Auto prompt improver using DSPy

9 min read · Just now

Jean-Baptiste Excoffier

Can a robot help another smaller one to become greater ? by Lenin Estrada on Unsplash

> *Hello ! You are an eager reader that wants to learn about* ***automatic prompt improvement techniques*** *and whether it really increases performances ?*

Well it already sounds like a template prompt, doesn't it ? 😏

Indeed, prompt engineering can be a bit tricky. So the question is : ***starting from a base prompt, can we automatically generate a better one ?***

What for ? We need a clear benchmark dataset in order to quantify the performances, without ambiguity.

How ? By using another language model for sure ! 🤖

Let's try it on a simple task consisting of arithmetics operations and compare multiple language models (large and small) either with basic and **self-improved prompts** !

⭐ **Here is the full repo on Github : https://github.com/JBExcoffier/prompt-improver**

Motivations

Prompt engineering is both context and model dependent.
Even when only considering its simplest form, *Zero Shot Prompting* corresponding to a direct instruction given to a generative language model, we usually need to 'fine-tune' the prompt to get the best responses.

And we need to do that with respect to the context, applied use-case with all its complexity, and usually the language model, with its inner knowledge limitations.

But, **a generative language model is by essence, able to produce a large variety of prompts with respect to a given task**. It could even improve the prompt and find the best one if we are also able to measure, in a quantitative manner, the performances of the responses.

This domain of generative AI is called *Automatic Prompt Engineering* (APE) and states that, as the name of a well-known research paper says, '*Large Language Models Are Human-Level Prompt Engineers*'.

🤔 Usually, a language model, called the **teacher model**, is used to improve the prompt of another language model, called **student model**.

Of course the teacher and student models can either be different (e.g. a large teacher model improving a smaller student model that would then be used in production since it's more efficient in terms of latency) or the same.

> *The teacher model generates prompts and selects the best one based on the performances they all yielded on a given dataset.*

Of course we then need a way of **measuring these performances** on a golden dataset, which is a set of requests and corresponding expected responses.
It can be a metrics, e.g. binary accuracy if it's a classification use-case, or a more complicated process, sometimes involving another language model to measure the quality of each response.

But let's keep it simple for this blogpost and let's focus on a rather straightforward use-case where multiple metrics can be used.

🚀 First step, let's benchmark multiple language models, both **external LLMs and local models**, with a basic prompt !

Arithmetics benchmark dataset

Let's take an **arithmetic operations task** !

A lot of benchmarks have already been established regarding the mathematical performances of language models. For instance the MATH level-5 benchmark by Epoch AI used a subset of the well known MATH dataset published in 2021 to evaluate numerous generative language models across various high-school level math problems.

👉 In this blogpost, we will keep it simple with mostly **basic arithmetic operations that only use integers and addition, subtraction, multiplication and division.**

We build a golden dataset of **100 arithmetic operations producing integer results.** We also ensured that all results are strictly different from 0 so there is no problem computing metrics such as Mean Absolute Percentage Error (MAPE).

Here are a few examples :

```
(((6 + 3) * 2) - 5) + ((9 / 3) + 1) # 17.0
(((5 + 4) * 2) - (9 / 3)) + ((12 / 4) + 2) # 20.0
(((14 - 6) / 2) + (5 * 3)) - ((12 / 3) + 1) # 14.0
```

Of course, such operations can be easily and correctly performed by a language model that has access to an external calculator.
But as I said, let's keep things simple and use this toy use-case to check if performances can be really increased with auto prompting.

Indeed, it is well known that generative language models are not the best, to say the least, at giving the exact result of a math operation, especially in a zero-shot prompting, *i.e.* direct answer without time and space for reasoning and performing step by step calculations before giving the final result.

Then, well, we need some **generative language models** ! We will use **both external and local** ones.

☁️ As for the external models, we will use the OpenAI GPT-4.1 models, with both the largest version (`gpt-4.1`) and its tinier 'nano' one (`gpt-4.1-nano`).

💻 As for the **local model setup**, we will use the SmolLM model with its 135 million parameters version , called SmolLM-135M-Instruct.
It really is a tiny model, but at least it can be **easily run on almost any machine, even on CPU** !

This SmolLM-135M-Instruct can be served locally using several libraries and softwares. We will use **vLLM** in Python, which has really strong performances for inference.

And, as I said, we will even set the `SmolLM-135M-Instruct` model up on CPU only as it is really a small model.

⌨️ ☁️ Setting up an **external** model in `DSPy` is really easy :

1. Just pass the name to the language model class of `DSPy` , e.g. `dspy.LM(model="gpt-4.1")` .

⌨️ 💻 Setting up the **local** model is a two step process :

1. Run a `vllm` instance of the model to make it available through a local API (based on `FastAPI` python package), with the following command (on CPU):

```
vllm serve $MODEL_PATH/ - tokenizer $TOKENIZER_PATH - host 0.0.0.0 - port 8080
```

2. You can then access this model using the language model class of `DSPy`. You just need an extra `"hosted_vllm/"` in the name of the local model as `DSPy` relies upon `LiteLLM`.

```
MODEL_NAME = "My-local-SmolLM-135M-Instruct-model"
BASE_URL = "http://0.0.0.0:8080/v1"
API_KEY = "super-strong-secret-token"

DSPY_MODEL_NAME = ("hosted_vllm" + "/" + MODEL_NAME)

lm = dspy.LM(DSPY_MODEL_NAME, api_base=BASE_URL, api_key=API_KEY)
```

📈 As for the benchmarking, we measure performances using the **accuracy** and the **MAPE** metrics.

The accuracy measures the model response is **exactly** the same as the expected answer. Of course, as results can have a lot of decimals, since it is a really strict metrics.
So we also use the Mean Average Percentage Error (MAPE) of how close (or far...) the responses are, on average, from the expected answers.

The **base prompt** is the following : `Answer` . Yes it is dumb, but at least straightforward !

Here are the performances for external and local LMs with base prompt.

Model	Accuracy (%)	MAPE (%)
gpt-4.1	26	63.7
gpt-4.1-nano	9	38.2
SmolLM-135M-Instruct	6	16.2

Auto Prompt Improver

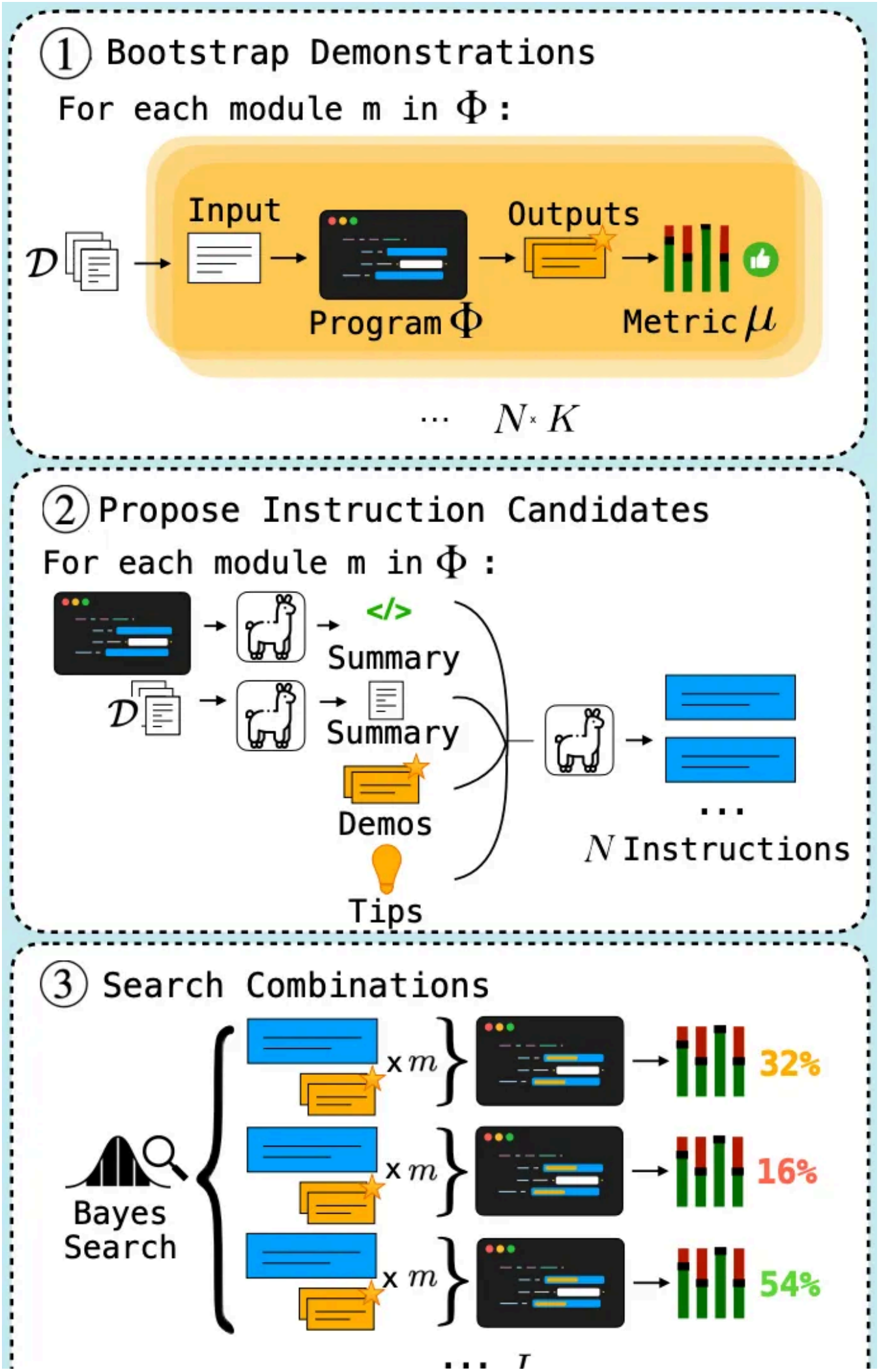
Now, what about the prompt improver technique ?

We are going to use the **DSPy Python package**, which stands for *Declarative Self-improving Python*, and use its *Optimizers* module to fine-tune the prompt to get the best instruction for our arithmetics task.

Multiple prompt improver techniques can be used but they all rely on the same general process explained just before.

⚙️ **A teacher model proposes a set of instructions for the task, then the student model is benchmarked for all of these prompts.**
Finally the best prompt is retained. This process is done multiple times in an **iterative** manner to really fine-tune the instruction.

📄 We will be using the MIPRO algorithm (*Multiprompt Instruction PRoposal Optimize*r in its second version) published in 2024.


① Bootstrap Demonstrations
For each module m in Φ:
Input
Outputs
Program Φ
Metric μ
... N × K
② Propose Instruction Candidates
For each module m in Φ:
</>
Summary
Summary
Demos
Tips
... N Instructions
③ Search Combinations
Bayes Search
x m
x m
x m
32%
16%
54%
... L

MIPRO main steps from original paper

For the **optimization metric**, we will use **MAPE**, as it provides a more nuanced measure of the difference between inferred and actual values compared to binary accuracy.

Obtained optimized prompts

Perfect ! Now let's see the optimized prompts for each couple of teacher & student models. Keep in note that the same student model can be 'instructed' by multiple teacher models, hence the different optimized prompts for a single student model.

Of course, we hope that a better (i.e. usually larger) teacher model proposes a better prompt, compared to a smaller teacher model, for the same student model.

Remember that the original prompt was a really simple `Answer` instruction !

- `gpt-4.1-nano` as **teacher**
  - `gpt-4.1-nano` as **student** : '*Given an input arithmetic expression, interpret and evaluate the expression to produce its numerical result. Handle complex nested structures with proper order of operations and provide the final answer as a floating-point number. Ensure accuracy in parsing and calculation to correctly reflect the expression's value.*'

- `gpt-4.1` as **teacher**
  - `gpt-4.1-nano` as **student** : '*Answer*'
  - `gpt-4.1` as **student** : '*Given a complex arithmetic expression written in standard mathematical notation (including nested parentheses and the four basic operations), calculate and provide the precise numerical result, following the correct order of operations. Output only the final computed value.*'
  - `SmolLM-135M-Instruct` as **student** : '*Given an arithmetic expression that may include integers, addition (+), subtraction (-), multiplication (*), division (/), and nested parentheses, carefully evaluate the expression by following the correct mathematical order of operations (parentheses first, then multiplication/division, then addition/subtraction). Parse and compute the expression step by step, ensuring all calculations are performed accurately and parentheses are respected. Output only the final numeric result of the expression.*'

- `SmolLM-135M-Instruct` as **teacher**
  - `SmolLM-135M-Instruct` as **student** : *'{proposed_instruction}'*

Two 'optimized' prompts are not really optimal...

One is of course when the local tiny language model, `SmolLM-135M-Instruct`, is both the student and the teacher, to self-improve its own prompt. It proposes a final non-sense prompt *'{proposed_instruction}'*. Most probably because it is still rather restricted in terms of generative performances to propose really various and original instructions at each step of the algorithm.

More surprisingly, `gpt-4.1` as the teacher model produces the same exact prompt as the base one (`Answer`) for `gpt-4.1-nano` student model. It might come from the fact that the student model is still rather small (not really a LLM), so for this task and its constraint of providing straightforward response (no chain of thoughts), a better prompt that really increases performances on the golden dataset could not be built.

However, all the other optimized prompts are much more descriptive. Indeed, they all propose a **clear three steps approach** :

1. Providing more context about the task.
2. Insisting on carefully parsing the input, step-by-step, in order to fully respect the order of operations.
3. Respecting the output format (i.e. direct answer).

Well ! On this simple task, that looks good at least !

Let's see the actual performances on the golden dataset for all prompts, either base and optimized ones.

Performances : original and optimized prompts

🎯 Remind that for accuracy, the higher the better. While it's the exact contrary for MAPE (the lower the better).

Models		Performance metrics	
Teacher	Student	Accuracy (%)	MAPE (%)
base prompt	SmolLM-135M-Instruct	6	63.7
base prompt	gpt-4.1-nano	9	38.2
base prompt	gpt-4.1	26	16.2
SmolLM-135M-Instruct	SmolLM-135M-Instruct	5	57
gpt-4.1	SmolLM-135M-Instruct	7	45
gpt-4.1-nano	gpt-4.1-nano	12	36
gpt-4.1	gpt-4.1-nano	9	38
gpt-4.1	gpt-4.1	22	14

🤔 Results are somewhat better, but not breathtaking...

For strong external models, `gpt-4.1` and `gpt-4.1-nano` there is a slight improvement in MAPE when comparing base prompt results to optimized prompts ones.
It is a bit better for the local `SmolLM-135M-Instruct` when 'taught' by `gpt-4.1` with a MAPE metrics dropping almost 20% (from 64% to 45%).

Final words

I wanted to check and measure the actual performance gain of a automatic prompt optimization method. The context was (too) simple to avoid high computation and all, but still generated prompts managed to get slightly higher performances.

Next step could be to **make automatic prompt optimization part of a broader use-case that would update itself over time to reduce model drift** (i.e. decrease in performances over time).

The multiple bricks of a system using language models (let's call them unsurprisingly AI agents) could update themselves when receiving new data that could either reflect a change in

- **users' behaviors** (incorporate them using some Reinforcement Learning from Human Feedback methods ?).
- **business objectives** (changes in the optimization target, let's it be a different optimization metrics or a different mix of multiple metrics).

This would make a great couple of nice things to try out ! 😃

💾 For-example as the number of prompts grows over time (optimized prompts for each AI agent alongside all the unretained ones), some prompt management tool could become very useful to track and monitor everything (as the MLFlow Prompt Registry tool).

⭐ As a reminder, the **full code is available on my GitHub page : https://github.com/JBExcoffier/prompt-improver**

> 👏 If you like this tutorial, give it some claps (here) and a star (on github) !

LLM | Prompt Engineering | AI | Benchmark | Data Science

Edit profile

Written by Jean-Baptiste Excoffier

36 followers · 164 following

Data Scientist : combining the best of Data & AI tools for clear and efficient business results