Training-Free Alignment of LLMs

A 3-Hour Hands-On Tutorial

Your Name

March 10, 2026

Prompt Engineering for Alignment

Designing effective instructions

Where Does Prompt Engineering Fit?

← *We are here* →

Stage	When	How
Pre-generation	**Before any token is produced**	**Modify the prompt**
In-generation	While tokens are being sampled	Shape the distribution
Post-generation	After full output exists	Filter / rerank / revise

Prompt Engineering Overview

- System prompts and instruction design
- Few-shot prompting strategies
- Chain-of-thought reasoning
- Automatic prompt engineering
- DSPy
- Hands-on tutorial demo

Prompt Engineering for Alignment

Why prompt alignment?

- Low cost: no need to retrain or fine-tune.
- Fast to iterate, easy to modify.
- No AI/ML expertise required, just typing!

Three Core Strategies:

1. **Zero-Shot**: Instructions only. No examples provided.
2. **Few-Shot**: Show examples of desired and undesired behavior.
3. **Chain-of-Thought**: Step-by-step reasoning before answering.

System Prompts for Alignment

Purpose: Set behavioral constraints and safety boundaries before any user interaction.

(Adapted from Google Prompting 101 guideline)

Four key elements:

- **Persona** — Role and scope of authority.
- **Task** — What to do, what not to do.
- **Context** — Provide background information
- **Format** — Output structure to reduce hallucination.

System Prompt Example (Zero-Shot Prompting)

Prompt: You are a math tutor. Solve the problem provided in the context based on a basic algebraic question. then provide the exact numeric answer only, without units, variables, or steps. Question: John had 10 books. He gave 3 to a friend and then bought 5 more. How many books does he have now?

Zero-Shot Prompting for Alignment:

- Instructions only, no examples provided.
- Relies on LLM pretrained knowledge.
- Alignment use: define safety rules the model must generalize.
- Limitation: fail on ambiguous edge cases.

Few-Shot Prompting for Alignment

Show the model examples of correct behavior before asking it to respond.

- Examples teach the model what to do and what not to do.
- Handles subtle cases better than instructions alone.
- Risk: bad examples lead to an unsatisfactory outcome.

Few-shot Example

You are a math tutor. Solve the problem provided in the context based on a basic algebraic question. then provide the exact numeric answer only, without units, variables, or steps.

[Example 1]
Question: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many does he have?
Answer: 11

[Example 2]
Question: The cafeteria had 23 apples. They used 20 for lunch and bought 6 more. How many do they have?
Answer: 9

Question: John had 10 books. He gave 3 to a friend and then bought 5 more. How many books does he have now?

Chain-of-Thought (CoT) for Alignment

- Requires the model to show reasoning steps before answering : making outputs transparent, verifiable, and more consistent with human expectations.

- Pairing CoT with few-shot examples teaches the model how to reason correctly : reducing errors, improving factual accuracy, and aligning multi-step decisions with human intent.

Zero-Shot CoT Example

Prompt:
You are a math tutor. Solve the problem provided in the context based on a basic algebraic question. then provide step-by-step reasoning before giving the final answer.
Question: John had 10 books. He gave 3 to a friend and then bought 5 more. How many books does he have now?
Answer: Let's think step by step

Few-Shot CoT Example

Prompt:
You are a math tutor. Solve the problem provided in the context based on a basic algebraic question. then provide step-by-step reasoning before giving the final answer.

[Example 1]
Question: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many does he have?
Answer: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

[Example 2]
Question: The cafeteria had 23 apples. They used 20 for lunch and bought 6 more. How many do they have?
Answer: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9.

Question: John had 10 books. He gave 3 to a friend and then bought 5 more. How many books does he have now?

Takeaway: Choosing the Right Prompt Alignment Strategy

Match the strategy to the task:

- **Zero-Shot**: Simple, clear tasks. Just give instructions.
- **Few-Shot**: Task has nuanced expectations that are hard to describe in words. Show examples instead (e.g., formatting, tone, refusal boundaries).
- **Chain-of-Thought** : Task requires multi-step reasoning or you need to verify how the model reached its answer (e.g., math, medical calculations, ethical decisions).

Automatic Prompt Engineering for Alignment

Why automate?

- Manual prompts are sensitive to LMs: small edits, big changes.
- Safety is hard to check by hand at scale.
- Automation can directly optimize for alignment goals.

How it works:

1. **Generate** : LLM proposes candidate prompts.
2. **Evaluate** : Score each for performance.
3. **Select** : Keep the best.

Key Frameworks

- **AutoPrompt** (Shin et al., 2020) : Gradient-guided search for trigger tokens.
- **Prefix/Prompt Tuning** (Li & Liang, 2021) : Optimize continuous soft prompts, weights frozen.
- **APE** (Zhou et al., 2022) : LLM generates instructions, scorer picks the best.
- **OPRO** (Yang et al., 2023) : LLM as optimizer: iteratively propose better prompts.
- **DSPy** (Khattab et al., 2023) : Modular LM pipelines compiled into optimized prompts.
- **TextGrad** (Pryzant et al., 2023) : Natural-language critiques as gradients to refine prompts.
- **MIPROv2** (Opsahl-Ong et al., 2024) : Bayesian optimization for prompt discovery in DSPy
- **SIMBA** (Under DSPy) : Stochastic Introspective Mini-Batch Ascent in DSPy
- **GEPA** (Srivastava et al., 2025) : Reflective prompt evolution in DSPy

This tutorial focuses on DSPy implemented optimizers, especially MIPROv2

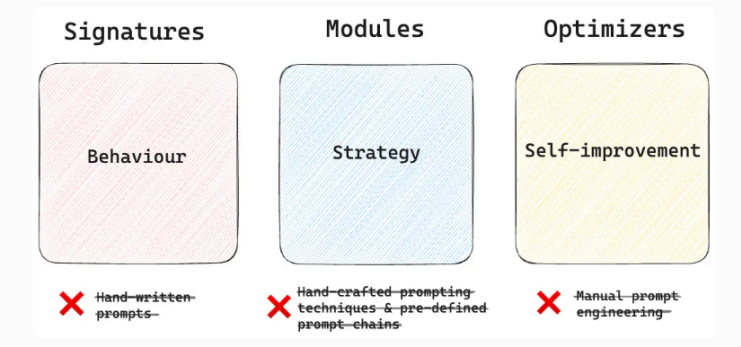
DSPy: An Example Automatic Prompt Alignment Method

DSPy replaces manual prompt engineering with modular, optimizable programs.

How it works:

- Define your task structure and reasoning strategy.
- DSPy compiles them into a system prompt automatically.
- Built-in optimizers tune the prompt using training data and a metric, no manual prompt writing needed.

DSPy: Core Abstractions



Source: Prashanth Rao, *Learning DSPy (1): The Power of Good Abstractions*, The Data Quarry Blog.
https://thedataquarry.com/blog/learning-dspy-1-the-power-of-good-abstractions/

How DSPy Compiles System Prompts For You

1. **Signature** : *What* the LLM should do.
   - Declare inputs and outputs (e.g., `question -> answer`).
2. **Module** : *How* the LLM should do it.
   - Choose a prompting strategy (e.g., `Predict`, `ChainOfThought`).

Together, Signature + Module generate the system prompt's structure.

3. **Optimizer** : Makes *better* prompt for better LLM response.
   - Finds the best demonstrations and instructions through training data.

Result: Signature + Module $\rightarrow$ prompt structure $\rightarrow$ Optimizer $\rightarrow$ fully aligned prompt.

DSPy Signatures

- Express what you want the model to do declaratively.
- Two types: simple shorthand and complex class-based definitions.

DSPy Signatures: Shorthand Version

String-based shorthand signature for basic and quick task. **Code:**

```
dspy_sig = dspy.Signature("str: question -> int: answer")
```

- Variable names matter: DSPy uses them to generate prompts.
- Typed fields guide structured, predictable outputs.
- Fast iteration for simple alignment.

DSPy Signatures: Class-Based Version

Define structured signatures with descriptions and constraints.

Code:

```
class QuestionAnswer(dspy.Signature):
   """Read the question carefully and solve the question."""
   question: str = dspy.InputField()
   answer: int = dspy.OutputField(desc="concise final answer, no units")
```

- Docstrings guide model behavior and can encode safety instructions.
- Field descriptions constrain valid outputs.

DSPy Modules

Building blocks for LM programs.

- Tell the model *how* to execute a signature (e.g., directly, step-by-step, with tools).
- Composable : chain modules together for multi-step tasks.
- Optimizable : the optimizer can improve module performance automatically.
- Built-ins: `Predict`, `ChainOfThought`, `ReAct`, and others.

DSPy Modules: Example with dspy.Predict

Code:

```
# Declare a Predict Module and type a shorthand signature
solve = dspy.Predict("question: str -> answer: int")


# Run the module
result = solve(question="Given 2x + 3 = 7, solve for x")
print(result.answer)
```

- Swap `Predict` for `ChainOfThought` with same the signature.

Signature

```
question: str -> answer: int
```

Module

```
dspy.Predict
```

Generated System Prompt:

```
System message:

Your input fields are:
1. `question` (str):
Your output fields are:
1. `answer` (int):
All interactions will be structured in the following way,
with the appropriate values filled in.

[[ ## question ## ]]
{question}

[[ ## answer ## ]]
{answer} # note: the value you produce must be a single int value

[[ ## completed ## ]]
In adhering to this structure, your objective is:
Given the fields `question`, produce the fields `answer`.
```

Signature

```
question: str -> answer: int
```

Module

```
dspy.ChainOfThought
```

Generated System Prompt:

```
System message:

Your input fields are:
1. `question` (str):
Your output fields are:
 1. `reasoning` (str):
2. `answer` (int):
All interactions will be structured in the following way,
with the appropriate values filled in.
[[ ## question ## ]]
{question}
 [[ ## reasoning ## ]]
 {reasoning}
[[ ## answer ## ]]
{answer} # note: the value you produce must be a single int value
[[ ## completed ## ]]
In adhering to this structure, your objective is:
Given the fields `question`, produce the fields `answer`.
```

DSPy Modules

`Predict`	`ChainOfThought`
`ChainOfThoughtWithHint`	`ReAct`
`ProgramOfThought`	`Refine`
`Majority`	`MultiChainComparison`

DSPy Modules

Each module wraps a different prompting strategy.

`Predict`	Direct input $\rightarrow$ output.
`ChainOfThought`	Step-by-step reasoning.
`ReAct`	Reason + act with tool use.
`ProgramOfThought`	Generate and execute code to solve.
`Refine`	Iterative self-correction.
`Majority`	Aggregate via majority vote.
`MultiChainComparison`	Compare multiple reasoning paths.
`ChainOfThoughtWithHint`	CoT with a guiding hint.

DSPy Optimizers

Automatically improve a module's performance using input-output examples.

- Tune prompts to maximize a metric of a task.
- No manual prompt iteration: the optimizer handles it.
- Alignment use: optimize for metrics of a task.

DSPy: Prompt Optimizers

Provide data and a metric, then the optimizer tunes the prompt automatically.

Optimizers

- `BootstrapFewShot` : Select high-scoring examples as few-shot demos.
- `MIPROv2` : Search over instructions and demos via Bayesian optimization.
- `GEPA` : Refines prompts by reflecting on past results.
- `SIMBA` : Improves prompts by looking at small batches of examples at a time.

DSPy Optimizer: Evaluation & Metrics

Every optimizer needs:

1. **Training data** : Labeled input-output examples (30–300+).
2. **A metric** : Scores output quality.

Built-in metrics:

- `answer_exact_match`
- `SemanticF1`
- **Custom** : Any Python function `(example, prediction) -> score`.
- LLM as a judge

DSPy Optimizer — MIPROv2 Example

Code:

```
import dspy
from dspy.datasets import MATH

# Configure LM
gpt4o_mini = dspy.LM('openai/gpt-4o-mini', max_tokens=2000)
dspy.configure(lm=gpt4o_mini)

# Load data and define program
dataset = MATH(subset='algebra')
module = dspy.ChainOfThought("question -> answer")

# Optimize with MIPROv2
optimizer = dspy.MIPROv2(metric=dataset.metric, auto="light")
optimized = optimizer.compile(module, trainset=dataset.train)

# Evaluate
evaluate = dspy.Evaluate(devset=dataset.dev, metric=dataset.metric)
evaluate(optimized)
```

DSPy: Iterative Alignment Loop

Compile — Evaluate — Iterate

1. **Program** : Define signatures and compose modules.
2. **Evaluate** : Measure against your alignment metric.
3. **Optimize** : Compile to improve performance.
4. **Iterate** : Refine until behavior matches intent.

What to adjust between iterations:

- **Data** : Add edge cases, fix labels.
- **Modules** : Swap modules (e.g., `Predict` → `ChainOfThought`).
- **Metric** : Target different alignment goals (correctness, factuality, tone).
- **Optimizer** : Scale from `BootstrapFewShot` to `MIPROv2` or `GEPA`.

Case Study

Do Aligned Prompts Transfer Across LLMs?

Our research: Prompt Alignment Transferability Under Automatic Optimization

Motivation

- Organizations switch LLMs often: new models, cost, policy changes.
- Optimized prompts are model-specific: alignment may not carry over.
- Re-optimizing for every model switch is expensive.
- **Goal:** Measure how well aligned prompts transfer across LLMs.

Research Questions

Q1: When you take a prompt optimized for one model and apply it to a different model, does it still perform well?

Q2: If a model has strong baseline performance, does it produce aligned prompts that work better on other models as well?

Setup: How We Test Alignment Transfer

Two steps:

1. Align : Optimize a prompt on a source model for a specific task.

2. Transfer : Apply that aligned prompt to a different model for the same task: does it still provide similar performance?

We repeat each evaluation n times and report the average score.

Self-Aligned vs. Cross-Aligned

- **Self-aligned:** Model uses a prompt optimized *for itself*
- **Cross-aligned:** Model uses a prompt aligned *on a different model*.
- **Key question:** Does alignment transfer or is it model-specific?

Measuring Alignment Transfer: Prompt Transferability Index (PTI)

$$\text{PTI} = \text{Cross-aligned score} - \text{Self-aligned score}$$

- PTI > 0 : Transferred alignment *improved* performance.
- PTI < 0 : Transferred alignment *degraded* performance.
- PTI ≈ 0 : No significant difference.

We use a statistical test (Wilcoxon rank-sum) to confirm whether the difference is meaningful.

Experiment Setup

Optimizers: MIPROv2 and SIMBA
Models: GPT-4.1-mini, Gemini-2.5-Flash-Lite, Llama-4-Maverick
Baseline: Zero-shot Chain-of-Thought

Two tasks:

- **Exp 1 : MedCalc-Bench:** Medical calculation reasoning (MIPROv2).
- **Exp 2 : MEDEC:** Clinical error detection (SIMBA).

For each task: align a prompt on one model, transfer it to the others, and measure if alignment holds.

Findings

Q1: When you take a prompt optimized for one model and apply it to a different model, does it still perform well?

- **No.** Transferred prompts often degrade vs. self-aligned prompts.

Q2: If a model has strong baseline performance, does it produce aligned prompts that work better on other models as well?

- **No.** And sometimes models with weaker baseline performance can generate *better* transferable aligned prompt.

Takeaway for the Case Study

- Aligned prompts don't fully transfer : switching models can break alignment.
- Model upgrades / switching may require re-aligning prompts, adding cost.
- **Ongoing work:** developing methods to make prompt alignment more generalized across models.

Poster Reception

Poster #33: Prompt Transferability Across LLMs: An Empirical Study Under Prompt Optimization

When: March 11, 2026, 5:00 PM to 6:30 PM
Where: Regency Foyer

Live Demo: Prompt Engineering for Alignment

Notebook:
prompt_engineering_for_training_free_alignment.ipynb

https://github.com/sonthenguyen/alignment