(E)") would  
    result in 2.0000
```

```
    """  
    expr = sympy.simplify(expression)  
    return expr.evalf(5)
```



```
sum_of_numbers_smarter =  
dspy.ReAct('numbers ->  
sum_of_numbers',  
tools=[symbolic_expression_sympy  
])
```

ReAct: Reasoning and Acting



```
sum_of_numbers_smarter(numbers=[ "2*sin(10)" , "pi" ])
```

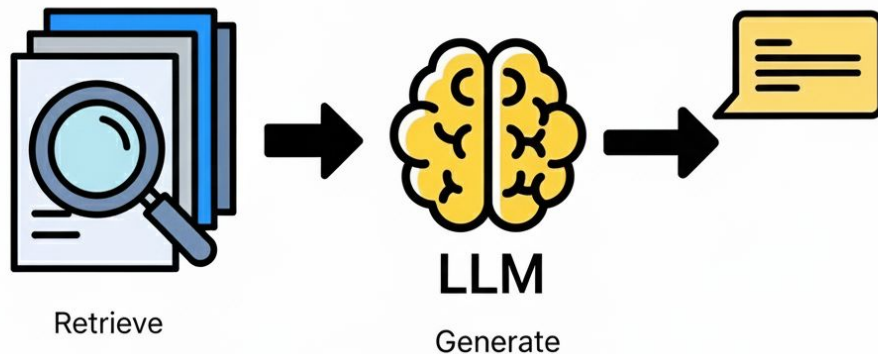
```
Prediction(
```

```
  trajectory={'thought_0': 'The user wants to calculate the sum of the given numbers. The numbers are provided as strings, some of which are mathematical expressions. I need to evaluate these expressions first and then sum them up. The `symbolic_expression_sympy` tool can be used to evaluate the expressions. I will start by evaluating the first expression "2*sin(10)".', 'tool_name_0': 'symbolic_expression_sympy', 'tool_args_0': {'expression': '2*sin(10)'}, 'observation_0': -1.0880, ... 'thought_3': 'The sum of the numbers has been calculated. I can now finish the task.', 'tool_name_3': 'finish', 'tool_args_3': {}, 'observation_3': 'Completed.'},
```

```
  reasoning='The user wants to sum the numbers provided in the `numbers` field.\n\nThe numbers are:\n1. "2*sin(10)"\n2. "pi"\n\nI will use the `symbolic_expression_sympy` tool to evaluate each expression and then sum the results.\n\n\nStep 1: Evaluate "2*sin(10)".\n\n`s symbolic_expression_sympy("2*sin(10)")` returns -1.0880.\n\n\nStep 2: Evaluate "pi".\n\n`s symbolic_expression_sympy("pi")` returns 3.1416.\n\n\nStep 3: Sum the results from Step 1 and Step 2.\n\n`s symbolic_expression_sympy("-1.0880 + 3.1416")` returns 2.0536.\n\n\nTherefore, the sum of the numbers is 2.0536.'
```

```
    sum_of_numbers='2.0536' )
```

Retreive strategy: RAG



Retreival-augmented generation is a common strategy to deliver context.

Retretival can use tools such as reverse index search (akin to old school search engines with keywords) or embedding based search.

DSPy can be easily connected to external vector databases (like chromadb), it plays nicely also with helper tools such as the ones in langchain (for reading standard fileformats, connecting to databases)

The build-in `dspy.Embeddings` tool uses FAISS internally

Retreive: Building context

```
!git clone http://github.com/python/peps.git
```

```
import os
```

```
documents = []
```

```
for filename in os.listdir('./peps/peps'):
```

```
    if filename.endswith('.rst'):
```

```
        filepath = os.path.join('./peps/peps',  
filename)
```

```
        with open(filepath, 'r',  
encoding='utf-8', errors='ignore') as f:
```

```
            documents.append(f.read())
```

```
print(f"Loaded {len(documents)} documents.")
```

```
#Making inverse index
```

```
from inverted_index.inverted_index import  
InvertedIndex
```

```
ii = InvertedIndex()
```

```
ii.index(documents)
```

OR

Making mini embeddings database

```
embedder =
```

```
dspy.Embedder("gemini/embedding-001",
```

```
dimensions = 768, api_key = GEMINI_API_KEY,  
batch_size = 20)
```

```
embeddings_peps = dspy.Embeddings(embedder =  
embedder, corpus = documents, k = 5)
```

Reverse Index based RAG

```
class PEPSearch(dspy.Module):
    def __init__(self):
        self.respond =
dspy.ChainOfThought('context_based_on_search,
python_question ->
easy_to_understand_response_based_on_PEP')
        self.reverseindexquery =
dspy.ChainOfThought('python_question ->
query_to_reverse_index')
    def forward(self, question):
        query = self.reverseindexquery(python_question =
question)
        #print(query)
        search_responses
        = "\n\n".join(ii.search(query.query_to_reverse_index) 0:
10])
        response = self.respond(context_based_on_search =
search_responses, python_question = question)
        return
response.easy_to_understand_response_based_on_PEP
```

```
pep_trivia = PEPSearch()
```

```
pep_trivia(question = "what is PEP 761 about?")
```

PEP 761 is about **deprecating and eventually removing PGP signatures for CPython artifacts ...

Embedding search based RAG

```
class PEPEmbeddingRetreival(dspy.Module):
    def __init__(self):
        self.respond =
dspy.ChainOfThought('context_based_on_search,
python_question ->
easy_to_understand_response_based_on_PEP')
        self.reverseindexquery =
dspy.ChainOfThought('python_question ->
query_to_embbeding_based_search')
    def forward(self, question):
        query = self.reverseindexquery(python_question =
question)
        #print(query)
        search_responses
=embeddings_peps(query.query_to_embbeding_based_search)
        response = self.respond(context_based_on_search =
search_responses, python_question = question)
        return
response.easy_to_understand_response_based_on_PEP
```

```
pep_trivia_embeddings =
PEPEmbeddingRetreival()
pep_trivia_embeddings(question = "any
news about python 3.14?")
```

Yes, there is information about the Python 3.14 release schedule.

The development for Python 3.14 began on Wednesday, May 8, 2024. The release schedule is based on a 17-month development period, leading to a 12-month release cadence between feature versions, as defined by PEP 602.

Here are some key expected dates:

* **3.14.0 alpha 1:** Tuesday, October 15, 2024

Logging and analyzing traces


```
import opik
from opik.integrations.dspy.callback import OpikCallback

opik.configure(use_local=False)
small_model =
dspy.LM("gemini/gemini-2.5-flash-lite",
api_key=GEMINI_API_KEY, max_tokens=20000,
cache=False)
opik_callback =
OpikCallback(project_name="inverse_index_search")
dspy.configure(lm=small_model, callbacks =
[opik_callback])
```

Trace 7 items ①

✓  PEPSearch

⌚ 6.1s

✓  ChainOfThought


⌚ 2.8s

✓  Predict

⌚ 2.7s

 LM: gemini - gemini-2.5-flash-lite

⌚ 1.7s ⚙️ gemini gemini-2.5-flash-lite

 LM: gemini - gemini-2.5-flash-lite


⌚ 1s ⚙️ gemini gemini-2.5-flash-lite

✓  ChainOfThought

⌚ 3s

✓  Predict

⌚ 2.9s

 LM: gemini - gemini-2.5-flash-lite

⌚ 2.9s ⚙️ gemini gemini-2.5-flash-lite

LM: gemini - gemini-2.5-flash-lite

1s gemini gemini-2.5-flash-lite



Input/Output

Feedback scores

Metadata

Input

Pretty ✨

```
[[ ## python_question ## ]]  
what is PEP 761 about?
```

Respond with a JSON object in the following order of fields: `'reasoning'`, then `'query_to_reverse_index'`.

Output

Pretty ✨

```
{  
  "reasoning": "The user is asking for information about a specific Python Enhancement Proposal (PEP). To answer documentation or discussions related to it.",  
  "query_to_reverse_index": "PEP 761 python"  
}
```

DSPY optimization

DSPy key idea is that we first create the information flow system from components, focus on the context engineering while later align the models behaviours through compilation

For that we need to have in place:

- some datasets with expected outcomes
- a metric (or metrics) that we will use to optimize. This metric can be a judge model with set of instructions (we can optimize even the judge)
- choice of optimizers (teleprompters). Those vary on requirements (number of examples, type of metrics, helper functions) and scope of optimization: they can provide important examples (demos), optimize prompts or collaborate when finetuneing the model itself

Datasets & Metrics

```
sum_of_numbers_smarter = dsp.ReAct(  
    dsp.Signature('numbers ->  
sum_of_numbers:float',  
        "Take only numbers (expressed in  
various ways) from numbers and output their sum"),  
    tools=[symbolic_expression_sympy],  
)
```

Datasets are built from Examples

```
sum_example = dsp.Example(numbers = [ 1,2,3],  
sum_of_numbers = 6)  
  
all_examples = [...]  
trainset,devset,testset = all_examples[ ...
```

Metrics – can be any function that returns a float

```
def metric(example, pred, trace=None):  
    gold = example.sum_of_numbers  
    pred = pred.sum_of_numbers  
    return abs(gold - pred)<0.0001 #lets give  
some margin of error
```

```
evaluate = dsp.Evaluate(devset=devset, metric=metric,  
num_threads=4, display_progress=True,  
display_table=0,  
max_errors=999)  
  
evaluate(sum_of_numbers_smarter)  
Average Metric: 13.00 / 15 (86.7%): 100%  
15/15 [00:34<00:00, 2.33s/it]2025/08/28 05:37:19 INFO  
dsp.evaluate.evaluate: Average Metric: 13 / 15 (86.7%)  
  
EvaluationResult(score=86.67, results=<list of 15  
results>)
```

DSPy optimizers ZOO

- selecting / finding or creating examples to pass in the prompt (FewShot approach)
- using best examples for a given prompt (**KNNFewShot**)
- optimizing instructions for each step (COPRO, MIPROv2 ...)
- **passing the trajectory to a separate language model that proposes prompts, using evolutionary process to select best to refine further (GEPA) can be also used with VLMs (to optimize prompt) New in DSPy 3!**
- optimizing prompts and finetuneing models (BootstrapFinetune)

Optimization

Optimizer requirements vary (number of examples, additional functions, models)

but in general the step is that we put the module into optimizer, a trainset, (valset)

than program optimizes / compiles to figure out best use of the pipeline

```
simba = dsp.SIMBA(metric=metric, max_steps=12,  
max_demos=10)  
optimized_agent =  
simba.compile(sum_of_numbers_smarter,  
trainset=trainset, seed=6793115)  
simba.save("optimized_simba.json")
```

The compiled model can be saved (with or without data)

It has added fields "demos" to the underlying modules

Bonus: GEPA

- Approach where a strong model is used to reflect on the answers and guide the optimization process

- More spots to add feedback

```
def metric_with_feedback(example, prediction, trace=None,
    pred_name=None, pred_trace = None):
    correct_answer = float(example.sum_of_numbers)
    try:
        llm_answer = float(prediction.sum_of_numbers)
    except:
        llm_answer = "it was not a number"
    score = float(metric(example, prediction))
    feedback_text = ""
    if score==1:
        feedback_text = f"Your answer is correct
{correct_answer} "
    else:
        print(example)
        feedback_text = f"Your answer: {llm_answer} is not
correct, it should be {correct_answer} "
        print(feedback_text)
    return dspy.Prediction(score = score, feedback =
feedback_text)
```

```
gepa_optimizer = optimizer = dspy.GEPA(
    metric=metric_with_feedback,
    auto="light",
    num_threads=32,
    track_stats=True,
    reflection_minibatch_size=3,
```

```
reflection_lm=dspy.LM(model="gemini/gemini-2.5-
flash", temperature=1.0, max_tokens=32000,
api_key=GEMINI_API_KEY)
)
gepa_optimized_program = optimizer.compile(
    sum_of_numbers_smarter,
    trainset=trainset,
    valset=devset,
)
```


Prompt created automatically

2025/08/28 06:16:25 INFO dspy.teleprompt.gepa.gepa: Iteration 16: Proposed new text for extract.predict: The task is to calculate the sum of numbers provided in the `numbers` field. The output `sum_of_numbers` must always be a single floating-point number.

****General Strategy and Important Rules**:**

- * Always identify and process only the valid numeric and mathematical components of the input.

...

N() Function from SymPy is used for numerical approximation, and it is essential for correctly evaluating functions like `arctan`, `sin`, `cos`, etc., and for general numerical results.

- * ****Error Handling and Default Value**:**

- * If the `symbolic_expression_sympy` tool returns an error, or if its output (even after being wrapped with `N()`) cannot be interpreted as a numeric float (e.g., for non-mathematical strings like "foo", or malformed expressions that cannot be numerically resolved), the `sum_of_numbers` for that input is explicitly `0.0`.

...

The `numbers` field can be provided in two main formats:

1. ****A list of items**:** The list can contain integers, floats, or strings.

- * ****Processing a list**:**

To wrap up DSPy



DSPy gives you a set of easy to use elements to compose into a AI system



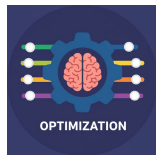
Engineer's focus on actual task, using LLM as a natural language specifications



AI Engineers figure out information flow



Good results from start (due to (AI) understandable structure, formatting rules and checks)



Model behaviours can be automatically optimized/ aligned to the task

Programmatic LLM & VLM use through DSPy

Igor Zubrycki @IgorZub



Measure your program

- prepare examples (input output pairs)
- use some metric (example output == model output, use a judge)
-

“Large Language Models are becoming extremely cheap/capable/ open source/ evolving

Behind is still essentially a stateless machine that accepts text/ images converted to tokens and outputs usually text / images

Magic is in packing essential context and creating a process to use output

```
code_snippet = """
from collections import deque

class Queue[T]:
    def __init__(self) -> None:
        self.elements: deque[T] = deque()

    def push(self, element: T) -> None:
        self.elements.append(element)

    def pop(self) -> T:
        return self.elements.popleft()
"""
language_guesser(code_example=code_snippet)
```

```
2025/08/10 21:05:33 WARNING dsp.predict.predict: Not all input fields were provided to module. Present:
['code_example']. Missing: ['hint'].
```

```
Prediction(
    programming_language_name='Python',
    programming_language_version='3.12'
)
```