SAMIR GENAI ACADEMY

PROFESSIONAL STUDY GUIDE SERIES

Phase 01 · GenAI Foundations

03

Prompt Engineering Mastery

Zero-shot, few-shot, CoT, ToT & structured output techniques

Module 03 of 25 · **1 hour**

Instructor: **Samir Kumar** · Edition 2026

STUDY GUIDE

01 Overview

> Zero-shot, few-shot, CoT, ToT & structured output techniques. This guide gives you the core theory, the tools that matter, a hands-on build, working code, and the pitfalls to avoid — everything you need to learn the topic to a professional standard and apply it in real projects.

1 hour	4 + 4	4	03/25
LESSON LENGTH	CONCEPTS + BUILD STEPS	FRAMEWORKS & TOOLS	PHASE 01

02 Learning Objectives

- Apply zero-shot, few-shot, chain-of-thought (CoT) and tree-of-thought (ToT) patterns.
- Force reliable structured (JSON) output for downstream systems.
- Use role, delimiter, and example design to control model behaviour.
- Systematically optimise prompts instead of guessing.

03 Core Concepts

The prompting ladder

Start zero-shot (instruction only). Add few-shot examples when the task has a specific format or edge cases. Add chain-of-thought ('think step by step') for multi-step reasoning. Use tree-of-thought when the model must explore and compare alternatives before answering.

Structure & delimiters

Clear role assignment, XML/markdown delimiters, and explicit output schemas dramatically improve reliability. Separate instructions, context, and data with delimiters so the model never confuses them — critical for avoiding prompt injection.

Structured output

For pipelines, free text is fragile. Request strict JSON, provide the schema, and validate the result. Many APIs support a tool/JSON mode that guarantees parseable output — prefer it over parsing prose.

Systematic optimisation

Treat prompts as code: version them, build an eval set of representative inputs with expected outputs, and measure changes. Frameworks like DSPy can optimise prompts and few-shot examples automatically against a metric.

04 Visual Diagram

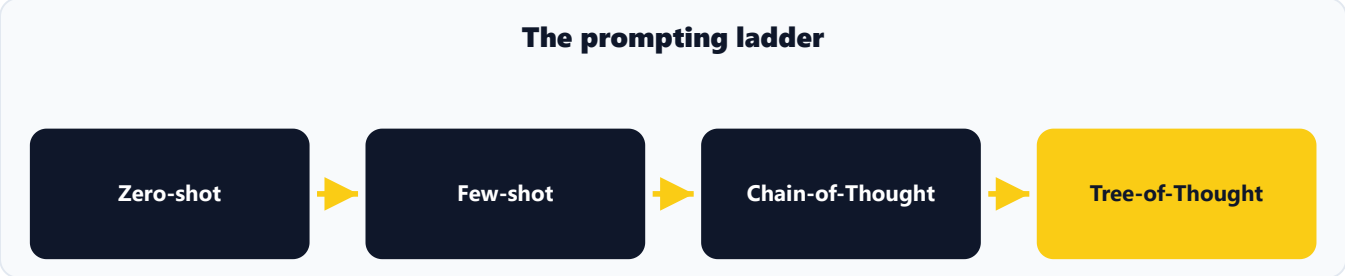

Figure 3 — Escalate techniques only as the task demands. Each rung adds reasoning power and cost.

05 Key Frameworks & Tools

DSPy | Pydantic (validation) | LangChain output parsers | Instructor

06 Hands-On Build

🛠 Structured JSON Output system + DSPy optimisation pipeline

1. Write a prompt that returns strict JSON for an extraction task; validate with Pydantic.
2. Compare zero-shot vs 3-shot vs CoT accuracy on a 10-item eval set.
3. Add delimiters and a role to harden against prompt injection.
4. Use DSPy to auto-optimise few-shot examples against your metric.

07 Code Walkthrough

Validated structured output with Pydantic — PYTHON

```python
from pydantic import BaseModel
import json

class Contact(BaseModel):
    name: str
    email: str
    company: str | None = None

prompt = (
    "Extract the contact as JSON matching: "
    '{"name": str, "email": str, "company": str|null}. '
    "Return ONLY JSON.\n\nText: Reach Asha Rao at asha@acme.io."
)
# ...send prompt to model -> raw...
raw = '{"name": "Asha Rao", "email": "asha@acme.io", "company": "Acme"}'
contact = Contact(**json.loads(raw))
print(contact)
```

08 🧪 Lab Assistant

Lab 3 · Reliable JSON extraction

Prerequisites:pip install pydantic · An LLM API key

1. **Prompt the model to return strict JSON for a contact-extraction task**
   > Expected: A parseable JSON object, no prose
2. **Validate the output with the Pydantic model**
   > Expected: A typed Contact object; ValidationError on bad output
3. **Compare zero-shot vs 3-shot accuracy on 10 inputs**
   > Expected: Few-shot improves edge-case handling

Troubleshooting

- Model wraps JSON in prose → add 'Return ONLY JSON'.
- Occasional invalid JSON → use the API's JSON/tool mode.

09 Pitfalls & Key Takeaways

⚠ COMMON PITFALLS

- Parsing prose with regex instead of requesting strict JSON.
- Adding examples that conflict with the instruction (confuses the model).
- Tweaking prompts by vibe with no eval set to measure impact.

✓ KEY TAKEAWAYS

- Escalate techniques only as the task demands: zero -> few-shot -> CoT -> ToT.
- Delimiters + explicit schema = reliable, injection-resistant prompts.
- Optimise prompts with evals, not intuition.

📺 **Companion video:** Watch the module 03 lesson in your Student Dashboard, then use this guide to revise, copy the code, and complete the build. Download both for offline study.

10 Further Resources

Anthropic & OpenAI prompting guides • DSPy documentation • 'Prompt Engineering Guide' (DAIR.AI)