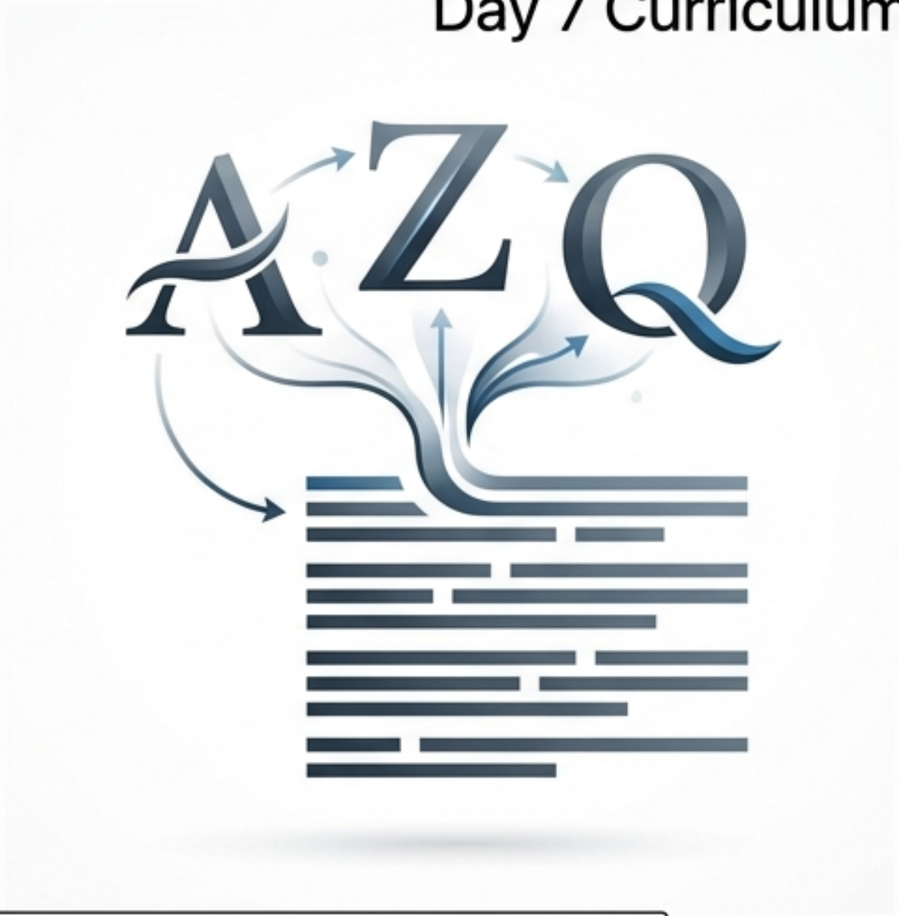
Language

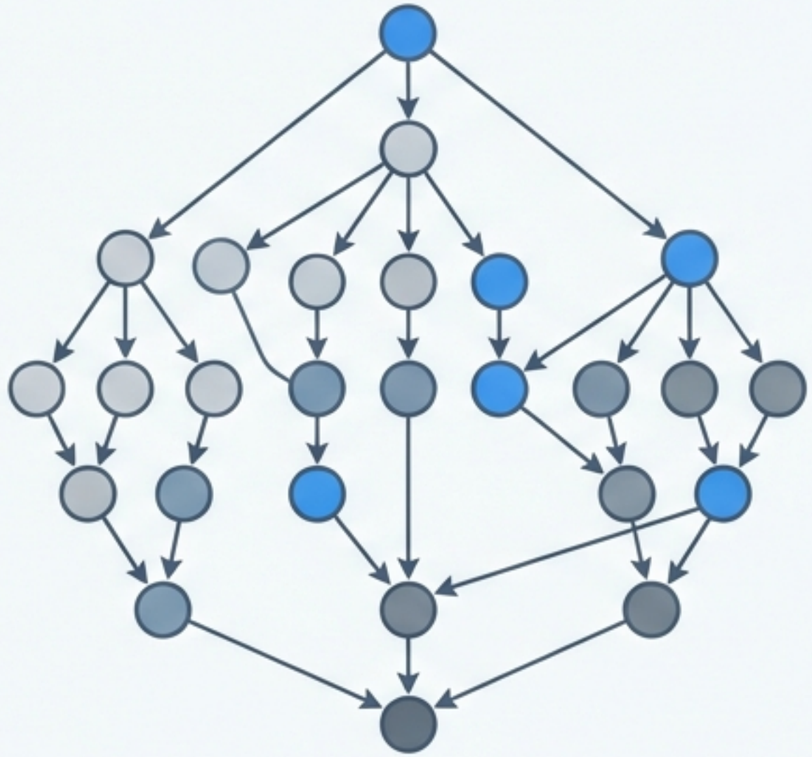
Structure

INDUSTRIAL AI PRACTICES

Advanced Prompt Engineering & Data Version Control

Day 7 Curriculum | From Prototype to Production





Trainer: Anshul | **Organization:** SkillOceans

Target Audience: 3rd Year AIML / Data Science

Workshop Roadmap

A Journey Through API Optimization, Data Management, and Applied AI

1

The Business Case

Reducing API costs & hallucinations.

Focusing on ROI and reliable AI performance.

2

Engineering Techniques

Role-Based, CoT, & Few-Shot methods.

Prompt engineering for enhanced reasoning.

3

Optimization

Constraints & token control.

Managing output length and ensuring efficiency.

4

The ‘Large Data’ Problem

Why Git fails with ML datasets.

Challenges of versioning large binary files.

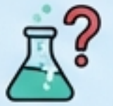
5

DVC Architecture

Workflow & mental models.

Utilizing Data Version Control for reproducible ML pipelines.

6



Lab Challenges

Applied exercises.

Hands-on practice to reinforce learning objectives.

Beyond the Chatbot: Engineering for Reliability

Concept

Definition: Advanced Prompt Engineering is the shift from conversational interaction to deterministic system design. It is the practice of designing inputs to produce consistent outputs from stochastic models.

The Problem: Basic prompts lead to hallucinations, format errors, and inconsistent tone.

The Cost: In production, bad prompts = wasted API credits and poor user experience.

Context

INDUSTRY INSIGHT

In real projects, prompt engineering is 50% language and 50% constraint management.

Technique 1: Role-Based Prompting

Anchoring the Latent Space

❌ Naive Approach

Write a marketing email.

Result: Vague, generic, often robotic.

✅ Engineered Approach

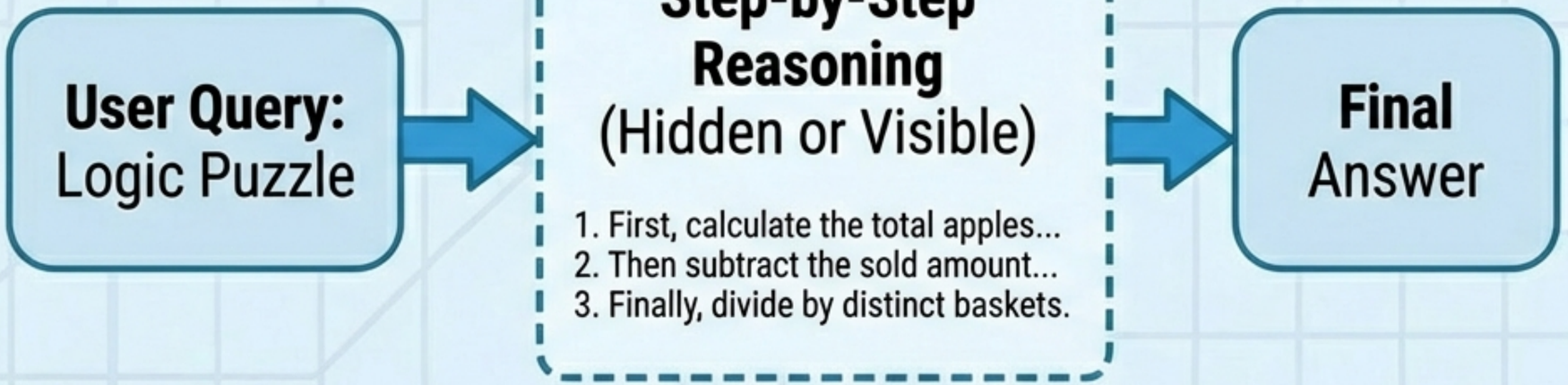
```
System: Act as a Senior Copywriter
for a B2B SaaS fintech.

User: Write a cold email to a CTO
regarding legacy infrastructure...
```

Result: Specific, professional, targeted tone.

Key Benefit: Reduces variance by narrowing the model's search space.

Technique 2: Chain-of-Thought (CoT)

Forcing the model to “show its work”



CONSTRAINT WARNING

Do not use CoT for simple retrieval tasks where low latency is required.

Reasoning steps increase token usage and cost.

Technique 3: Few-Shot Prompting

Pattern Matching via Examples

Zero-Shot

[Task] + [Query]

Few-Shot

Instruction: Extract Company Names

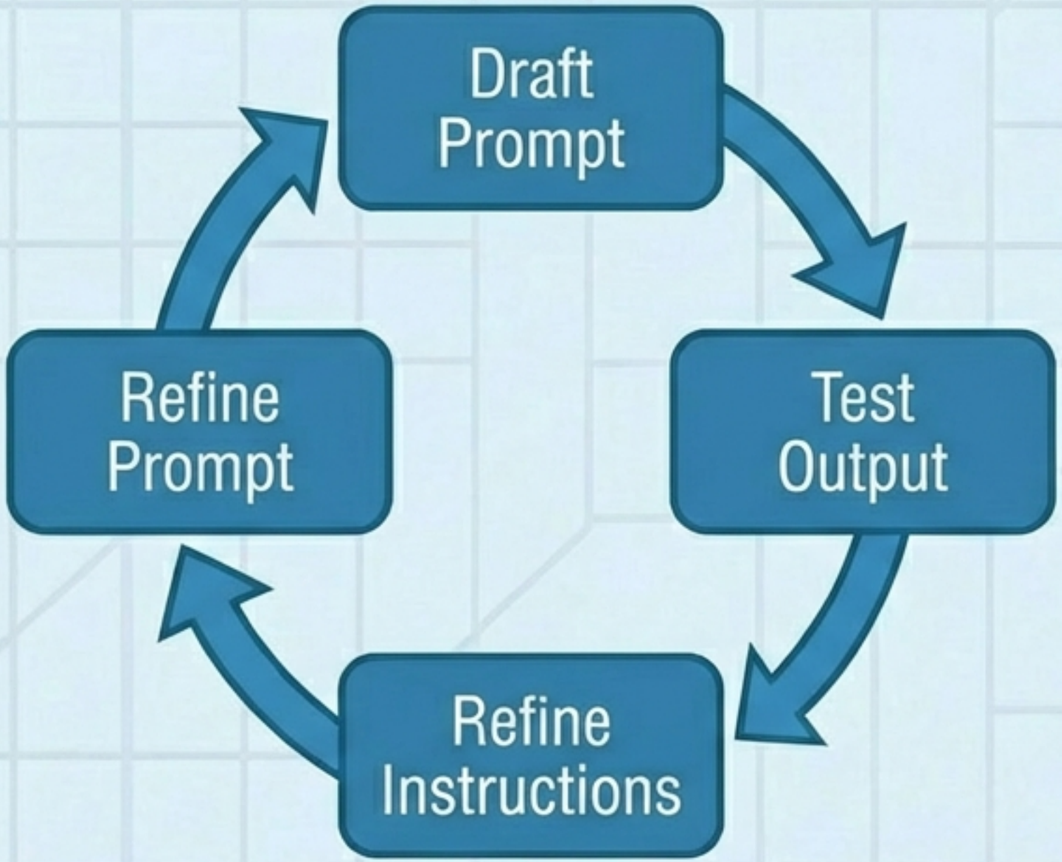
Example 1: “Apple released a new phone.” -> **Output:** Apple

Example 2: “The chaotic market impacted Tesla.” -> **Output:** Tesla

Query: “Microsoft announced earnings.” -> **Output:** ?

- **Definition:** Providing specific input-output examples within the context window.
- **Best Practice:** Use diverse examples that cover edge cases.

Optimization & Constraints



Production Controls

- **Token Control:** Balance clarity (verbosity) vs. context window costs.
- **Formatting Constraints:** Enforce machine-readability.

```
Instruction: Do not include preamble.
Output ONLY valid JSON format.
```

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

The ‘Heavy Data’ Problem

Why Standard Version Control Fails ML

The Conflict:
Git is optimized for text (code diffs), not large binary files.

The Symptoms:

- Bloated repositories (slow clones)
- “data_final_v2_real.csv” naming chaos
- Inability to reproduce old model versions

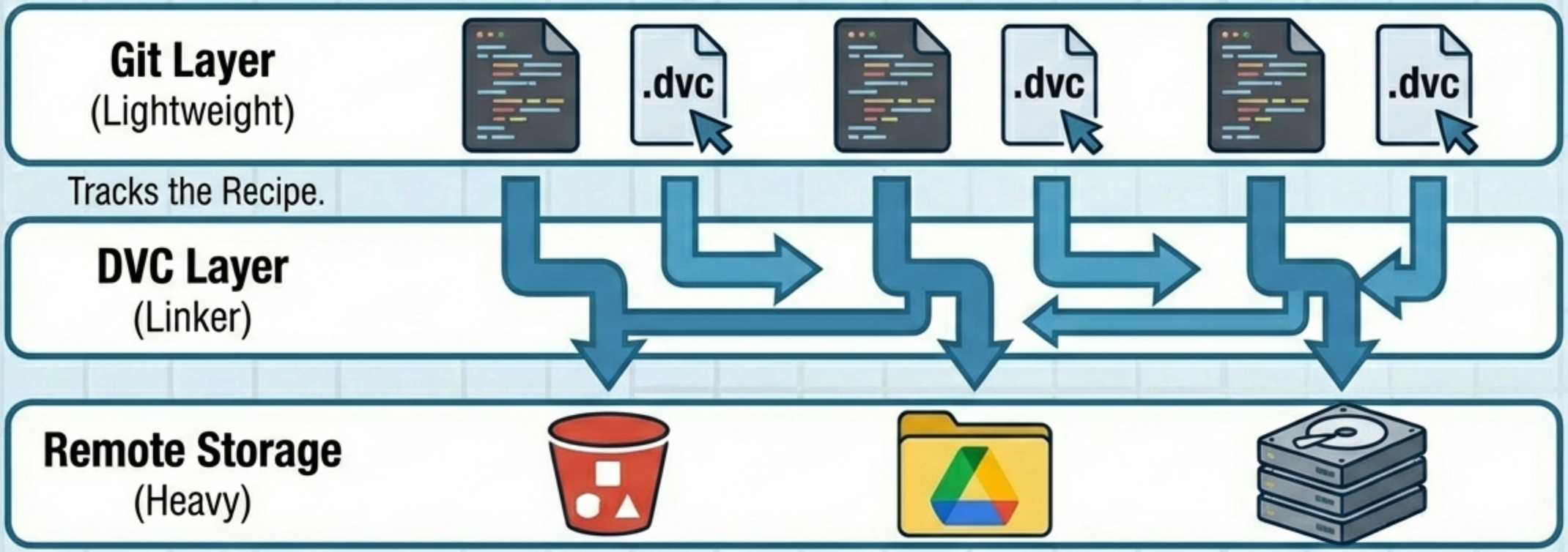
The Solution:

DVC acts as a pointer system, managing large files alongside Git.

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

DVC Architecture & Mental Model



Git tracks the pointers. **DVC** handles the actual data transfer.

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

Tool Comparison: Git vs. DVC

Feature	Git (Standard)	DVC (ML Optimized)
Code Tracking	✅ Excellent	❌ Not designed for this
Large Data Files	❌ Bloats repo, crashes	✅ Optimized storage
ML Pipelines	❌ No support	✅ DAG management
Storage Location	Internal (.git folder)	External (S3/Remote)

Key Insight: They are not competitors. **DVC** runs **ON TOP** of **Git.**

⚠ **INDUSTRY INSIGHT**

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

Installation & Initialization

```
$ pip install dvc
[Success message...]

$ dvc init
Initialized DVC project. Created .dvc/ config directory.

$ git commit -m 'Initialize DVC'
 1 file changed, 4 insertions(+)
```

Creates internal directory structure similar to **.git/**

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

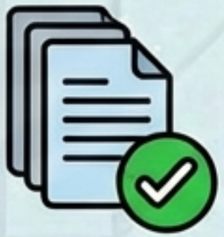
The Data Tracking Workflow

Tracking ‘data.csv’

```
$ dvc add data.csv
```

data.csv.dvc
(The Pointer)

```
$ git add data.csv.dvc
```



Git Staging Area

```
$ git commit
```

Git Repository

```
$ dvc push
```

Cloud/Remote Storage

REMEMBER

You never ‘git add’ the heavy file itself.

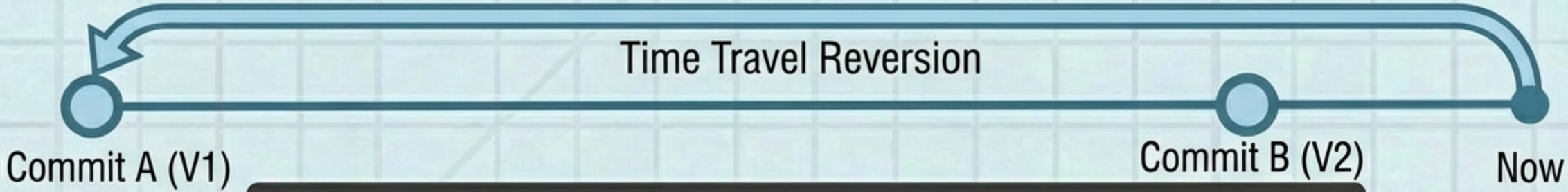
You only ‘git add’ the **.dvc** pointer file.

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

Time Travel with Data

Scenario: You cleaned the data (V2), but it broke the model. You need Raw Data (V1) back.




```
1. $ git checkout HEAD~1 data.csv.dvc
   (Reverts the pointer file in Git)

2. $ dvc checkout
   (DVC sees the old pointer and restores the physical V1 file)
```

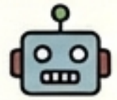
Result: Total reproducibility of experiments.

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

Lab Challenge: Prompt Engineering

CUT HERE



Objective:

Design a prompt to extract structured data from a messy invoice text.

1. **Role**: Act as a Data Extraction Bot.
2. **Strategy**: Use Few-Shot (Provide 1 example of Text -> JSON).
3. **Constraint**: Enforce strict JSON formatting.

Reflection Question: Why is this structured approach better than simply asking “What did I buy?”

INDUSTRY INSIGHT

Always treat LLM output as untrusted user input.
Enforce strict formatting to prevent pipeline breaks.

Production Readiness Checklist

Prompting Pitfalls

- ☐ Avoid over-complicating context.
- ☐ Don’t neglect edge cases in examples.
- ☐ Simplification is key.

DVC Pitfalls

- ☐ Never commit large files to Git directly.
- ☐ Don’t forget `dvc push` (colleagues need the data!).
- ☐ Always track the `.dvc` file.

INDUSTRY INSIGHT

AI Engineering requires mastering both the algorithm (the prompt) and the infrastructure (the data). Neglect either, and the system fails.