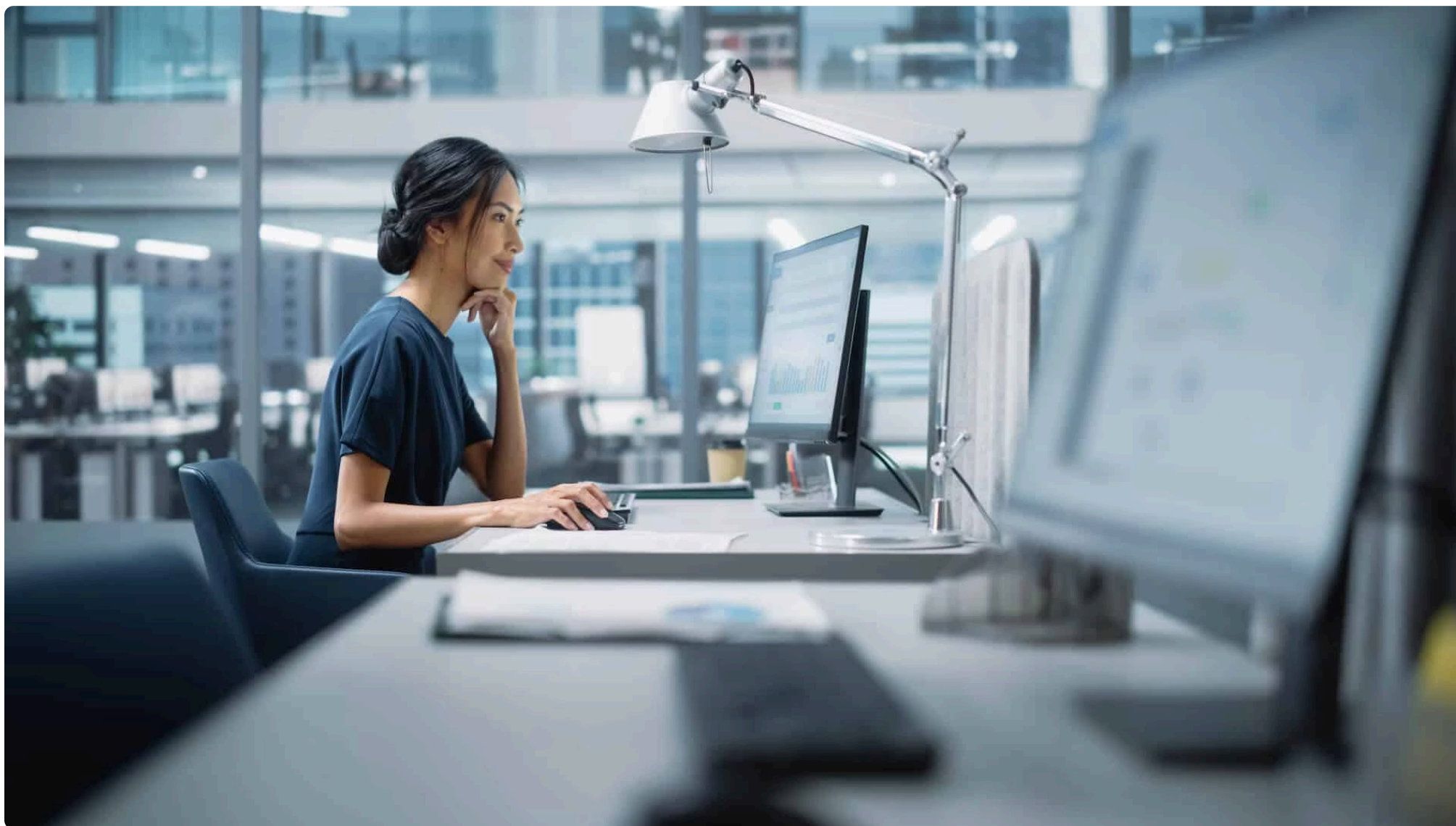


Advanced Prompt Engineering & AI-Augmented Development



Session 1 – Prompt Patterns & Testing

Presented by: Versha Jain

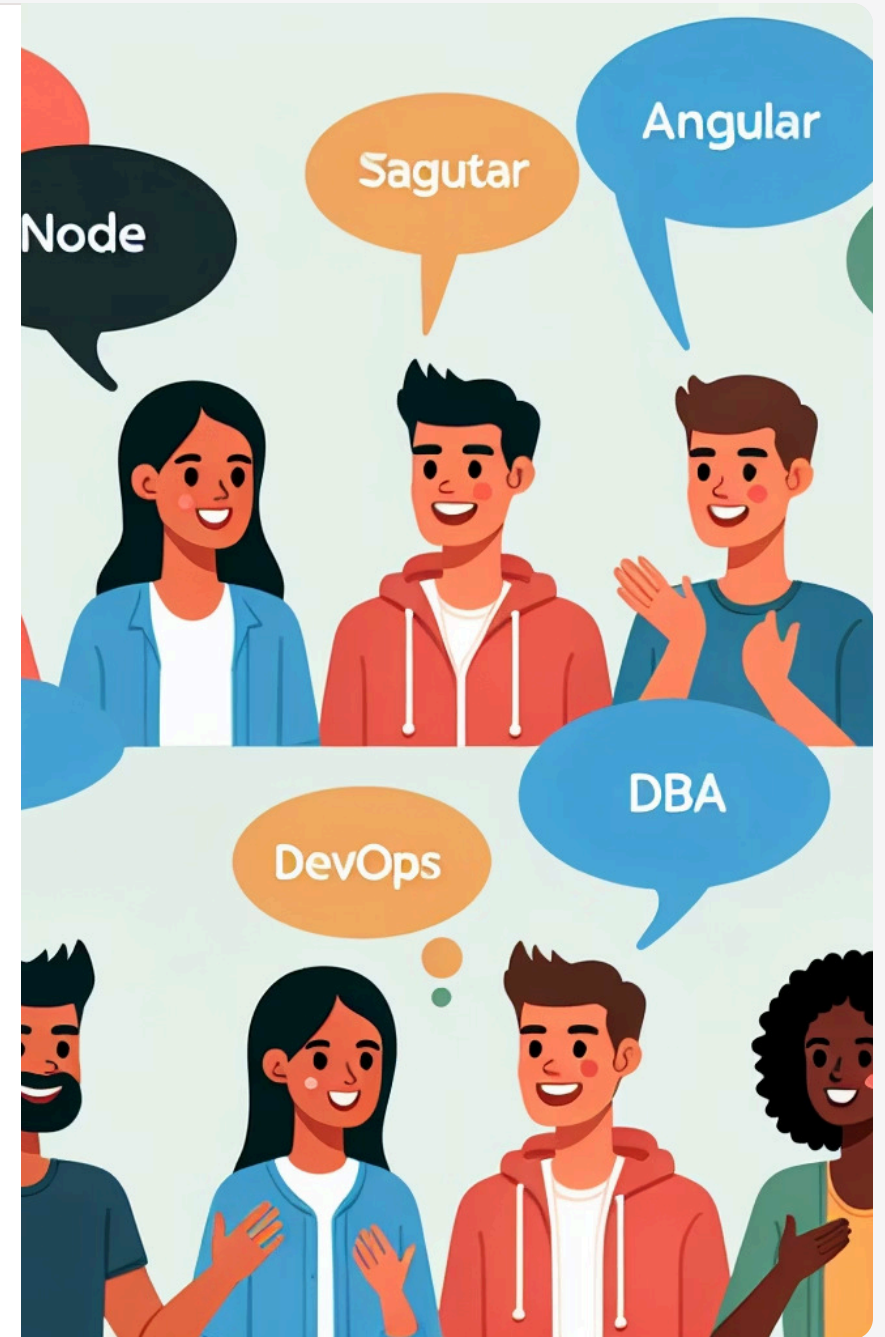
Who This Is For

Audience Profile

- Junior Devs (1–2 yrs)
- Mid-level Devs (3–5 yrs)
- Node, Angular, React teams
- Java, .NET specialists
- DevOps, DBAs, QA engineers
- LLM-curious Test Engineers

Common Goals

- Move beyond basic ChatGPT usage
- Apply LLMs to production tasks
- Improve codebase maintainability with AI
- Automate documentation, debugging, and refactor planning
- Establish a Prompt Library & Versioning Process





Session Objectives & Takeaways

Learning Goals

- Optimize LLM workspace settings and profile for daily use
- Understand and apply the **5 Golden Rules of Prompt Engineering**
- Compare and choose the right prompt pattern (**Zero-Shot, Few-Shot, CoT**)
- Produce structured, CI-valid output (**e.g., JSON, YAML**)
- Practice prompt debugging and validation in **Lab A**

Expected Outcomes

- Confidence using structured prompts in real development workflows
- Know how to **write testable, predictable AI prompts**
- Hands-on experience fixing broken prompts in real-time
- Preview of enterprise-grade AI augmentation coming in future sessions

Lab Roadmap

- **Lab 1: Code Review Assistant**
– Working tool that analyzes code quality
- **Lab 2: Documentation Generator** – Automated docs from messy codebases
- **Lab 3: Debug Helper System** – Systematic approach to complex bugs



Profile Setup for LLM Workspace

Profile Details

- **Full Name:** Use your real name or team handle (e.g., versha)
- **Role Description:** *"A senior software architect with 15+ years experience in enterprise systems"*

Expertise Tags

- Legacy system analysis
- Java, .NET, Python architecture
- DB tuning, production reliability

System Prompt / Instructions

Add to main chat profile:

"You are a senior enterprise architect. Respond with step-by-step analysis, implementation-ready suggestions, and fallback plans. Consider scalability, compliance, and reliability constraints."

Personalizing ChatGPT for Better Results



Why Customize ChatGPT?

- Makes interactions more natural and context-aware
- Helps ChatGPT adapt to your work style
- Reduces repetitive instructions in prompts



Customization Options

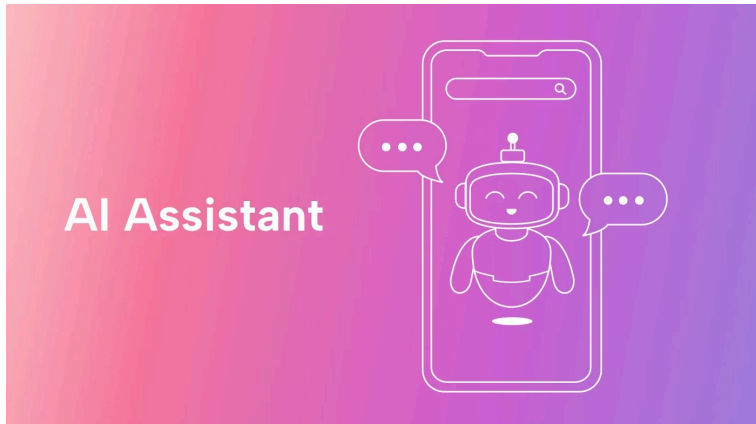
- **Name:** Personal tone in replies
- **What you do:** Guides ChatGPT to align responses to your domain
- **Traits:** Defines personality and response style



Example Traits to Use

- Professional yet concise
- Explains with real-world coding examples
- Step-by-step guidance for troubleshooting
- Structured responses (bullet points, numbered steps)
- Acts like a senior developer mentor
- Gives quick copy-paste ready code

Customizing Your AI Persona



Define your AI's persona for tailored responses.

To ensure your AI assistant provides tailored and effective responses, it's crucial to define its persona and guidelines clearly. This example demonstrates a detailed prompt for a DevOps engineer:

"I am a DevOps engineer working on high-risk production servers. Act as a cautious mentor: avoid destructive commands, suggest safe automation scripts, include rollback plans, and recommend cost-saving API strategies."

This instruction set guides the AI to adopt a specific role, prioritize safety, and align its suggestions with the user's operational constraints and goals.

Recommended Traits for Software Engineers



Concise Code Suggestions

Offers clear, simple, and production-ready code that adheres to KISS principles.



Production Safety

Provides safe commands, avoiding destructive actions by default.



Risk Flagging

Identifies and flags potentially risky commands (e.g., `rm -rf`, `DROP DATABASE`) before suggestion.



API Cost Optimization

Explains cost implications of API calls and suggests optimizations.



Code Readability & Maintainability

Emphasizes and guides on writing clean, maintainable code.



Step-by-Step Debugging

Provides clear, sequential guidance for troubleshooting and debugging.



QA Test Coverage Advice

Advises on comprehensive QA test coverage, including edge cases.



Structured Clarity

Ensures responses are clear, organized, and use structured formats like bullet points.



Settings-to-Traits Mapping — How to Ask It

Here's how to phrase prompts that instruct ChatGPT to use its settings effectively:

Act as my ChatGPT setup expert. Confirm my settings:

1. Data Controls: chats not used to train.
2. Custom Instructions: role = production DBA; traits = safe SQL, explicit warnings, backup steps.
3. Connectors: GitHub and Drive enabled, read-only.
4. Memory: on/off — summarize memory scope.

Then run test alert: "Give me a script to clean old logs on production. Warn me if any unsafe commands are implied."



Data Privacy & Workspace Integrity



Data Control Settings

Go to **Settings** → **Data Controls** and turn off "Improve the model for everyone." This ensures chats are not used for model retraining, and OpenAI stores them only for safety review for up to 30 days.



Temporary Chat Mode

Utilize Temporary Chat mode for conversations that auto-delete. This is ideal for sensitive or PII-heavy topics, ensuring transient data handling.



Secure Account Practices

Enable Multi-Factor Authentication (MFA) for enhanced security. Additionally, use anonymous or pseudonymous emails to protect your identity and maintain privacy.



Custom Connector Setup

Enable Connectors

Access this feature within **Settings** → **Beta Features** → **Connectors** to unlock integration capabilities.

Utilize Built-in Options

Leverage existing connectors like Google Drive, GitHub, and Datadog for direct data querying and inline responses.

Enhanced Context with Memory

With "Memory" enabled, ChatGPT can combine remembered details and connector data for more contextually rich responses.

Data Training Assurance

For Enterprise, Education, and Team deployments, connector data is **not** used for model training (Free/Plus plans may use it if opted in).

Create Your Own GPT

Unlock the full potential of AI by building custom GPTs tailored to your specific needs and workflows.

Tailored AI Behavior

Customize your GPT's instructions, capabilities, and knowledge base to align perfectly with your unique tasks and information domains.

Integrated Data Sources

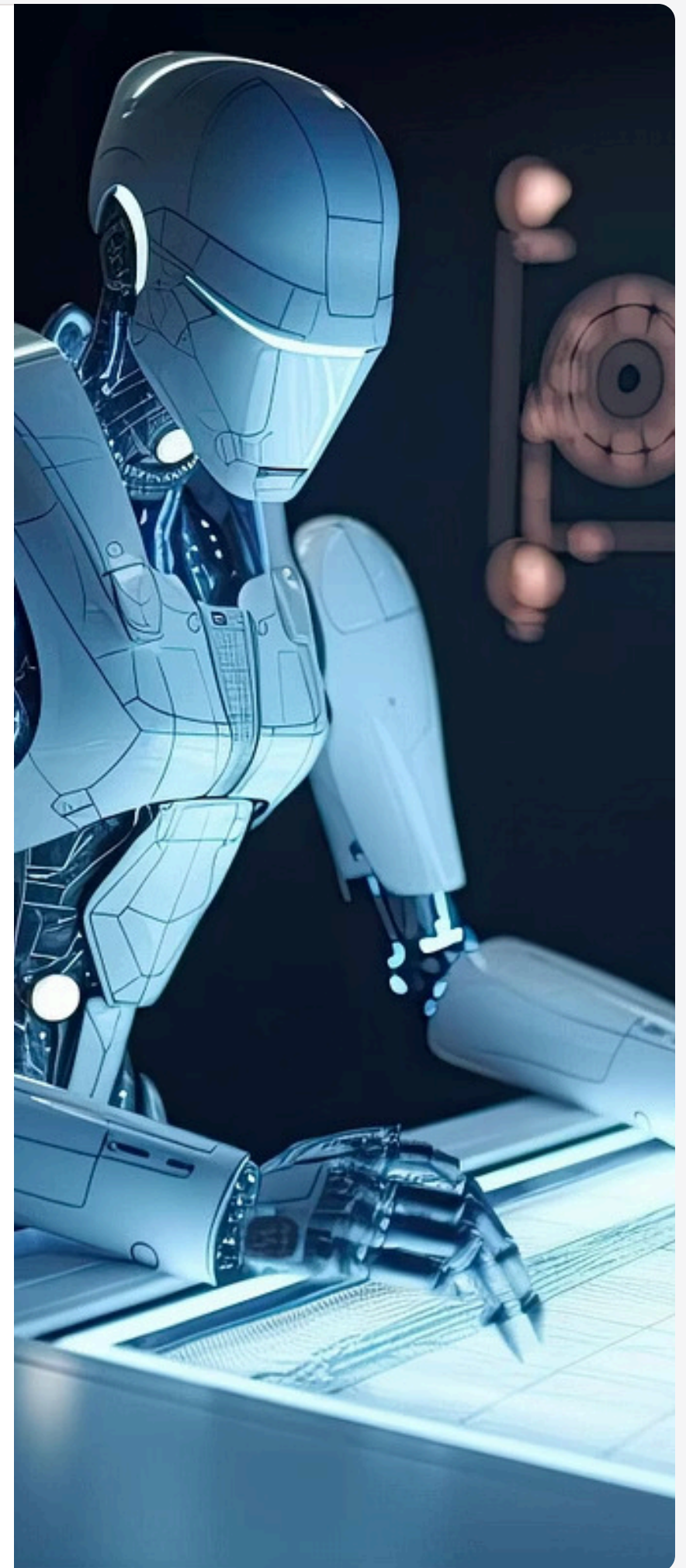
Connect your custom GPT to external data sources and APIs, allowing it to access real-time information and provide contextually rich responses.

Boost Productivity

Automate repetitive tasks, streamline complex processes, and enhance the accuracy and relevance of AI interactions for your specific use cases.

No-Code Creation

Build powerful custom GPTs without writing a single line of code, making advanced AI development accessible to everyone.





Build Your Own GPTs (Hands-On Lab)

Session Goal

- Learn how to create **custom GPTs** using *My GPTs* feature.

Build 3 Specialized GPTs for Real Software Engineering Tasks:



Log Parser Assistant

(DevOps/Support)



CI/CD Pipeline Reviewer

(DevOps)



SQL Best-Practice Advisor

(Developers/DBAs)

Focus on:

- Custom instructions
- File knowledge upload
- Safety & governance settings
- Hands-on building and testing



GPT Builder Essentials

Key Features

- Custom instructions: Define role, tone, scope.
- Knowledge upload: Private docs, logs, manuals (20 files, 512 MB each).
- Tools: Code Interpreter, Web Search, DALL·E.
- API Actions: External integrations (optional).

Governance

- Avoid sensitive data or secrets.
- Keep GPT **Private** or **Link-only** for internal tools.
- Opt out of training data use (Data Controls).

Outcome

A reusable assistant tailored to your workflow.



GPT #1: Log Parser Assistant

- **Goal:** Analyze server logs to extract errors, timelines, and root causes.
- **Setup:**
 - **Instructions:** *"You are an expert log analysis assistant. Parse timestamps, identify repeated errors, summarize key events, warn about anomalies."*
 - **Knowledge:** Upload sample log files and error code reference sheet.
 - **Tools:** Enable *Code Interpreter* for parsing/counting errors.
 - **Starters:** "What are top 3 errors?" | "Summarize crash events 2–3 PM."
- **Expected Result:**
 - Answers with counts, patterns, root causes quickly.
 - Suggests next steps for resolution.



GPT #2: CI/CD Pipeline Reviewer

- **Goal:** Automatically review pipeline configs for quality, speed, and security.
- **Setup:**
 - **Instructions:** *"You are a senior DevOps expert reviewing CI/CD YAML/Jenkins files. Identify inefficiencies, missing tests, caching, and security flaws."*
 - **Knowledge:** Upload sample pipeline file + best practices checklist.
 - **Tools:** Code Interpreter enabled.
 - **Starters:** "Analyze this workflow for risks." | "How can we speed up this pipeline?"
- **Expected Result:**
 - Flags misconfigurations and missing validation steps.
 - Suggests safe improvements (no hardcoded secrets, add health checks).



GPT #3: SQL Best-Practice Advisor

- **Goal:** Optimize SQL queries and schema design for performance and reliability.
- **Setup:**
 - **Instructions:** *"You are a senior database mentor. Analyze SQL queries for inefficiencies, missing indexes, or risky patterns. Suggest optimized queries and best practices."*
 - **Knowledge:** Upload company SQL guidelines or schema export.
 - **Tools:** Code Interpreter for analyzing execution plans or sample data.
 - **Starters:** "Optimize this query..." | "Review this schema for normalization issues."
- **Expected Result:**
 - Provides rewrite suggestions with reasoning.
 - Highlights indexing and design improvements.



Hands-On Lab Steps

1. Open ChatGPT → **GPTs** → **Create GPT**.
2. Configure each GPT:
 - Name + Description
 - Instructions (role, safety, tone)
 - Upload files (logs, pipeline YAML, SQL docs)
 - Enable required tools
 - Add conversation starters
3. Test in preview with real queries.
4. Iterate and refine until responses are accurate.
5. Save as **Private or Link-only** GPT for team use.



Best Practices

- Keep GPTs **focused on one task** for best results.
- Add clear safety rules (never suggest destructive commands, confirm risky steps).
- Regularly **update files and instructions** as tech stack evolves.
- Use **link-sharing internally**, avoid public listing unless sanitized.
- Combine multiple GPTs for different team needs (specialized > one big GPT).



Structuring Your GPT Project Folder








Goal: Keep all resources for GPT building organized for faster setup, testing, and updates.

Recommended Folder Layout

pgsql

Copy

Edit

 MyCustomGPTs_Project |  Logs_Samples → Raw server logs, error files for GPT testing |  Pipelines → Sample CI/CD
YAML, Jenkinsfiles |  SQL_Guides → SQL best-practice docs, schema exports |  Knowledge_Files → Any extra domain
PDFs, manuals |  Prompts_&_Instructions | | role_prompts.md → Pre-written system instructions for each GPT | |
test_prompts.md → Starter questions for preview testing |  Versions → Archived GPT configs or notes for v1, v2, etc. | |
README.md → Quick summary of folder purpose & usage

◆ Best Practices

- Keep **files small, relevant**, no sensitive data.
- **Version control** (Git or local copies) for instruction changes.
- Use a **README** to guide other team members on how to reuse GPT resources.
- Separate **testing inputs** from final reference docs to avoid confusion.
- **Link this folder** when running labs, so files are ready for upload to GPT builder.



The 5 Golden Rules of Prompt Engineering



Clarity

Ask exactly what you want



Context & Constraints

Give background, set boundaries



Role Assignment

Assign expertise



Few-Shot

Show before/after or input/output



Iterate & Test

Prompt → Try → Refine

These aren't tips — they're requirements.

Golden Rule 1 – Clarity

Say What You Mean, Mean What You Say

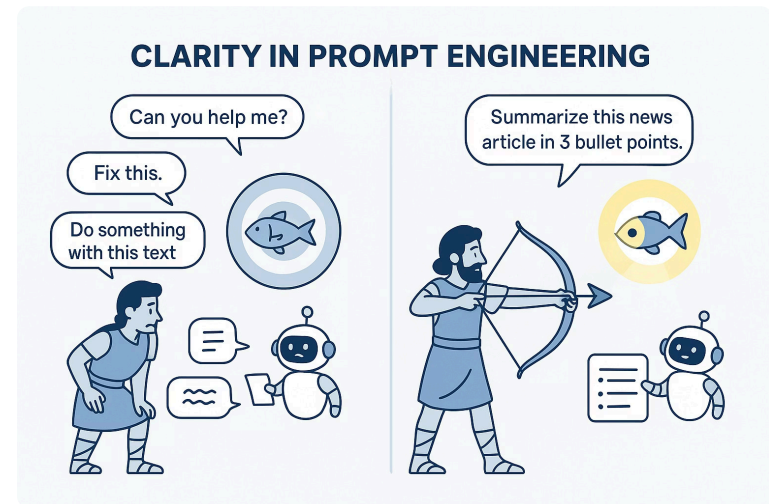
Clarity in prompts means being direct and unambiguous, ensuring the AI understands your exact intent.

❌ ❌ "Help me with this log file"

✅ ✓ "Extract error messages from this Apache access log.
Return only HTTP 5xx errors with timestamp and URL."

Why the "Good" Prompt Works:

- ✓ **Specific Action:** Clearly states "Extract error messages"
- ✓ **Clear Criteria:** Defines "HTTP 5xx errors"
- ✓ **Defined Output:** Specifies "timestamp and URL"





Be Clear & Specific

API Error Troubleshooting

Broken: "Fix this API error."

Fixed: "Investigate the 500 error returned by the /checkout endpoint in our FastAPI backend when payload includes more than 50 line items; identify root cause and suggest code fix."

Performance Optimisation

Broken: "Improve performance."

Fixed: "Profile and optimise our process_payments() function in Node.js (Express, MongoDB backend) that spikes to 2s latency under 5k concurrent users. Suggest code and DB index changes."

Test Case Generation

Broken: "Generate test cases."

Fixed: "Generate Jest unit tests for orderValidator.js covering invalid inputs (empty cart, negative quantity, expired coupon) with expect assertions for thrown errors."

Golden Rule 2 – Context & Constraints

Give AI the Background It Needs

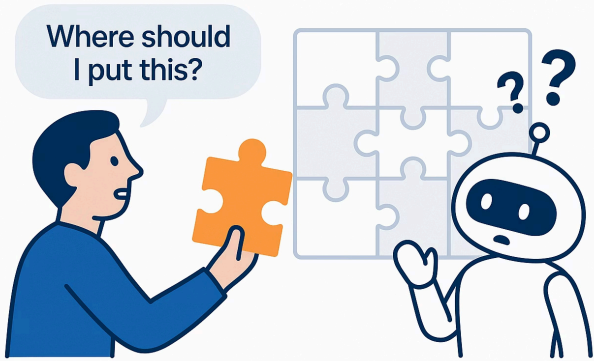
✗ Broken Example:

"Fix this function"

✓ Fixed Example:

"You're a senior Python developer reviewing legacy code. This function processes user uploads but crashes on large files. Constraints: Must maintain backward compatibility. Fix the memory issue without changing the API."

Context Matters



Key Elements

Avoid destructive commands, suggest safe automation, include rollback plans, recommend cost-saving API strategies



Role

Senior Python developer



Problem

Legacy upload function with memory crash



Why

Crashes on large files



Constraint

Must not change the API



Provide Context

Clarity isn't enough; your AI needs the right background information to deliver precise and relevant outputs. Providing context helps the model understand the exact scope and parameters of your request.



Log Analysis

✗ Broken: "Summarise this log."

✓ Fixed: "Summarise this Kubernetes pod log (container payment-service, namespace prod), focusing on repeated connection refused errors to Redis, from the past 1 hour."



Database Query

✗ Broken: "Write SQL query."

✓ Fixed: "In our Postgres DB (table transactions with indexes on user_id and timestamp), write a query to fetch last 10 failed payment attempts for a specific user, ordered by timestamp."



Bug Fixing

✗ Broken: "Fix this bug."

✓ Fixed: "Fix the bug in notifications-service where queued emails aren't sent when Kafka topic email-events has >1000 messages, using Spring Boot 3.2."

Golden Rule 3 – Role Assignment- Expertise Matters



Senior SRE

Reliability, scalability, and performance.



DBA

Database design, optimization, and security.



Security Architect

Vulnerability assessment and secure coding.



Software Engineer

Code structure, algorithms, and design patterns.

SAME INPUT. DIFFERENT MINDS.

Refactor for readability and maintainability.



TESTER

```
data =data
process_data(a)
print(result)
```

What's the attack surface?

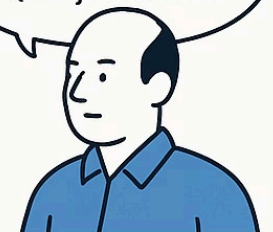


SECURITY ENGINEER

What edge cases could break this?



Check query performance and SQL injection risk



"Review this database query"

Problems with the broken example:

- No expertise level specified (e.g., "junior", "senior").
- No specialisation defined (e.g., "PostgreSQL", "MongoDB").
- No review criteria provided (e.g., "performance issues", "security risks").



"You are a senior DBA with 10 years PostgreSQL experience. Review this query for performance issues and security risks."



Define Constraints

Clearly defining constraints for the AI ensures highly relevant and precise outputs. Think about format, length, and specific elements to include or exclude.



API Documentation

Broken: "Write API documentation."

Fixed: "Generate OpenAPI 3.0 spec documentation for /api/v2/orders, JSON output under 200 lines, including example request/response bodies and HTTP status codes."



Dashboard Creation

Broken: "Make a dashboard."

Fixed: "Create a Grafana dashboard JSON config showing error rate and p95 latency for order-service over last 24 hours, limit to 4 panels on one screen."



Script Writing

Broken: "Write a script."

Fixed: "Write a Bash script to clean up temp files >7 days old under /var/log/myapp/, under 30 lines, safe to run on Ubuntu 22.04 without deleting hidden files."

Golden Rule 4 – Few-Shot Examples

Show, Don't Just Tell

Few-shot examples are critical for guiding the AI when the exact output structure is essential. By providing concrete examples, you clearly teach the AI the desired format and reduce ambiguity, ensuring precise and predictable outputs.

Zero-shot (telling):

"Convert log lines to JSON format"

Few-shot (showing):

"Convert log lines to JSON format.

Input: '2024-01-15 ERROR Database connection failed'

Output: {"timestamp": "2024-01-15", "level": "ERROR",
"message": "Database connection failed"}

Input: '2024-01-15 INFO User login successful'

Output: {"timestamp": "2024-01-15", "level": "INFO",
"message": "User login successful"}

Now convert: '2024-01-16 WARN Memory usage at 85%'

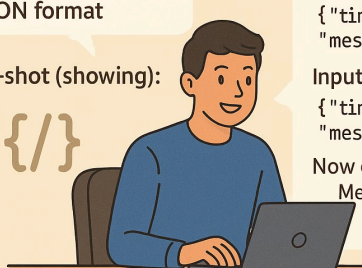
Golden Rule 4 – Fow-Shot Examples

Show, Don't Just Tell

Zero-shot (telling):

Convert log lines to JSON format

Few-shot (showing):



Convert log lines to JSON format.

Input: "2024-01-15 JSON format.
{ "timestamp": "2024-01-15", "ERROR",
"message": "Database connection failed"}"

Input: "2024-01-15 INFO User login successful
{ "timestamp": "2024-01-15", "INFO",
"message": "User login successful"}"

Now convert: "2024-01-16 WARN
Memory usage at 85%"

Now convert: '2024-01-16 Memory u85%'

Teaches Structure
Provides clear examples for the AI o follow.

Reduces Ambiguity
Ensures precise formatting and
avoids misinterpretations.



Teaches Structure

Provides clear examples for the AI to follow.



Reduces Ambiguity

Ensures precise formatting and avoids misinterpretations.



Critical for Output Format

Most effective when the exact output structure is essential.



Specify Role & Tone

1

Technical Explanation

Broken: "Explain data pipeline."

Fixed: "As a senior data engineer mentoring juniors, explain our Airflow DAG structure for nightly ETL jobs moving data from S3 → Snowflake, in plain English, under 200 words."

2

Client Communication

Broken: "Write an email to client."

Fixed: "As a professional project manager, draft a concise email to ClientX explaining yesterday's AWS outage, our mitigation steps, and expected resolution ETA, in a calm reassuring tone."

3

Documentation

Broken: "Generate README."

Fixed: "As a senior dev on a microservices repo, generate a concise README for the auth-service module, tone: technical and instructional, target audience: internal dev team."



Show, Don't Just Tell (Few-Shot Examples)

Commit Messages

Demonstrate the desired format for commit messages with examples.

Example:

Input: Fixed null pointer on login

Output: fix(auth): handle null token in login flow

Input: Added retry for failed payments

Output: feat(payments): add exponential retry to payment processor

Now convert: Fixed error on profile page when user data is empty

Log Conversion

Illustrate how to convert raw log entries into structured JSON format.

Input: [2025-07-29 12:03:55]

WARN: Disk usage 92%

Output: {"timestamp":"2025-07-29T12:03:55Z","level":"WARN","message":"Disk usage 92%"}

Now convert: [2025-07-29 12:04:10] ERROR: Redis timeout

PR Titles

Show how to generate concise and informative Pull Request titles.

Example:

Diff summary: Adds unit tests for order API

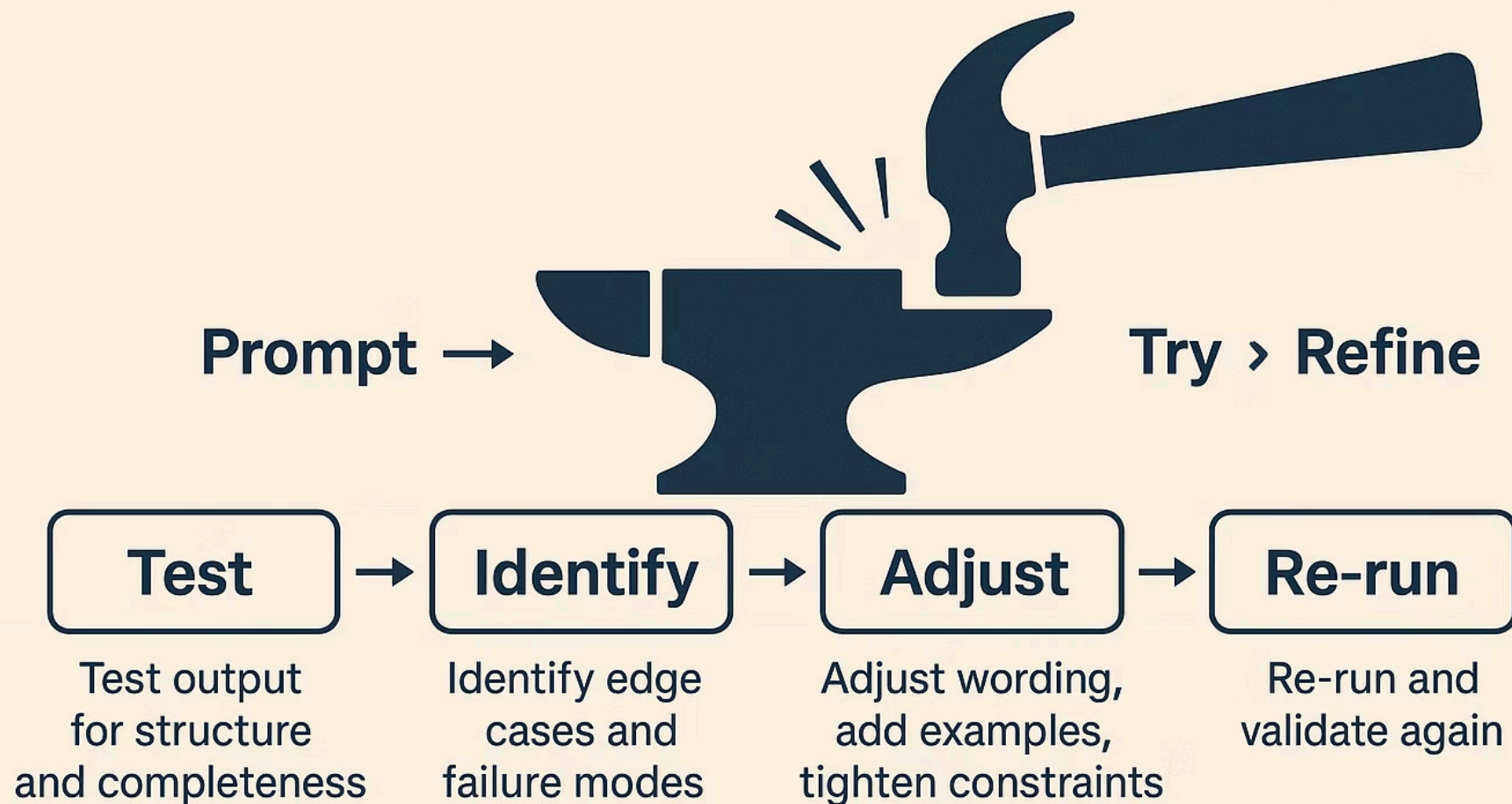
Output: test(order-api): add coverage for GET and POST endpoints

Now generate PR title for: Refactored Kafka consumer to handle batch events

Golden Rule 5 – Chain of Thought

Prompt → Try → Refine

Golden Rule 5 – Chain of Thought



Best Practice: Always test prompts with representative inputs before integrating into CI or production flows



Live Demo – Before & After (Legacy Code)

Task: Analyze a legacy .NET Framework application error

Goal:

Show how applying all 5 rules transforms an unusable prompt into a production-grade diagnostic tool.

Before (Breaking All 5 Rules):

✖

Broken Rule: Ambiguous Request

"Fix this code problem"

(No role, context, constraints, examples, or clarity)

1

Clarity

Specific task with defined output format

2

Context

System specs, architecture, and constraints

3

Role

Expert .NET architect with legacy focus

4

Few-Shot Examples

Example analysis with input/output format

5

Test Ready

Structured for validation and iteration



After (Following All 5 Rules)

Production-Grade Prompt:

You are a senior .NET architect specializing in legacy enterprise application modernization.

This legacy .NET Framework 4.5 application handles insurance claim processing (50K+ claims daily). The system crashes during peak hours with `OutOfMemoryException` in `ClaimProcessor.ProcessBatch()`.

Context:

- Monolithic architecture, 800K+ lines of code
- SQL Server 2012 backend with 500GB+ data
- Memory: 8GB heap, no garbage collection tuning
- Peak load: 200 concurrent users, 9–11 AM EST

Example format:

Input: StackTrace showing `OutOfMemoryException` in data processing loop

Output: Root cause (unbounded collections), specific fix (implement `IDisposable` pattern), estimated impact (70% memory reduction)

Now analyze this stack trace and provide structured remediation.



Prompt Pattern Toolkit – Big 3 Patterns

Choose the Right Structure for the Task

Zero-Shot

Direct instruction without examples

- Simple, clear tasks
- When format isn't complex
- Quick answers with minimal setup

Few-Shot

Input-output samples to guide formatting

- Consistent formatting needs
- Parse or transform data
- When precision matters

Chain of Thought

Step-by-step reasoning to break down complexity

- Complex analysis
- Multi-step problems
- Debugging and troubleshooting



Pattern Comparison – Same Task, Three Approaches

Challenge: Analyze this production incident log for root cause

2024-07-27 14:32:15 ERROR PaymentService: Connection timeout to database
2024-07-27 14:32:16 WARN LoadBalancer: Health check failed for payment-db-01
2024-07-27 14:32:17 ERROR PaymentService: Retrying connection (attempt 2/3)
2024-07-27 14:32:20 CRITICAL PaymentService: All database connections exhausted
2024-07-27 14:32:21 ERROR APIGateway: 503 Service Unavailable returned to client

Zero-Shot

"Analyze this production log for root cause and recommend fixes"

Few-Shot

Add example-based structure showing cause → fix

Chain of Thought

Prompt for step-by-step analysis with rationale

Pattern Comparison – Same Log, Three Prompts

Prompting Styles Applied to the Same Log

Zero-Shot

Prompt: "Analyze this production log for root cause and recommend fixes."

Result: "The logs show database connectivity issues. Consider checking network connectivity and restarting services."

Issues:

- Generic advice
- Missed cascading events
- No specific recommendations

Few-Shot

Prompt: Example-based analysis, showing structure and root cause/fix pairs.

Result: "ROOT CAUSE: Connection pool exhaustion after health check failure. FIX: Check payment-db-01, increase timeout, scale pool."


Improvements:

- More targeted
- Actionable next steps
- Follows learned pattern

Chain of Thought

Prompt: Step-by-step prompt guiding through timeline, cause, and fix.

Result (Preview): Timeline and cascade mapped clearly, root cause pinpointed, multilayer fix plan provided.

 **Strength:** Deep reasoning for accurate remediation

👉 See next slide for full Chain of Thought breakdown



Deep Dive – Chain of Thought Wins

Chain of Thought (CoT) prompting significantly enhances LLM's ability to perform complex root cause analysis by guiding it through a structured reasoning process.

Problem Analysis:

By prompting the LLM to reconstruct the timeline and map cascade events, CoT ensures a thorough understanding of the incident's progression.

- **Timeline Reconstruction:**
14:32:15 – DB timeout
14:32:16 – Health check fail
14:32:17 – Retry begins
14:32:20 – Connection pool exhausted
14:32:21 – 503 returned to client
- **Cascade Mapping:** Connection issue → Health check failure → Pool exhaustion → Outage
- **Root Cause:** payment-db-01 became unresponsive, leading to rapid connection depletion during failover.

Actionable Fix Plan:



Immediate Fix

Check DB server status (disk, CPU, memory) for payment-db-01.



Short-Term Fix

Implement a circuit breaker for faster failure detection and graceful degradation.



Long-Term Strategy

Set up continuous monitoring for connection pools and enable auto-scaling based on load.



✓ Why Chain of Thought excels:

- Handles real-world complexity effectively.
- Minimizes oversimplified or incorrect diagnoses.
- Builds trust for critical production environments.



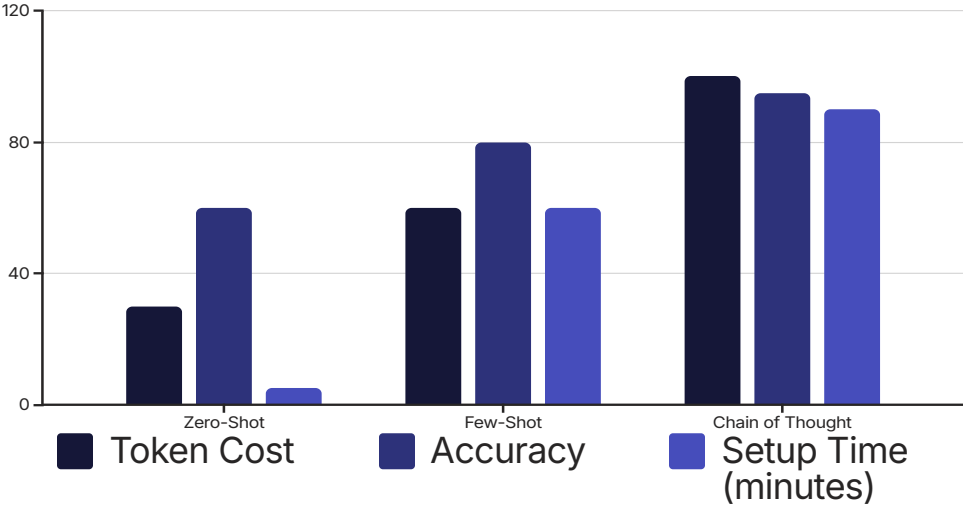
Pattern Selection Decision Tree

Choosing the Right Prompting Strategy

Task-Based Recommendations

◆ Simple API documentation	Zero-Shot	Direct, fast, minimal setup
◆ Company log format parsing	Few-Shot	Learns from your exact format
◆ Standard error extraction	Zero-Shot	One-step instructions are enough
◆ Custom error categorization	Few-Shot	Pattern-based decisions needed
▲ Root cause investigation	Chain of Thought	Requires deep reasoning and timelines
▲ System-wide bug tracing	Chain of Thought	Handles cascading failures well

Performance Comparison



Higher setup time and token cost often correlate with better accuracy for complex tasks.



Prompt Pattern Lab Preview

Hands-on Preview: How You'll Use This

Lab Focus & Task:

Fix a broken prompt and validate structured output in CI/CD. The task involves using a poorly performing prompt to return a weather API JSON object.

Requirements:

- Input: "Get weather for Delhi tomorrow"
- Output must match this schema:

```
{
  "location": "Delhi",
  "forecast": {
    "date": "2024-08-01",
    "tempC": 34,
    "condition": "Sunny"
  }
}
```

Prompt Fix Cycle:

Observe

Observe malformed JSON or vague result

Apply



Apply 5 Golden Rules (clarity, format, role)

Use

Use few-shot schema correction

Re-test

Re-test and iterate until CI passes

  **Outcome:** Predictable JSON → testable output → real dev utility



Prompt Training Lab: Ready-to-Play Prompts

Dive Deeper into Hands-On Prompt Engineering

Ready to put your prompt engineering skills to the test? Our Prompt Training Lab provides a collection of ready-to-use prompts and scenarios designed to help you practice and evaluate different prompting strategies.

Explore the lab and start experimenting with practical examples:

[Access the GitHub Prompt Library Here](#)

© 2025 TheDBAdmin.com | All rights reserved



Setting Up Your AI Workspace

Prerequisite Checklist



API keys (OpenAI/Anthropic) & environment variables

```
export OPENAI_API_KEY=sk-  
abcd...  
export  
ANTHROPIC_API_KEY=sk-ant-...
```



IDE plugins installation

- VS Code Copilot
- Claude Code extension
- GitHub Copilot Chat



HTTP client access

- Postman collections
- cURL/HTTPIe commands
- Python requests library



Data Privacy & Workspace Integrity



Sanitise PII in Prompts

Always remove personally identifiable information before sending to LLMs:

- User IDs, names, email addresses
- Database connection strings
- Authentication tokens



Use Synthetic/Minted Data

When testing on private code:

- Generate realistic but fake data
- Replace real customer info with fabricated examples
- Use tools like Faker libraries to create test datasets



Region-specific Compliance

Be aware of data protection regulations:

- GDPR (Europe) - strict consent requirements
- CCPA (California) - consumer rights focused
- India Data Protection Bill - data localisation rules

Custom Connector & ChatGPT Plugin Setup



Define your connector

Create an OpenAI/ChatGPT Plugin by building an ai-plugin.json manifest that points to your API



Register & Enable

In ChatGPT → Settings → Plugins → "Develop your own plugin" → Upload manifest



Test inside ChatGPT

Chat with your plugin: "Fetch latest PRs from GitHub repo X"



Iterate on manifest

Refine endpoints until it works end-to-end

CLI Quick-Start Example

1. Install & login

```
npm install -g openai
```

```
openai login
```

2. Register your plugin

```
openai plugins register --name github-connector --manifest https://your-domain.com/.well-known/ai-plugin.json
```

3. Call ChatGPT with that plugin

```
openai chat --model gpt-4o --plugins github-connector --prompt "List open issues in repo acme/project"
```



Optimum ChatGPT Setup for VS Code

Why Optimize ChatGPT in VS Code?



Boost Productivity

Enhance coding efficiency with AI-driven suggestions and intelligent completions.



Enforce Safety Rules

Implement and adhere to production environment safety rules directly within your IDE.



Seamless Integration

Integrate ChatGPT directly into your development workflow, eliminating context switching.



Minimize Repetition

Reduce repetitive prompts and manual information lookups for a smoother experience.



Extension Setup

Select & Install Extension

Choose and install one of the recommended VS Code extensions:

- **Code GPT (Daniel San)**
- **ChatGPT - Genie AI**

Find in VS Code

Open VS Code, navigate to Extensions (Ctrl+Shift+X), and search for *ChatGPT*.

Configure API Key

Install your selected extension and configure your **OpenAI API Key**. Ensure it is stored securely as an environment variable.



Pro Tip: Secure Your Keys

Never hardcode API keys directly into your repository. Always use environment variables or a secure key management system for sensitive credentials.



ChatGPT Settings Optimization

Custom Instructions

Role: *Senior DevOps/DBA Mentor*

Traits:

- Avoid destructive commands
(`rm -rf`, `DROP DATABASE`)
- Always provide rollback and backup steps.
- Prefer KISS coding principles.
- Consider API costs and optimization.
- Suggest edge case test coverage.

Temperature

Low (0.2) for precise answers, reducing creative or unexpected outputs.

Structured Output

Prefer outputs as bullet points, clear code snippets, and step-by-step instructions for clarity and ease of use.



Privacy & Connectors

Data Privacy & Workspace Integrity

- Disable *training on chats* in OpenAI Data Controls.
- Use **Temporary Chat** for sensitive requests.
- Avoid pasting secrets or production DB names.

Custom Connector & ChatGPT Plugin Setup

- GitHub, Google Drive → enable read-only access only.
- Review connector permissions before use.



Agent Setup in VS Code

Install Python or Node.js local agent scripts to automate tasks:

```
python agent.py --task "Summarize this folder and insert purpose comments"
```

Copy

Edit



Analyze Folder Structures



Generate Documentation



Provide Safe Automation Scripts

Test Prompt:

You are a cautious DevOps mentor in VS Code. I want to clean old logs from production servers. Suggest safe steps, highlight risks, rollback plan.

Copy

Edit

Expected response: No destructive commands, includes safe and rollback instructions.

Validate setup with a test prompt before real usage.



Key Takeaways

Combine VS Code extensions + optimized ChatGPT settings for best results.

Enforce safety and cost-awareness via traits.

Use connectors carefully and disable chat training.



LABs

© 2025 TheDBAdmin.com | All rights reserved