

High-Level Architecture (Explain First)

Say this clearly:

“Before looking at code, let’s understand how data flows inside our AI application.”

Architecture Flow

PDF File

↓

Python PDF Reader

↓

Clean Extracted Text

↓

AI Model (ChatGPT-5 Mini)

↓

Generated Summary

↓

Output (Console / File / Email)

 **Audience interaction question:**

“Which step do you think takes the most time — PDF reading or AI processing?”

Internal Working – Step by Step (Deep but Simple)

Environment & Security Layer

```
load_dotenv()
```

```
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
```

Explain like this:

- .env file stores **secret credentials**
- Python loads it into memory
- API key is **never hardcoded**
- This is how **real production systems** work

💬 Ask audience:

“Why do you think we don’t put API keys directly in code?”

(Expected answer: security)

2 PDF Processing Layer (Data Ingestion)

```
reader = PdfReader(file_path)
```

Internal working:

- PDF is split into **pages**
- Each page contains **raw text objects**
- Python extracts text line by line
- Text is **unstructured** at this stage

```
for page in reader.pages:
```

```
    text += page.extract_text()
```

Key teaching point:

“At this stage, AI is NOT involved. This is pure Python processing.”

💬 Interaction:

“What happens if the PDF is scanned and not text-based?”

(Answer: text extraction fails → need OCR)

3 Text Preprocessing Layer (Cost Control)

```
text = text[:4000]
```

Explain this **very well** (important):

- AI models charge **per token**
- Large PDFs = higher cost
- We limit input to control:
 - Cost 💰
 - Speed ⚡

- API stability

💬 Ask:

“Would you prefer full summary or fast & cheap summary?”

4 AI Interaction Layer (Core Intelligence)

```
client.responses.create(  
    model="gpt-5-mini",  
    input="Summarize the following text..."  
)
```

Internal working (VERY IMPORTANT SECTION):

Explain this conceptually 🙋

1. Text is converted into **tokens**
2. Tokens are sent to OpenAI servers
3. Model predicts **next best words**
4. AI generates **human-like summary**
5. Response is returned to Python

💬 Powerful line:

“We are not calling magic — we are calling a probability-based language model.”

5 Response Handling Layer

```
response.output_text
```

Explain:

- AI response is **structured JSON**
- `output_text` extracts **final readable content**
- This abstraction makes coding simpler

💬 Ask audience:

“Should we store this summary in a file or send it by email next?”

(Tease next video)

Output Layer (Presentation)

```
print(summary)
```

Explain:

- Currently output → terminal
- But architecture supports:
 - File save
 - Email
 - Web app
 - API

 Line:

“Same AI logic, multiple applications.”

Code Architecture (Explain Professionally)

Explain Modules:

Layer	Responsibility
Config	API keys & env
Data Ingestion	Read PDF
Processing	Clean & limit text
AI Engine	Summarization
Output	Display / Save

Say:

“This separation makes code scalable and testable.”

Enhanced Functionalities You Can Show (Optional Demos)

Add File Output

```
with open("summary.txt", "w", encoding="utf-8") as f:
```

```
    f.write(summary)
```

2 Handle Empty PDFs

```
if not text.strip():
```

```
    raise ValueError("PDF contains no readable text")
```

3 Custom Summary Type

```
input=f"Summarize in bullet points:\n{text}"
```

💬 Ask:

“Bullet points or paragraph summary — which do you prefer?”

How to Interact Throughout the Video

Use these hooks 🙋

- “Pause here and think...”
 - “Comment below what you would automate next”
 - “This is how real AI systems work in companies”
-

Strong Ending Statement

Say this confidently:

“Today we didn’t just write code — we understood how AI systems think, process, and respond.”