Prompt Engineering

A Complete Guide for Beginners

Imagine working with a **super-smart intern**. They can write,

summarize, code, and explain anything —

but only if you tell them exactly what you want.

That's how Large Language Models work, and that's why *Prompt Engineering* is so important.

Introduction to Prompt Engineering for Developers

What You'll Learn?

- Understand **what Prompt Engineering is** and why it matters for developers.
- "Prompt Engineering is not about coding — it's about **thinking clearly** and **communicating precisely** with an LLM.
- Learn **best practices** for crafting clear, specific, and structured prompts.
- Explore **four key LLM use cases**:
  - Summarizing
  - Inferring
  - Transforming
  - Expanding
- Building chatbots
- Get hands-on with **OpenAI API calls** and integrate LLMs into your own apps.

Two Types of Large Language Models(LLMs)

- Base LLM
- Instruction Tuned LLM

Base LLM:

- **What it is:**

•Trained on large amounts of text

•Learns **patterns of language**

•Predicts the **next token (word piece)**

- **What it can do:**

•Continue stories

•Generate fluent text

•Mimic writing styles

- **Example Prompt:**

Once upon a time, there was a unicorn…

- **Example Output:**

…that lived in a magical forest with all her unicorn friends.

- **Limitation:**
- Does NOT reliably follow instructions
- May not give direct answers to questions
- **Training Method:**
- Self-supervised learning on raw text

Instruction-Tuned LLM

- **What it is:**
- A Base LLM that is further trained to **follow instructions**
- Optimized for helpfulness, safety, and accuracy
- **What it can do:**
- Answer questions
- Summarize text
- Translate
- Reason step-by-step

- **Example Prompt:**
  - What is the capital of France?
- **Example Output:**
  - The capital of France is Paris.
- **Advantage:**
  - Follows user intent
  - Gives structured, task-oriented responses
- **Training Method:**
  - Fine-tuning on instruction–response pairs
  - Further improved using Reinforcement Learning with Human Feedback (RLHF)

Key Difference:

- Base LLMs complete text.
- Instruction-Tuned LLMs complete tasks.
- **Why ChatGPT feels “smart”:**
- ChatGPT is an **instruction-tuned LLM**, not just a base LLM.

So the Base LLM is like a language pattern generator — it completes text.

But the Instruction-Tuned LLM is like an intelligent assistant — it follows what you *ask it to do*.

This evolution makes prompt engineering far more reliable for developers.

Learning Goals

- By the end of this session, you will be able to:
- Craft effective prompts for various NLP tasks.
- Understand how instruction-tuned LLMs differ from base models.
- Apply prompting best practices to **real-world software**.
- Build your own **mini chatbot** using the OpenAI API.

Guidelines for Prompting

- **Objective**
- Learn two key principles for effective prompt engineering with LLMs:
- **Write clear and specific instructions**
- **Give the model time to think**

Principle 1: Write Clear and Specific Instructions

“Clear doesn't mean short”
“Be explicit, structured, and contextual — clarity beats brevity.”

Tactic 1:Use Delimiters

- A **delimiter** is a **clear boundary marker** you use in a prompt to **separate different parts of input**, so the model knows **what belongs where.**
- **Eg: Triple quotes : ”””,**
- **Triple backticks : ”’ ,**
- **Triple dashes: ---,**
- **Angle brackets: < >**
- **XML tags: <tag>, </tag>**

- **prompt = f"""Summarize the text delimited by triple backticks \ into a single sentence. ```{text}``` """**
- If your prompt is more than 3 lines, use structured formatting — lists, bullets, or JSON — to make it readable for both humans and LLMs.

Tactic 2: Ask for Structured Output: JSON, HTML

prompt = f"""

Generate a list of three made-up book titles along \ with their authors and genres. Provide them in JSON format with the following keys: book_id, title, author, genre.

"""

Tactic3: Check Conditions Before Executing

prompt = f"""

You will be provided with text delimited by triple quotes. If it contains a sequence of instructions, \ re-write those instructions in the following format:

Step 1 - ...

Step 2 - ……

Step N - …

If the text does not contain a sequence of instructions, \ then simply write \"No steps provided.\" \"\"\"{text_1}\"\"\"

"""

Tactic 4: Use Few-Shot Prompting

- Give successful examples of completing tasks
- prompt = f"""

Your task is to answer in a consistent style.

<child>: Teach me about patience.

<grandparent>: The river that carves the deepest \ valley flows from a modest spring; the \ grandest symphony originates from a single note; \ the most intricate tapestry begins with a solitary thread.

<child>: Teach me about resilience.

"""

Principle 2: Give the model time to “think”

Tactic 1: Specify the steps required to complete a task

```
prompt_1 = f"""
Perform the following actions:
1 - Summarize the following text delimited by triple \ backticks with 1 sentence.
2 - Translate the summary into French.
3 - List each name in the French summary.
4 - Output a json object that contains the following \keys: french_summary,
num_names.
Separate your answers with line breaks.
Text:
```{text}```
"""
```
```

Ask for output in a specified format

```
prompt_2 = f"""
Your task is to perform the following actions:
1 - Summarize the following text delimited by   <> with 1 sentence.
2 - Translate the summary into French.
3 - List each name in the French summary.
4 - Output a json object that contains the   following keys: french_summary, num_names.
Use the following format:
Text: <text to summarize>
Summary: <summary>
Translation: <summary translation>
Names: <list of names in summary>
Output JSON: <json with summary and num_names>
Text: <{text}>
"""
```

Tactic 2: Instruct the model to work out its own solution before rushing to a conclusion

- prompt = f""“

Determine if the student's solution is correct or not.

Question: I'm building a solar power installation and I need \ help working out the financials.

- Land costs $100 / square foot.
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost \ me a flat $100k per year, and an additional $10 / square \foot.

What is the total cost for the first year of operations as a function of the number of square feet.

……. Continue on next slide……..

Student's Solution:

Let x be the size of the installation in square feet.Costs:

1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x

Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000

"""

- **Note that the student's solution is actually not correct.**
- **We can fix this by instructing the model to work out its own solution first**.

Model Limitation: Hallucination

- LLMs don’t ‘know’ facts; they generate likely text based on patterns — always verify factual outputs

From Old to New: The Shift in LLM Challenges

Era	Key Challenge	Status
GPT-3 (2020)	Hallucination	❌ Major issue
GPT-4 (2023)	Factual grounding	⚙️ Improved
GPT-5 (2025)	Deep reasoning, memory, bias	⚠️ Still limited

Current Technical Limitations of GPT-5 and Modern LLMs

Limitation	🔍 Description	💡 Impact
1. Reasoning Gaps	Can mimic logic but not causal reasoning.	Struggles with physics or chain-of-cause tasks.
2. Weak World Model	No real concept of time or continuity.	Forgets past events or updates.
3. Numeric Fragility	Not built for precise math.	Errors in long calculations.
4. Context Window ≠ Memory	Large context, but no persistence.	Forgetful across sessions.
5. Bias & Cultural Framing	Data still language-region heavy.	Subtle bias in tone or viewpoint.

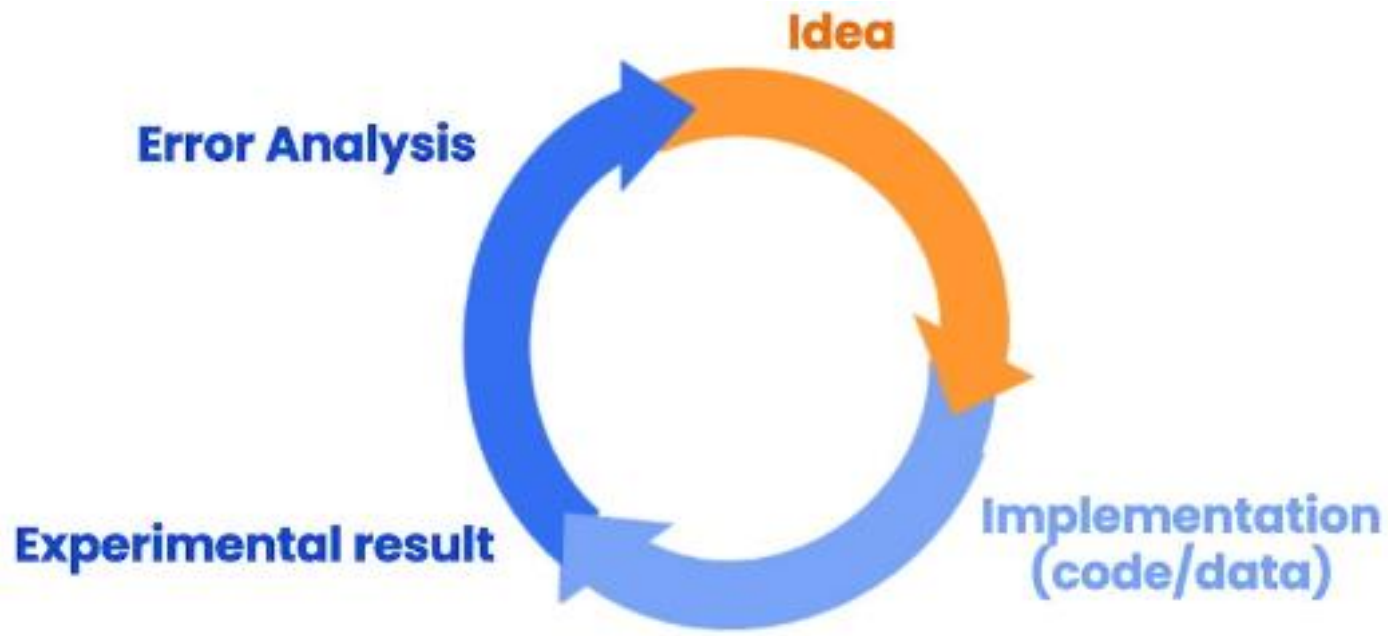
Emerging Challenges: What's Next to Fix

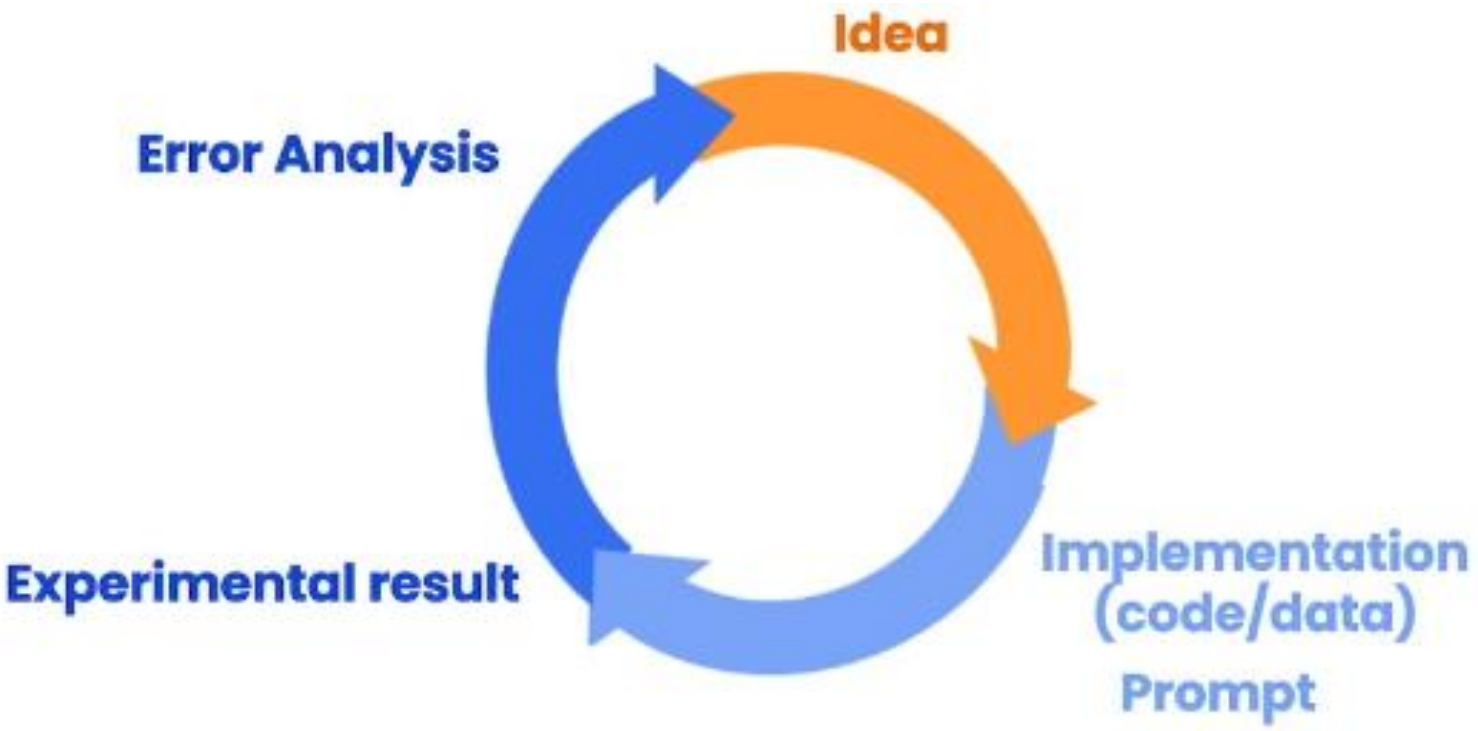
Challenge	Explanation	Research Direction
Dependence on Retrieval	Truth tied to data source reliability.	Source-quality filtering, evidence scoring.
Compute & Energy Cost	Verification adds heavy load.	Efficient architectures, sparse attention.
Limited Common Sense	Literal interpretation of human nuance.	Multimodal + emotional context fusion.
Ethical & Privacy Boundaries	Live web data risks personal leaks.	Regulated data governance (DPDP, AI Act).

Iterative Prompt Development

Think → Try → Observe → Refine → Repeat

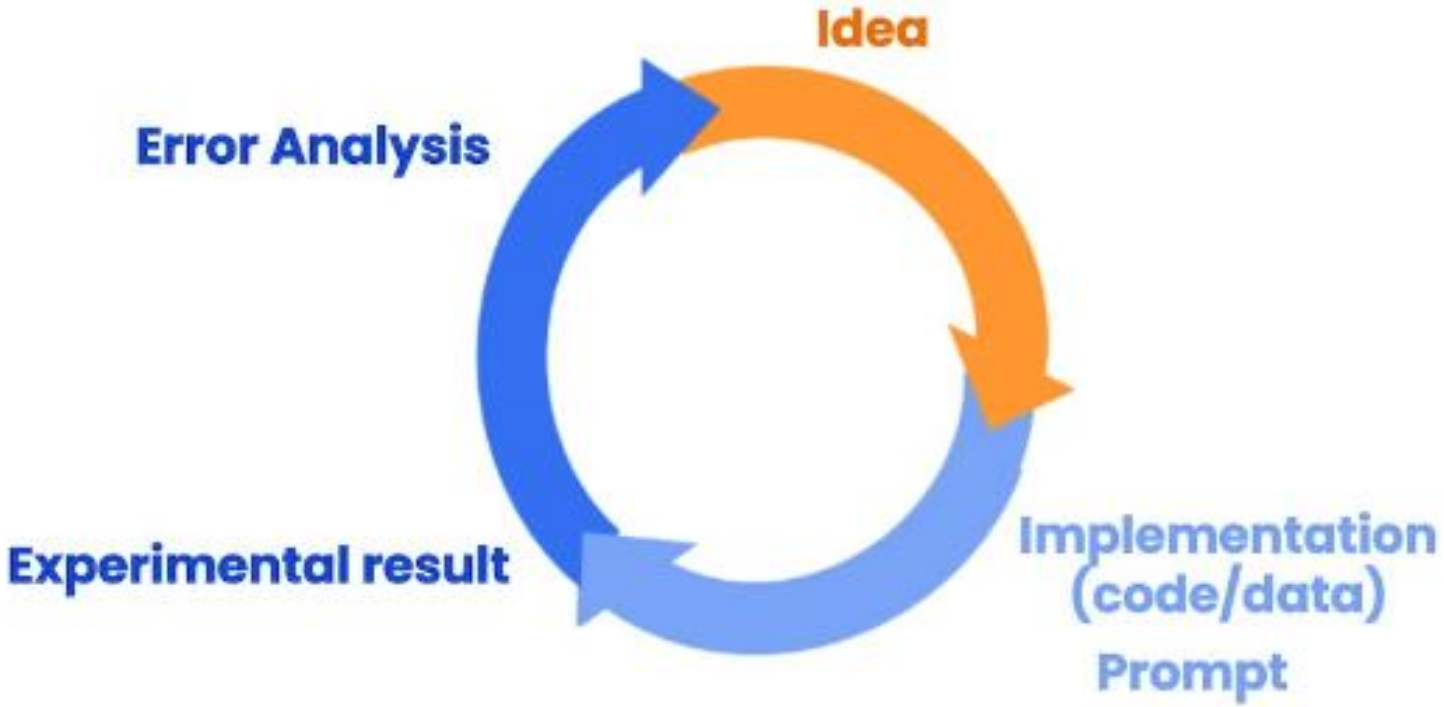
Iterative Prompt Development





Prompt guidelines

- Be clear and specific
- Analyze why result does not give desired output.
- Refine the idea and the prompt
- Repeat



Iterative Process

- Try something
- Analyze where the result does not give what you want
- Clarify instructions, give more time to think
- Refine prompts with a batch of examples

Summarizing

Summarizing the Text

- LLMs can **read long or multiple pieces of text** and generate a **condensed summary** that captures:
- Main ideas
- Sentiment (positive, neutral, negative)
- Key features mentioned by users
- Pros and cons

Why Summarization Matters

- In today's world, there's **too much text** and **too little time**.
- Summarization using **Large Language Models (LLMs)** helps condense lengthy text into key ideas.
- This is one of the **most useful and practical applications** of LLMs today.
- It's now built into many tools — including **ChatGPT**, browsers, and enterprise software.
- *Use ChatGPT to summarize articles, reports, or long documents instantly.*

Summarization in ChatGPT

- You can directly paste long text into ChatGPT and ask:
- “Summarize this article in 5 bullet points.”
- Great for quickly reading more content in less time.
- Works both **interactively (in chat)** and **programmatically (via API)**.

E-Commerce Example — Product Review Summary

- **Use Case:** Summarizing long product reviews for quick insights.
- **Example Review:**
- “Got this panda plush toy for my daughter’s birthday. She loves it and takes it everywhere. A bit small for the price but arrived early.”
- **Prompt:**
- Your task is to generate a short summary of a product review from an e-commerce website. Summarize in at most 30 words.
- **Output Example:**
- “Soft, cute panda plush toy loved by daughter. Small for the price but arrived early.”

Customizing Summaries by Purpose

Department	Focus	Example Summary
Shipping Dept.	Delivery speed	“Product arrived earlier than expected.”
Pricing Dept.	Value perception	“Cute toy but small for the price.”
Product Team	Quality & design	“Soft and loved by children, slightly undersized.”

Targeted Summarization Prompt

- Summarize the following review focusing on aspects related to shipping and delivery.
- Summarize the following review focusing on price and perceived value.

Extraction vs. Summarization

Task	Description	Example Output
Summarize	Condense overall review	“Soft, cute toy. A bit small but arrived early.”
Extract	Pull out only specific info	“Arrived earlier than expected.”

Multi-Review Summarization

- **Use Case:** Summarizing hundreds of reviews for a dashboard view.
- Collect all reviews into a list.
- Loop through each review.
- Summarize each in 20–30 words.
- Display summaries side by side for easy browsing.
- **Result:**

  A compact table or dashboard where every long review becomes a one-line insight.
- Helps you or your team quickly scan and understand customer sentiment.

Example Output – Multiple Reviews

Product	Summary
Panda Plush Toy	“Soft, cute, small for price, arrived early.”
Standing Lamp	“Bright and modern, fits well in bedroom.”
Electric Toothbrush	“Cleans well, recommended by dentist, slightly noisy.”
Blender	“Powerful, blends smoothly, good deal on sale.”

- Managers can now review 100+ feedbacks in minutes.

Why Summarization Is Powerful

- Saves **time** — read insights, not paragraphs.
- Increases **understanding** — focuses on main ideas.
- Enables **analysis** — can be combined with sentiment or topic inference.
- Useful for:
  - E-commerce feedback
  - News articles
  - Research papers
  - Meeting notes
  - Customer support logs

Prompt Design Tips for Summarization

- Be specific about output size and format:
- “Summarize in 3 bullet points / 30 words / 2 sentences.”
- Define the focus area (quality, price, shipping, tone, etc.)
- Use delimiters for input clarity:
- """ [Paste your text here] """
- For large data → summarize in **chunks**, then summarize the summaries (hierarchical summarization).

Key Takeaways

Concept	Description
Summarization	Condenses large text into key ideas.
Context-Specific Summaries	Tailored to purpose or department.
Extraction	Retrieves only facts or data points.
Automation	Multiple reviews summarized programmatically.
Business Impact	Faster insights, smarter decisions, improved productivity.

💡 *If you deal with long text, ChatGPT can be your "summary assistant."*

Real-Life Application Ideas

- **Teachers:** Summarize research papers or class readings.
- **Managers:** Summarize meeting transcripts.
- **Marketers:** Summarize customer reviews or campaign feedback.
- **Students:** Summarize lecture notes or articles.
- **Writers:** Summarize references for quick recall.

Sample Prompt Templates

Purpose	Prompt
Quick overview	“Summarize in 5 bullet points.”
Custom department	“Summarize focusing on delivery issues.”
Extract key fact	“Extract only price-related comments.”
Limit length	“Summarize in under 30 words.”
Comparative summary	“Summarize differences between these two reviews.”

Inferring:

Inference = Understanding Beyond Words

What Is Inferring?

- **Definition:**

  Inference means using ChatGPT to **analyze text** and draw **meaning, emotion, or structured information** from it.

- **Purpose:**

  To *understand* what the text implies — not just what it says.

Examples of Inference Tasks:

- Sentiment analysis (positive / negative)
- Emotion detection (anger, joy, surprise)
- Topic identification
- Name or entity extraction
- Detecting missing parts or issues
- 💡 *Inference = "Let ChatGPT tell you what's going on inside the text."*

Traditional ML vs. LLM-Based Inference

Traditional Machine Learning	ChatGPT / LLMs
Collect labeled dataset	No dataset needed
Train & deploy separate models	One model handles all tasks
Each task = new model (sentiment, entity, etc.)	One prompt = many inferences
Requires coding + cloud setup	Works instantly with a prompt

Example – Sentiment Analysis

- **Task:** Identify if a review is positive or negative.
- **Prompt:**
- What is the sentiment of the following product review? """ Needed a nice lamp for the bedroom and this one had additional storage. Great company that cares about customers. """
- **Output:**
- “The sentiment of the product review is **positive**.”
- ***ChatGPT immediately infers sentiment without training any model.***

Concise Output for Automation

- Sometimes, you need **simple, process able outputs** for dashboards or Excel.
- **Prompt:**
- What is the sentiment (positive/negative) of the following review? """ Needed a nice lamp for the bedroom and this one had additional storage. """
- **Output:**
- **Positive**
- ✅ Easier for business tools or spreadsheets to process automatically.

Emotion Extraction

- **Prompt:**
- Identify a list of emotions that the writer is expressing. Include no more than 5 items. """ Needed a nice lamp for the bedroom and this one had additional storage. The light is warm and cozy. """
- **Output Example:**
- ["Happiness", "Satisfaction", "Comfort", "Appreciation"]
- 💡 *Use this to measure customer emotion or tone in feedback.*

Anger or Urgency Detection

- **Use Case:** Customer Support Prioritization
- **Prompt:**
- Is the writer of the following review expressing anger? Answer Yes or No. """ The product arrived late and the box was damaged! """
- **Output:**
- **Yes**
- *Helps customer-care teams identify high-priority complaints instantly.*

Information Extraction

- **Goal:** Extract key data (entities, product names, brands, etc.)
- **Prompt:**
- Identify the following from the review: 1. Item purchased 2. Company that made it Return output as JSON. """ I bought a lamp with additional storage from Lumina. It's stylish and affordable. """
- **Output:**
- { "Item": "lamp with additional storage", "Brand": "Lumina" }
- *Perfect for structured data extraction without coding.*

Multiple Inferences in One Prompt

- **Prompt:**
- Extract the following details: 1. Sentiment (Positive/Negative)
- 2. Is the customer angry? (true/false)
- 3. Item purchased
- 4. Brand name Return as JSON. """ I bought a Lumina lamp with extra storage. Works perfectly and arrived early. """
- **Output:**
- { "sentiment": "positive", "anger": false, "item": "lamp with extra storage", "brand": "Lumina" }
- 💡 *One prompt → multiple analyses at once.*

Topic Inference (Text Classification)

- **Task:** Identify what topics a text is about.
- **Prompt:**
- List 5 topics discussed in the text below. Make each topic 1–2 words long, separated by commas. """ A government survey revealed high job satisfaction among NASA employees working on space projects. """
- **Output:**
- "NASA, job satisfaction, government survey, space projects, employee morale"
- *ChatGPT can detect subjects and themes instantly.*

Zero-Shot Topic Detection

- **Prompt:**
- Given these topics: ["NASA", "local government", "engineering", "employee satisfaction", "federal government"] Determine which topics appear in the article below. Output 1 for present, 0 for absent. """ A government report praised NASA for high employee satisfaction and federal project success. """
- **Output Example:**
- [1, 0, 0, 1, 1]
- 💡 *Zero-shot learning:* ChatGPT infers topics **without any pre-labeled data.**

From Inference to Automation

- **You Can Build:**
- News alert system → “Alert: New NASA Story!”
- Customer insight dashboards → Track product sentiment
- HR tools → Detect employee tone in survey text
- Market research reports → Extract brand names and issues
- All powered by **simple prompts**, not ML pipelines.

Best Practices for Inference Prompts

- Be clear about what to infer (tone, topic, emotion, etc.)
- Request specific formats (JSON, list, 0/1)
- Add limits (“no more than 5 emotions”)
- Include examples for complex tasks
- Avoid over-inference (“If not mentioned, reply Unknown”)
- 💡 *Precise prompts = reliable inferences.*

Why Inference Is Powerful

Benefit	Description
Instant insight	Understand tone, topic, and data without coding
Multi-purpose	Same model can classify, extract, and analyze
Zero-shot	No labeled data needed
Fast deployment	Ready in minutes, not weeks
Integrates easily	Outputs can feed dashboards, Excel, or chatbots

Real-Life Use Cases (Non-Coders)

- **Teachers:** Detect sentiment in student feedback
- **Managers:** Understand tone of employee comments
- **Marketers:** Identify customer pain points
- **Support Teams:** Prioritize angry or urgent messages
- **Analysts:** Extract topics from news or reports

Key Takeaways

Concept	Description
Inference	Understanding meaning, tone, or data from text
Sentiment Detection	Positive / Negative / Neutral
Emotion Analysis	Anger, joy, satisfaction, etc.
Information Extraction	Product, brand, location
Topic Detection	What the text is about
Zero-Shot Inference	Works instantly without training data

💡 *Inference turns ChatGPT into your personal text analyst.*

Transformation with LLMs (ChatGPT)

How to rewrite, translate, reformat and proofread text using prompts — no coding required.

What is Transformation?

- **Definition:** Converting input text into a different *form*, *style*, or *format* while preserving meaning.
- **Examples:** Translation, tone change, grammar correction, format conversion (JSON→HTML), role-based rewrites.
- **Why it matters:** Saves time, removes tedious regex/ETL work, improves communication quality.

Core Transformation Capabilities

- **Translation** (many languages, formal/informal variants)
- **Tone & style transfer** (slang → business letter, casual → formal)
- **Format conversion** (JSON ↔ HTML, Markdown, CSV)
- **Proofreading & grammar correction**
- **Content reframing** (marketing copy, academic style, bullet → paragraph)

Example: Simple Translation

- **Prompt template**
- Translate the following English text to Spanish: """Hi, I would like to order a blender."""
- **Example output**
- Hola, me gustaría ordenar una licuadora.
- **Teaching note:** LLMs know many languages; useful for global support & localization.

Language Detection + Multi-target Translation

- **Use case:** Universal translator for multilingual user messages.
  Workflow (no-code):
- Ask: “What language is this?”
- Ask: “Translate it to English and Korean.”
- Loop over messages and present translations.
- **Prompt tip:** Ask for single-word language names or JSON for easier parsing.
- Where **Parsing is the process of converting unstructured or semi-structured text into structured data that a program can process**

Formal vs. Informal Translations

- **Prompt**
- Translate the following to Spanish in formal and informal forms: "Would you like to order a pillow?"
- **Output examples**
- Formal: “¿Le gustaría pedir una almohada?”
- Informal: “¿Te gustaría pedir una almohada?”
- **Teaching note:** Use when tone depends on audience (customers vs. friends).

Tone Transformation (Slang → Business)

- **Prompt**
- Translate this slang to a business letter: "Dude, this is Joe, check out this spec on the standing lamp."
- **Result:** A professional business-style paragraph suitable for email.
- **Use-cases:** Customer-facing replies, internal memos, professional outreach

Format Conversion (JSON → HTML)

- **Prompt**
- Translate the following JSON (employee list) into an HTML table with headers and title.
- **Result:** Fully-formed <table> HTML you can render in a page or notebook.
- **Teaching note:** Great for quick UI previews, admin dashboards, documentation.

Proofreading & Grammar Correction

- **Prompt templates**
- “Proofread and correct this sentence. If no errors, say ‘No errors found’.”
- “Proofread, correct, and rewrite the following paragraph in polished business English.”
- **Examples:** Fixes spelling, punctuation, grammar, sentence flow.
- **Tooling tip:** Show differences using diff/redlines packages to demonstrate edits.

Advanced Rewrite: Tone + Style + Format

- Proofread and correct; make it more compelling; follow APA style for advanced readers; return result in Markdown.
- Result: Expanded, APA-styled review in Markdown — useful for reports and publications.

- Teaching note: Combine multiple constraints in one prompt for high-value rewrites.
- Where **APA style** refers to a **standardized format for academic writing, citation, and referencing**, created by the **American Psychological Association (APA)**.

Practical Workshop Example (no-code)

- **Task:** Build a simple “translator + tone fixer” flow in ChatGPT:
- Paste user message.
- Ask ChatGPT: “Detect language (one word).”
- Ask: “Translate to English.”
- Ask: “Rewrite in polite customer-support tone.”
- **Outcome:** Localized, polite reply ready to send.

Prompt Patterns & Templates

- **Translate:** Translate the following to <language>: """..."""
- **Detect language:** What language is this? Respond with one word.
- **Tone change:** Rewrite in a <tone> tone for <audience>.
- **Format conversion:** Convert this JSON to an HTML table with headers.
- **Proofread:** Proofread and correct the following. If no errors, reply "No errors found".

Best Practices

- Be explicit about **target form** (language, tone, format).
- Use **delimiters** (""") for clarity.
- Ask for **structured output** (JSON, single-word, Markdown) so that processing the result becomes extremely easy for a computer.
- Combine instructions (proofread + style + format) but keep prompts readable.
- Iterate: if outputs are inconsistent, refine prompts or add examples.

Common Pitfalls & Fixes

- **Issue:** Model adds extra facts or changes meaning → **Fix:** “Do not add new facts.”
- **Issue:** Formatting inconsistent → **Fix:** “Return only JSON” or “Return only HTML.”
- **Issue:** Language detection returns a sentence → **Fix:** “Respond with one-word language name.”
- **Issue:** Rare languages / dialects perform worse → **Fix:** validate outputs or fallback to human review.

Summary

- Transformation = practical, high-impact use of ChatGPT.
- Covers translation, tone transfer, format conversion, proofreading, and more.
- Works with simple prompts — no coding necessary.
- Combine with Summarization, Inference, and Expansion to build powerful workflows.

Expanding:

Expansion = (Seed Idea + Context + Constraints + Temperature) → Draft

What Is Expanding?

- **Definition:**

  Expanding means taking a **short piece of text** (like an idea, instruction, or sentence)and using ChatGPT to **generate a longer, more detailed** piece of writing — such as an email, report, blog, or essay.

- **Examples:**
- Turning *bullet points → paragraph*
- Turning *a topic → blog post*
- Turning *a review → full email reply*
- 💡 *Think of expanding as moving from outline to full draft.*

Key Use Cases

Use Case	Description
Email generation	Write personalized replies or follow-ups
💡 **Brainstorming**	Generate ideas, examples, or variations
✍️ **Content creation**	Expand notes into blogs or essays
📈 **Report writing**	Turn data summaries into business insights
💬 **Training/education**	Create case studies or Q&A from topics

Example — Expanding a Customer Review

- **Task:**

  Generate an email reply to a product review.
- **Prompt:**
- You are a customer service AI assistant. Your task is to send an email reply to a valued customer. Given the customer review below, write a professional email: - If sentiment is positive → thank the customer. - If sentiment is negative → apologize and suggest contacting support. - Include details from the review. - Sign as "AI Customer Agent".

Output Example:

- “Dear Customer,

  Thank you for your review of our blender.

  We’re sorry to hear it was smaller than expected.

  Please reach out to our service team for assistance.

  Best regards,

  AI Customer Agent.”

Ethical Use of Expansion

- Use to:
- Personalize genuine communications
- Create helpful educational or marketing content
- 🚫 Avoid using to:
- Generate **spam** or **misleading** bulk content
- Impersonate humans — always **declare AI involvement**
- 🧩 *Transparency builds trust — e.g., "Message generated by AI Assistant."*

Personalization Through Expansion

- ChatGPT can include **specific details** from user input:
- Mentions product name
- Acknowledges review tone
- Suggests next steps
- **Prompt Tip:**
- “Make the response polite, concise, and reference details from the review.”
- *Good expansion = specific, relevant, and human-like tone.*

Introducing the Temperature Parameter

What is Temperature?

- A setting that controls **creativity** or **randomness** in LLM output.

Temperature	Behavior	Example Use
0.0 – 0.3	Deterministic, factual	Email replies, summaries
0.4 – 0.6	Balanced & natural	Explanations, Q&A
0.7 – 1.0	Creative, variable	Brainstorming, storytelling

Understanding Temperature (Example)

- **Complete: "My favorite food is"**

Temperature	Output
0.0	“My favorite food is pizza.”
0.7	“My favorite food is sushi, but sometimes tacos too.”
1.0	“My favorite food is tacos on a sunny beach with salsa!”

🎨 *Higher temperature → more creative and unpredictable responses.*

Example — Temperature in Action

- **Prompt:** (Email reply example reused)
- You are an AI support agent. Write a thank-you email for this positive review: "The blender works great and arrived early!"
- ✅ Use **low temperature** for consistent business tasks

  🎨 Use **high temperature** for creative writing

Temperature	Output Example
0.0	“Thank you for your feedback. We’re glad the blender arrived early.”
0.7	“We’re thrilled you love your new blender and appreciated our fast delivery!”
1.0	“Woohoo! Your blender came early — that’s the kind of surprise we love!”

Expanding with Control

- You can combine **instructions + temperature** to balance creativity and consistency.
- **Prompt Example:**
- Generate a short 3-paragraph blog on “AI in Education”. Be factual but slightly creative.

  Use temperature = 0.5.
- **Output:**
- Balanced mix of factual information + engaging tone.
- 💡 *Prompt + temperature together define output style.*

Expansion Workflow

- **Step-by-step approach:**
- Start with a short seed idea or summary
- Instruct ChatGPT to elaborate or expand
- Define tone (formal / friendly / creative)
- Adjust length or format (paragraph / post / script)
- Use temperature for variation
- Review and refine the result

Sample Prompts for Expanding

Goal	Prompt
Email writing	“Expand this feedback into a polite thank-you email.”
Blog writing	“Write a 200-word article on this topic.”
Brainstorming	“Generate 5 ideas based on this theme.”
Storytelling	“Expand this line into a short creative story.”
Academic writing	“Expand this point into a paragraph with examples.”

Expansion in Practice (Non-Coders)

- **Business use:** Draft customer responses, emails, reports
  - 🎓 **Education:** Generate learning materials or questions
  - 📝 **Writers:** Expand outlines into full drafts
  - 🎤 **Trainers:** Create examples or exercises
  - 🧠 **Analysts:** Turn insights into narratives or executive summaries

Summary — Expansion with ChatGPT

Concept	Description
Expansion	Create detailed, longer content from short input
Applications	Email, blog, ideas, essays, Q&A
Temperature	Controls randomness & creativity
Low temp (0-0.3)	Reliable, consistent outputs
High temp (0.7-1)	Creative, varied responses

Key Takeaways

- **Expansion** = idea → full content
- **Temperature** = creativity control
- **Combine context + constraints for clarity**
- **Use responsibly** — keep transparency
- **Experiment and compare multiple outputs**

Chatbots with LLMs

Build conversational agents easily using system/user/assistant messages, context, and retrieval — no heavy ML required.

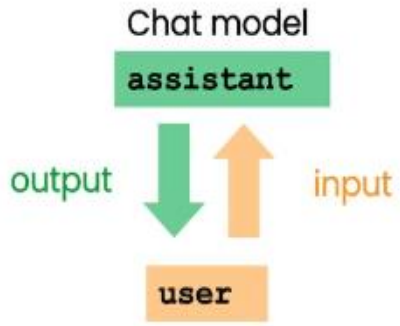
Why Chatbots with LLMs?

- All four prompt techniques (**Summarize, Infer, Transform, Expand**) come together in a Chatbot — which reads, analyzes, rewrites, and responds interactively
- LLMs let you create **custom chatbots** with modest effort.
- **Use cases: Customer support agents**, **order takers**, **tutors**, **virtual assistants**.
- **ChatGPT web UI is one example**; you can build custom bots using the chat-completions format and simple code.

OpenAI API call

```
def get_completion(prompt,
                   model="gpt-3.5-turbo"):
messages = [{"role": "user",
             "content": prompt}]

response = openai.ChatCompletion.create(
             model=model,
             messages=messages,
             temperature=0)
```

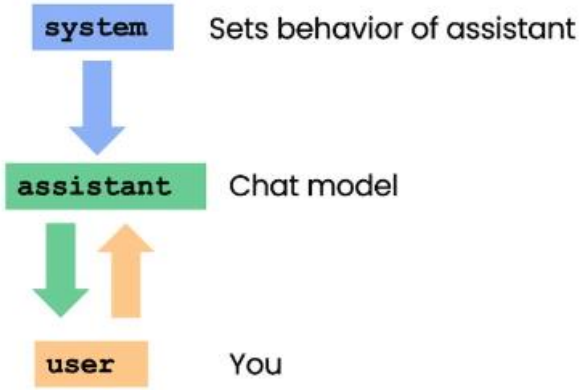


Chat Model: Message Format

- Chat models take a **sequence of messages** as input and produce an **assistant message** as output.
- **Typical roles**: **system, user, assistant.**
- **Multi-turn conversation** = a growing list of messages (context).
- This design makes **multi-turn** conversations simple and flexible.

Role

```
messages =
[
 {"role": "system",
  "content": "You are an assistant… "},
 {"role": "user",
  "content": "tell me a joke "},
 {"role": "assistant",
  "content": "Why did the chicken… "},
 …
]
```



LLM has no memory

- We fake memory by sending past conversation again.
- ChatGPT remembers because *we resend the conversation*, not because the model remembers.

```
[
  {"role": "system", "content": "..."},
  {"role": "user", "content": "..."},
  {"role": "assistant", "content": "..."}
]
```

- **system → Teacher / rules of the class**
- **user → Student asking questions**
- **assistant → AI responding**

Role of the system Message

- **System message** = high-level instruction / persona for the bot.
- Acts like a whisper to the model: sets tone, behavior, constraints.
- Users don’t see it in the conversation, but it guides responses.
- Example: *“You are a friendly pizza-ordering assistant. Keep replies short and helpful.”*

Basic Example: Shakespearean Assistant

- **System:** “You speak like Shakespeare.”

  User: “Tell me a joke.”

  Assistant: Responds in Shakespearean style (e.g., “To get to the other side, fair sir or madam.”)

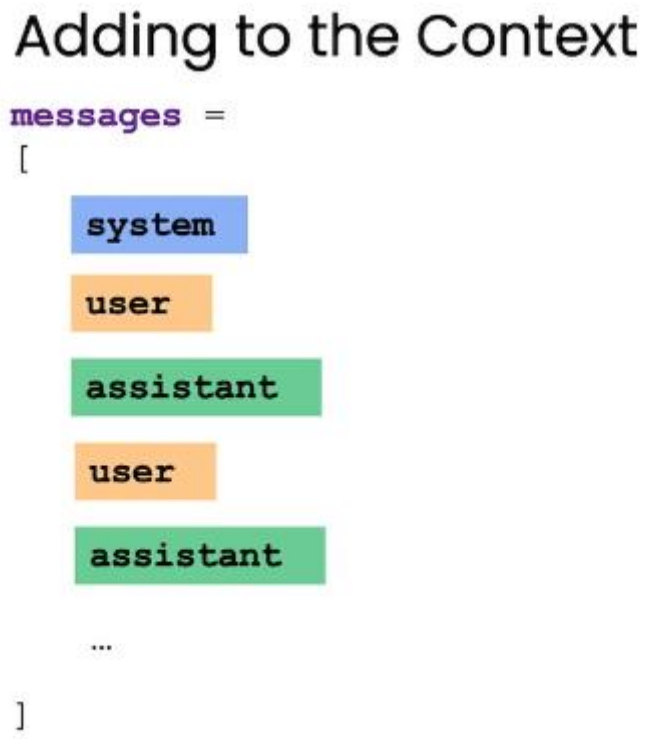
- **Teaching point:** System message controls persona & creative voice.

Context & Memory (Stateless Nature)

- Each request is **standalone**: the model only knows what you send.
- To preserve “memory”, include previous messages in the input (context).
- If you want the bot to remember a user name, include the earlier message where the user gave the name.

Building an OrderBot — Overview

- Create a **context** list that starts with a system message (menu + instructions).
- On each user input: append the user message, call the model, append assistant reply.
- Keep the context growing so bot can reference earlier turns.



OrderBot System Prompt (Example)

- You are OrderBot, an automated service to collect pizza orders.
- Greet, collect full order, ask pickup or delivery, ask address if delivery,
- clarify options/extras/sizes, summarize order and collect payment.
- Respond short, friendly, and confirm clearly.
- Menu: pepperoni (M/L), cheese (M/L), eggplant (M/L), fries, water.

Teaching tip: Put menu + flow in system message for predictable behavior.

How we talk to OpenAI: get_completion_from_messages()

- This function is the *only place* where we actually talk to OpenAI.
- Responsibilities:
- Take messages (context)
- Send to LLM
- Get response
- Return text

```
def get_completion_from_messages(messages, model="gpt-3.5-turbo", temperature=0):
    response = client.responses.create(
        model=model,
        input=messages,
        temperature=temperature, # this is the degree of randomness of the model's output
    )
#     print(str(response.choices[0].message))
    return response.output_text
```

Heart of Chatbot

```
def collect_messages(_):
    prompt = inp.value_input
    inp.value = ''
    context.append({'role':'user', 'content':f"{prompt}"})
    response = get_completion_from_messages(context)
    context.append({'role':'assistant', 'content':f"{response}"})
    panels.append(
        pn.Row('User:', pn.pane.Markdown(prompt, width=600)))
    panels.append(
        pn.Row('Assistant:', pn.pane.Markdown(response, width=600, styles={'background-color': '#F6F6F6'})))

    return pn.Column(*panels)
```

Step-by-step narrative

- **Step 1: Read user input**
  - prompt = inp.value_input
  - GUI gives text
  - We capture it
- **Step 2: Clear input box**
  - inp.value = ‘ ‘
- **Step 3: Append user message to context**
  - context.append({'role':'user', 'content': prompt})
  - Here memory start

- **Step 4: Call LLM with full context**
  - response = get_completion_from_messages(context)
  - Entire history is sent
  - LLM answers only based on this list
  - LLM does not know which message is new. It just sees a list.

- **Step 5: Store assistant response**
  - **context.append({'role':'assistant', 'content': response})**
  - This is how **conversation grows.**
  - Next user query will include this
  - What happens if I remove this line?
  - Bot forgets previous answer

- **Step 6 : UI Rendering (Panels)**
  - **panels is for display**
  - **context is for thinking**
  - **where** Panels are what you *see*.
  - ***and*** Context is what the *model sees*.

Importing & Initializing Panel (GUI layer)

```
import panel as pn  # GUI
pn.extension()

panels = [] # collect display

context = [ {'role':'system', 'content':"""
You are OrderBot, an automated service to collect orders for a pizza restaurant. \
You first greet the customer, then collects the order, \
and then asks if it's a pickup or delivery. \
You wait to collect the entire order, then summarize it and check for a final \
time if the customer wants to add anything else. \
```

```
bottled water 5.00 \
"""} ]  # accumulate messages


inp = pn.widgets.TextInput(value="Hi", placeholder='Enter text here…')
button_conversation = pn.widgets.Button(name="Chat!")

interactive_conversation = pn.bind(collect_messages, button_conversation)

dashboard = pn.Column(
    inp,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=300),
)
dashboard
```

- **import panel as pn # GUI**
- **pn.extension()**
- **What this does (conceptually)**
- Panel is a Python GUI framework
- Which is used to build interactive web apps inside jupyter.
- Panel is only the *body* of the chatbot.
- **Why pn.extension() is required?**
- loads JavaScript + CSS
- Without it → widgets won't render
- This is like switching on electricity before using appliances.
- Panel = **Python (backend logic)➕ JavaScript (browser behavior)➕ CSS (styling & layout)**
- Notebook or browser can **only understand HTML, CSS, and JavaScript,not Python objects.**
- Panel has a job to convert Python widgets-→ Web components

- **What happens if you write only this?**
- import panel as pn
- Python knows what pn.widgets.TextInput is
- Browser knows Nothing.
- It is like: “I know how to cook, but the gas stove is still off”.
- **What pn.extension() actually does?**
- **1. Injects JavaScript into the notebook**
- This JS is responsible for:
- Button clicks, Text input events, Sending user actions back to Python, Updating UI dynamically
- **Without JS:**
- Button won’t click, Text won’t update, No interaction possible.
- Without JavaScript, widgets are just pictures.

- **2. Injects CSS styles** (styling & layout)
- CSS controls:
- Layout, Spacing, Font, Colors, Scrollbars
- Without CSS:
- UI may look broken, Overlapping elements, No Proper sizing.
- CSS is the clothes; JS is the nervous system.
- **3. Registers Panel with the notebook / server**
- Panel tells Jupyter:
- I’m going to send dynamic content.
- Allow bi-directional communication.
- This enables:
- Python ↔ Browser communication
- Live updates without page refresh

- **4. Sets up a communication bridge (WebSocket)**
- **Browser (JS) ⟷ WebSocket ⟷ Python Kernel**
- **So When:**
- **User click 'Chat'→ JavaScript sends event→ Python function runs→ Response sent back**
- **→ UI Updates**
- **All of this fails silently without pn.extension().**
- **5. Real-world analogy**
- **Panel widgets are like remote-controlled toys**
- Python → the remote
- JavaScript → the signal
- Browser → the toy
- **If you don't turn on the signal(pn.extension()), pressing buttons does nothing.**
- **No extension-→ No interaction**
- **So pn.extension loads** the required JavaScript, CSS, and communication layer that allows Python widgets to render and interact dynamically in the browser.

What is a Widget?

- **A widget is an interactive UI element** that lets a user **give input or see output.**
- Think of widgets as: Bridges between human actions and Python code.

Widget	**Real-life analogy**
TextInput	Talking
Button	Doorbell
Slider	Volume knob
Dropdown	Menu card
Checkbox	Yes / No question

In Panel specifically

- A **Panel widget** is:
- A Python object that appear as a web element and can trigger Python functions
- pn.widgets.TextInput()
- Python object as a text box.
- **AI without widgets is a brain without a mouth.**
- **Widgets do NOT contain logic. They** only **collect actions.**
- Widgets capture user intent and pass it to Python logic.
- **Why widgets matter in AI apps?**
- Without widgets: You hardcode inputs, No interaction, Not a chatbot, just a script
- With widgets:User types a message, Clicks a button, AI responds, Conversation happens

Panels = [ ] -----Display Memory(Not AI memory)

- This is **not** ChatGPT memory.
- Panels are for *us*. Context is for *the model*.

Variable	**Purpose**
panels	What **user sees**
context	What **model remembers**

context— The Core of Conversation Memory

- context = [{'role':'system', 'content':""" ... """}]
- What context really is?
- A Python List
- Each element = One message
- This list is sent to the LLM every time.
- **LLM has no memory. We *re-send* memory every time.**
- **The system role:**
- **Defines personality**
- **Defines rules**
- **Defines conversation flow.**
- This is **prompt conditioning**, not learning.

- **Why the backslashes \ are used**
- It is used for python line continuation
- It makes text readable
- **Why menu is embedded in the system prompt**
- **LLM does not know your menu**
- **We inject domain knowledge via prompt**
- **Text Input Widget(User Mouth)**
- **inp = pn.widgets.TextInput(value="Hi", placeholder='Enter text here…')**
- **What this does?-→ create a text box, default text = “Hi”**
- **Why ‘Hi’---. Forces chatbot to start greeting.**

- **Button Widget (Trigger)**
- **Button_conversation = pn.widgets.Button(name="Chat!")**
- Role of button -→ Just an **event trigger**
- **pn.bind()--→ Connecting UI to logic**
- **interactive_conversation =pn.bind(collect_messages,button_conversation)**
- What it means:When button is clicked --→ Call collect_messages()
- **What collect_messages() does:**
- **1. Reads Input**
- **2. Appends to context**
- **3. Calls LLM**
- **4. Append response**
- **5. Update panels**
- **Button is the doorbell. Function is the house.**

Building the Dashboard Layout

- **dashboard** = pn.Column(inp, pn.Row(button_conversation), pn.panel(interactive_conversation,loading_indicator=True,height=300),)
- **pn.column(…):** vertical layout , Top ---→ bottom
- Order: text input----- Button -------Chat display
- **pn.panel(interactive_conversation):**
- **This renders:**
- **Output returned by collect_messages()**
- **Live-updating chat window**
- **Why loading_indicator = True**
- **Shows spinner while model responds**
- **UX improvement**
- **Why height = 300**
- **Scrollable chat window**
- **Prevents page overflow**

- **Render = Convert something into a visible form**
- Render means “show it on the screen.”
- Dashboard--→ show this
- **I am telling jupyter/panel**
- “Take this Python object and display it visually in the browser.”
- Rendering is the moment when code becomes experience.

Step-by-step what happens during rendering

1.Python creates the dashboard object

2.Panel translates it to web components

3.Browser receives HTML + CSS + JS

4.Browser draws it on screen

5.User sees the app

💡 Key line:

“Render = Python → Web → Screen”

Full Mental Model

- User types → inp
- Button click → collect_messages()
- collect_messages():
- ├── adds user message to context
- ├── sends context to LLM
- ├── receives reply
- ├── appends reply to context
- └── updates panels
- UI renders updated panels

Critical Questions

- **Where is memory stored?**

  → context

- **Can AI remember after refresh?**

  → ❌ No

- **Is this scalable for 1 million users?**

  → ❌ No (leads to DB, sessions, RAG)

- Chatbots don’t remember. Developers *simulate* memory using context.

Example Conversation (Order Flow)

- **User:** “Hi, I’d like to order a pizza.”
- **Bot:** “Great — which pizza would you like? Pepperoni, cheese, eggplant?”
- **User:** “Medium eggplant, no extra toppings.”
- **Bot:** “Any sides or drinks?” → User: “Fries and water.”
- **Bot:** “Pickup or delivery?” → collects address if delivery.
- **Bot:** Summarizes order and asks for confirmation → then collects payment.

Tips for Good Multi-turn Design

- **Clarify**: Ask extra questions to uniquely identify items (size, extras).
- **Wait**: Don’t finalize until full order collected.
- **Summarize**: Present a final summary and ask for confirmation.
- **Error-handling**: Handle ambiguous inputs by asking clarifying questions.

Temperature & Predictability

- For conversational tone: you may set moderate temperature (e.g., 0.3–0.7).
- For structured outputs (e.g., JSON for order submission): use **low temperature (0.0–0.2)** for consistency.
- Example: Use low temperature when generating the machine-readable order summary.

Producing Structured Output (JSON)

- After finalizing the order, ask the model to **create a JSON summary**:
  - Fields: pizzas (with size & extras), sides, drinks, delivery/pickup, address, total price.
- This JSON can be submitted to an order system or used downstream.
- **Prompt pattern:** Add a system/user message asking for JSON only with exact field names.

Context Management & Costs

- Context grows with each turn — be mindful of token limits and API costs.
- **Strategies:**
- Summarize earlier turns (compact history) instead of sending full chat.
- Store user profile or session info separately and include only necessary parts.
- Trim irrelevant messages.

Guardrails & Safety

•**Add guardrails in system message**: refuse harmful requests, protect privacy, avoid legal/medical advice.

•**Example:** “If the user asks for medical advice, respond: ‘I’m not qualified; please consult a professional.’”

•**Log conversations and have human fallback for escalations** (e.g., angry customers).

RAG & External Knowledge (Optional)

•For factual/enterprise bots, use **Retrieval-Augmented Generation**: fetch documents or FAQs before answering.

•RAG improves factual accuracy and keeps the bot up to date with company policies or menu changes.

Deployment Considerations (High Level)

- **Front-end**: Web chat, WhatsApp, or voice assistant.
- **Back-end**: Maintain session state, integrate payments / order system.
- **Observability**: Monitor failed intents, hallucinations, and user satisfaction.
- **Compliance**: Data retention, consent, and privacy policies.

Demo Ideas (Hands-on)

- **Live build:** Run OrderBot notebook and place sample orders.
- **Workshop tasks:** Modify system prompt to change persona (formal waiter, humorous bot).
- **Exercise:** Add JSON order export and validate in a downstream mock API.

Summary: Chabot's = Convergence of Skills

- Combines **Summarization, Inference, Transformation, Expansion**.
- System prompt + context design = chatbot behavior.
- Low effort to prototype, but production needs testing, guardrails, and monitoring.

Quick Prompt Templates

- **System (OrderBot):**
- You are OrderBot... [menu & flow instructions]. Respond short and confirm orders.
- **User → Finalize (JSON):**
- Create a JSON summary of the previous order with keys: pizzas, toppings, sides, drinks, delivery, address, total_price. Return JSON only.
- **Safety guard:**
- If user requests banned items or illegal help, reply: "I cannot assist with that."