

CSAI 422 - Applied Generative AI




From Completion to Instruction

- **Reinforcement Learning from Human Feedback (RLHF)** is a process that aligns large language models (LLMs) with human expectations.
- It improves **helpfulness, honesty, and harmlessness (HHH)** in AI models.
- RLHF involves multiple training steps, refining a base model into an aligned assistant.

Steps to Build an RLHF Model

1. **Start with a Base Model** (e.g., GPT-3)
2. **Supervised Fine-Tuning (SFT) Model** - Teaches the model to follow instructions.
3. **Reward Model** - Trains an evaluator to score completions.
4. **RLHF Model** - Uses reinforcement learning to optimize outputs.

 **Key Outcome:** The model becomes a **well-mannered assistant**, avoiding hallucinations, biases, and harmful responses.

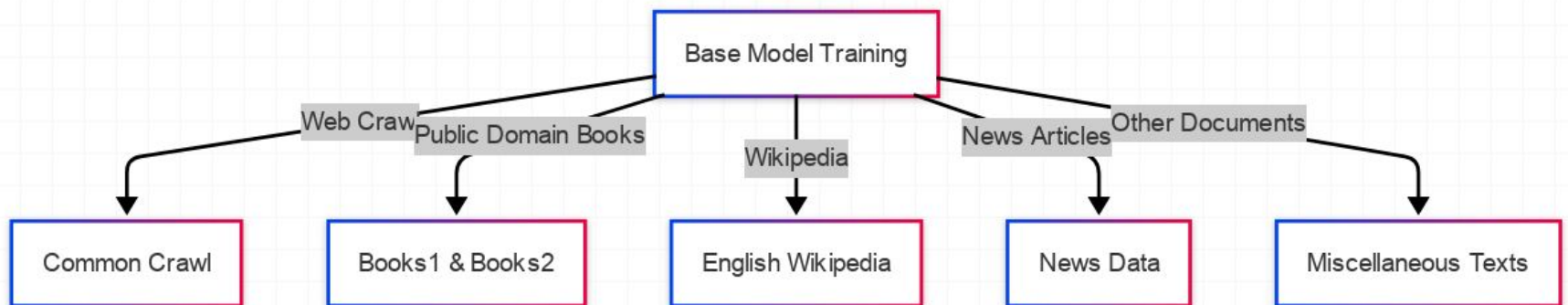
Base Model

Example: **GPT-3 / davinci-002**

Training Data:

- Public internet data, books, Wikipedia, etc.
- 499 billion tokens from diverse sources.

Capabilities: Mimics various document styles but **hallucinates** details.



Supervised Fine-Tuning (SFT)

Purpose: Make the model follow instructions and chat coherently.

Training Data:

- 13,000 human-written conversations.

Process:

- Model learns to **replicate ideal human responses**.
- Fine-tuned on smaller, high-quality datasets.

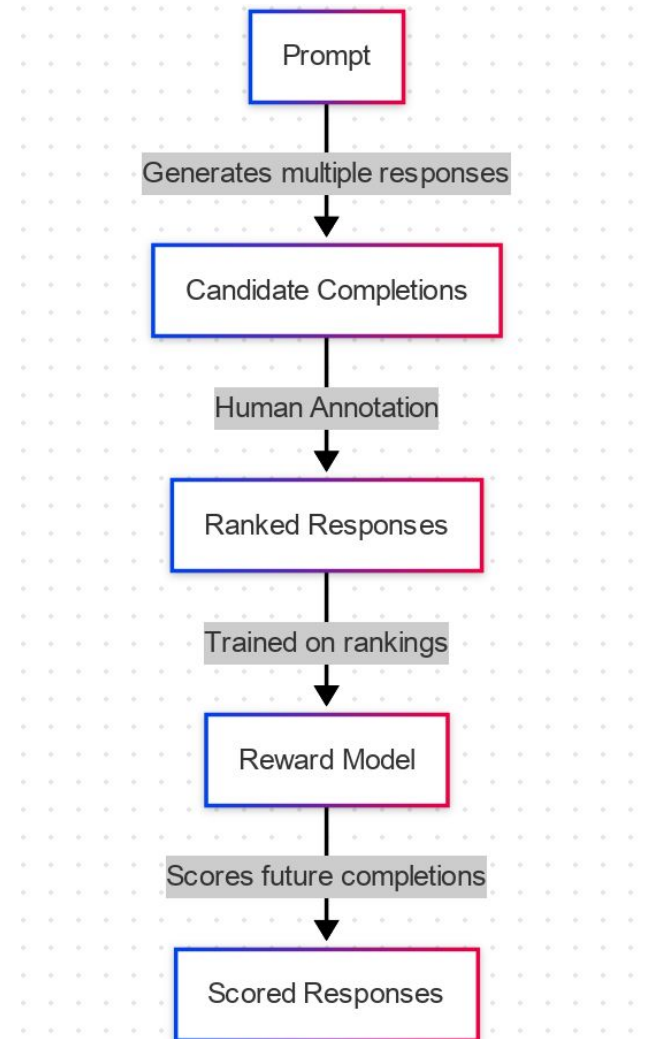
- **Purpose:** Judge completion quality based on human rankings.
- **Training Data:**
 1. 33,000 documents with human-ranked completions.
- **Process:**
 1. Generate multiple completions for a prompt.
 2. Human annotators **rank** responses.
 3. Model learns to score responses like a human.

Reinforcement Learning (RL)

RLHF integrates reinforcement learning (RL) with human feedback.

Model generates completions →
Reward model assigns scores →
Fine-tuning optimizes response generation.

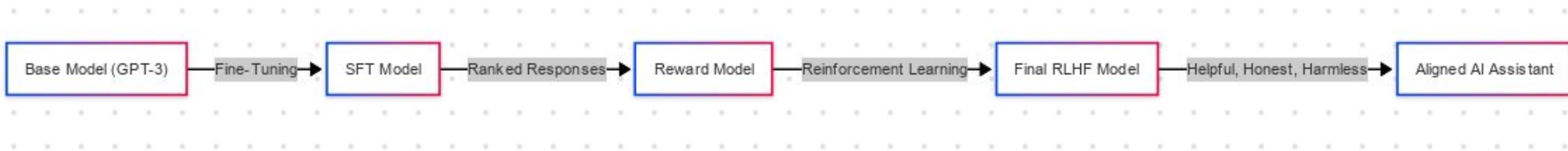
Uses **Proximal Policy Optimization (PPO)** to ensure responses remain human-like.



Summary of Model Stages

NEWGITA UNIVERSITY

Model	Purpose	Training Data	Docs Used
Base Model	Predict next token	Web, Books, Wikipedia	499B tokens
SFT Model	Follow instructions	Curated chat responses	13,000 docs
Reward Model	Score completion quality	Human-ranked responses	33,000 docs
RLHF Model	Optimize completions	Reward scores from RM	31,000 docs



Understanding Chat API Conversations

When building a Chat API application, it's important to distinguish between two types of conversations:

1. **User-to-Assistant Conversation** – The direct interaction between the end user (a human) and the AI assistant.
2. **Application-to-Model Communication** – The structured exchange formatted using **ChatML**.

Both involve a **user** and an **assistant**, but they are **not the same conversation**.

What is ChatML?


Chat Markup Language (ChatML) is a structured format that organizes messages exchanged between the application and the model.

Why ChatML?

- It ensures consistency and context retention in AI interactions.
- It helps define the roles of different messages in a conversation.

Key Roles in ChatML

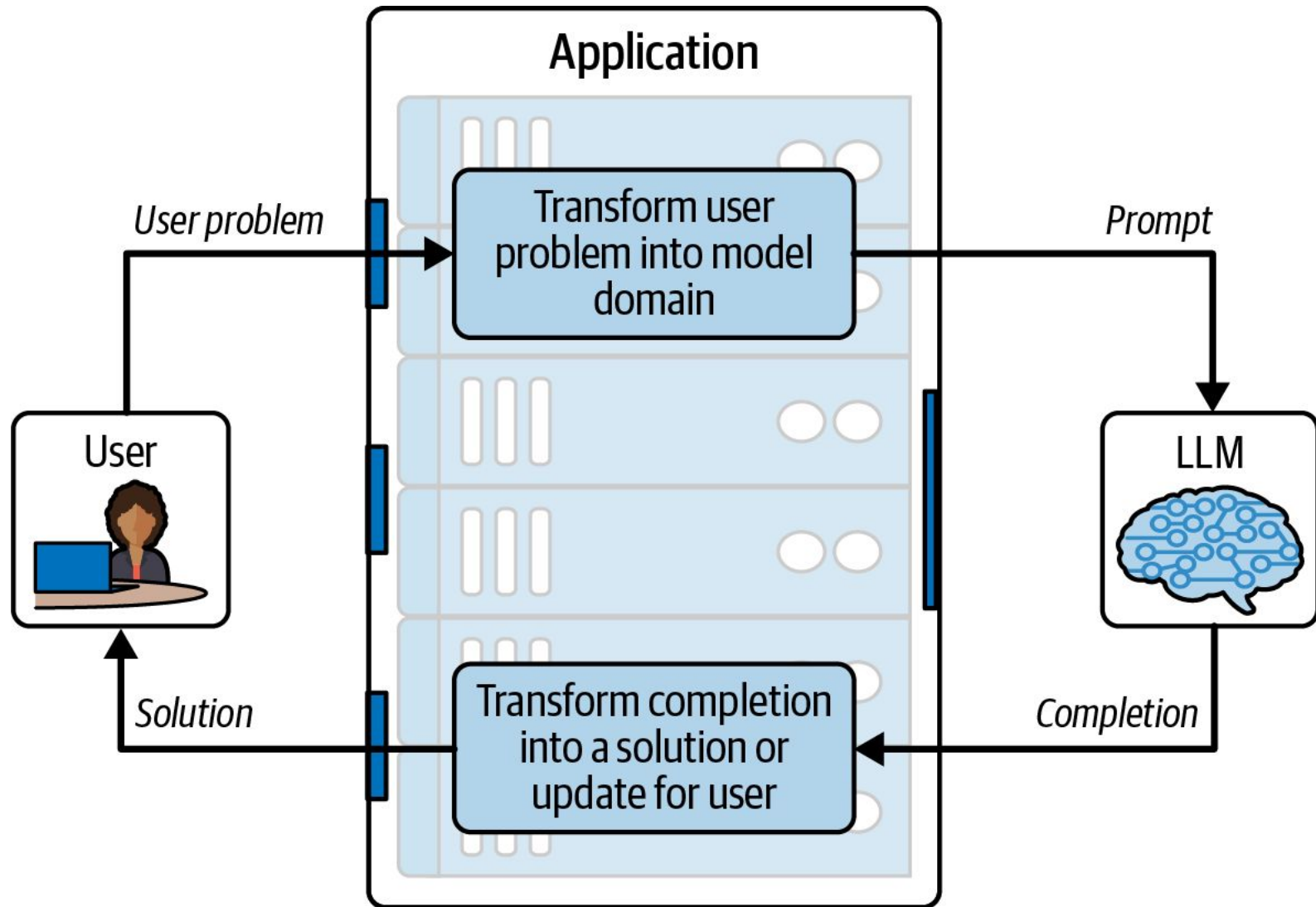
1. **User** – Represents the human input in the structured transcript.
2. **Assistant** – The AI model's response to user queries.
3. **System** – Provides context, rules, or instructions for the assistant.
4. **Function** – Represents function call outputs, often used for API integrations.

 These roles help structure AI conversations effectively within applications.

LLM Application!

The LLM Application Loop

NEWGITA UNIVERSITY



- **User-LLM Interaction Modeled as Loop**
 - Back-and-forth interaction between user and language model (LLM)
 - User domains vary greatly, often distinct from LLM's domain
- **Diverse User Activities**
 - Simple tasks: Writing emails, finding suitable phrasing
 - Complex tasks: Coordinating group travel, booking tickets, arranging accommodations
- **Non-direct Interaction Scenarios**
 - Scheduled analyses or recurring tasks triggered by new data availability

- **Model's Primary Function**
 - Completes documents based on given prompts
- **Flexibility Afforded by Document Completion**
 - Enables generation of various text types:
 - Emails
 - Code
 - Stories
 - Documentation
 - Specialized transcripts (for chat apps and tool execution)
- **Versatile Use Cases**
 - Text completion enables a wide range of applications, such as:
 - Chat interfaces
 - Tool execution via function calling syntax
 - Almost unlimited use cases based on text generation capabilities

The Loop in LLM Applications

- **Understanding the Loop**
 - Implements transformation between user domain and model domain
 - Converts user problem into document or transcript for the model to complete
 - Transforms model output back into a solution for the user
 - Can be used for:
 - Single iterations (e.g., converting bullet points to prose)
 - Multiple iterations in complex applications

Iterations and State Management

- **Single vs. Multiple Iterations**
 - Single iteration: Simple tasks requiring one loop through the model
 - Multiple iterations: Complex tasks like chat assistants or travel planning
- **State Management**
 - No state retention in simple use cases
 - Maintain state across iterations for evolving problems (e.g., travel planning)
 - Example workflow:
 1. Brainstorming travel ideas
 2. Making arrangements
 3. Setting reminders

Dimensions of the User's Problem

- **Medium:**
 - How the problem is conveyed (e.g., text, images)
 - Text is most natural for LLMs
- **Abstraction Level:**
 - Varies from simple to complex reasoning required
- **Context Information:**
 - Amount of additional information needed beyond what's provided by the user
 - Most domains require retrieval of extra info
- **Statefulness:**
 - Memory of past interactions and user preferences
 - More complex problems need this

Application Examples

	<i>Increasing complexity →</i>		
	Proofreading	IT support assistance	Travel planning
<i>Medium of the problem</i>	Text	Voice over the phone	Complex interactions on the website, text input from the user, and interactions with APIs.
<i>Level of abstraction</i>	The problem is concrete, well-defined, and small.	A large abstract problem space and a large solution space, but constrained by available documentation	The problem involves understanding the user's subjective tastes and objective constraints in order to coordinate a complex solution.
<i>Context required</i>	Nothing more than the text submitted by the user.	Searchable access to technical documentation and example support transcripts	Access to calendars, airlines APIs, recent news articles, government travel recommendations, Wikipedia, etc.
<i>Statefulness</i>	No statefulness—every call to the API contains a distinct problem statement.	Must track the conversation history and solutions attempted	Must track interaction across weeks of planning, different mediums of interaction, and aborted branches of planning.

Converting the User's Problem to the Model Domain

- **Transforming User Input:**

The process begins by converting the user's problem into a format that aligns with what language models are trained to understand and generate responses for. This involves translating real-world problems into structured prompts or inputs that guide the model effectively.

- **Role of Prompt Engineering:**

At the heart of this transformation is prompt engineering, which focuses on crafting prompts that not only capture the essence of the user's problem but also ensure that the model generates accurate and relevant responses. The effectiveness of this step significantly influences the quality of the solution provided by the language model.

Crafting the Perfect Prompt

- **1. Resembling Training Data:**

The prompt should closely mirror the structure and style of the data used to train the model, ensuring that it aligns with what the model is familiar with.

- **2. Including All Relevant Information:**

It's crucial to provide all necessary details related to the problem to enable the model to generate a comprehensive and accurate response.

- **3. Guiding Model Generation:**

The prompt should be designed in a way that directs the model toward producing a completion that directly addresses the user's problem, ensuring relevance and precision.

- **4. Ensuring a Reasonable Endpoint:**

A well-crafted prompt includes cues or instructions that guide the model to conclude its response at an appropriate point, preventing unnecessary continuation and enhancing clarity.

Optimizing Prompts by Leveraging Document Familiarity in LLMs

Key Points:

- **Ask About Document Types:** Inquire directly into document types relevant to your needs. Example query: "What types of formal documents are useful for specifying financial information about a company?"
- **Generate Example Documents:** Request the model to create an example of the identified document type.
- **Use as Templates:** Utilize these examples as templates for crafting more effective prompts.

Why It Works:

- Aligns with the model's training data, enhancing prompt effectiveness.
- Provides insights into structures familiar to the LLM.

The Evolution to Function-Calling Models

- **Function-Calling Models :**
 - Define specific tasks or functions within the prompt.
 - Model generates function calls instead of raw text.
- **Example Functions :**
 - `lookup_airline_flights()`: Retrieve flight information based on user criteria.
 - `book_hotel_room()`: Book a hotel room for specified dates.

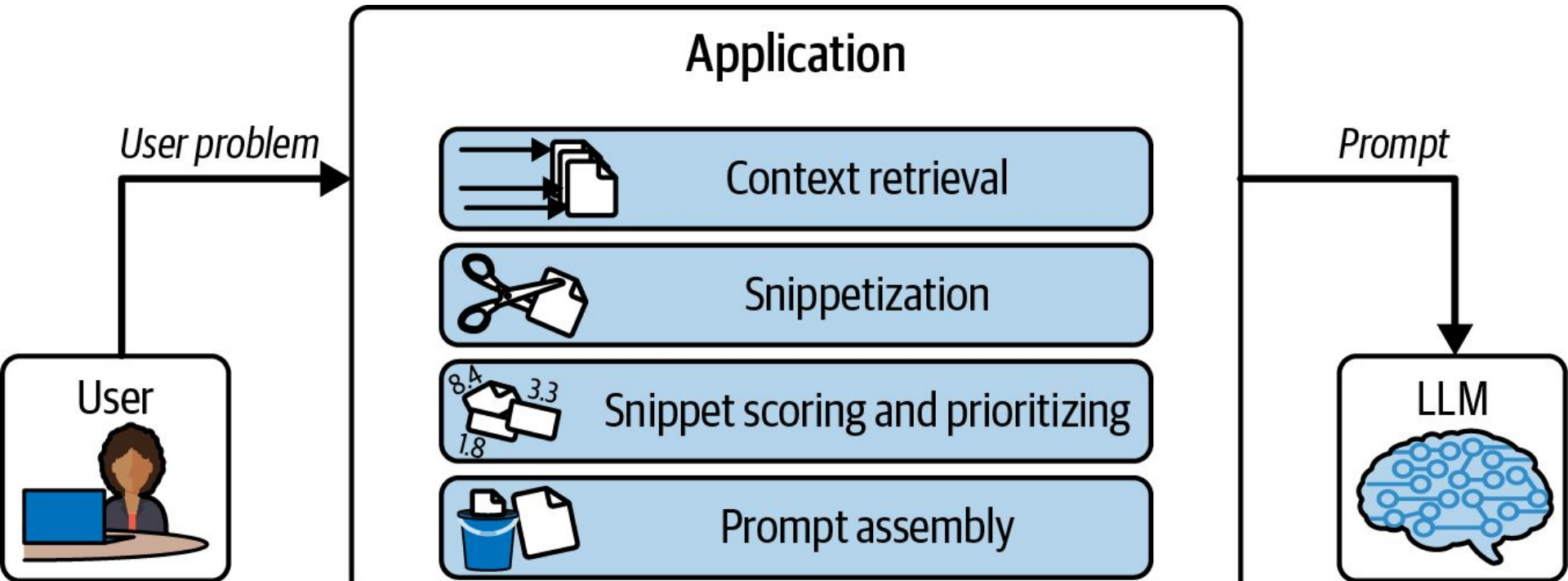
How Function-Calling Models Work

1. **Problem Description :**
 - User specifies their needs (e.g., travel goals).
2. **Function Call Generation :**
 - Model interprets the problem and generates appropriate function calls.
 - Example: `lookup_airline_flights(origin="New York", destination="Paris", date="2023-10-15")`.
3. **API Interaction :**
 - Application uses these function calls to interact with external APIs (e.g., airline systems).
4. **Result Presentation :**
 - Retrieved information is formatted and presented back to the user.

Example in a Travel App

- **Scenario :**
 - User wants to book a flight from New York to Paris on October 15, 2023.
- **Function Call :**
 - Model generates:
`lookup_airline_flights(origin="New York",
destination="Paris", date="2023-10-15").`
- **Outcome :**
 - Application retrieves available flights and displays them to the user.

Zooming In to the Feedforward Pass

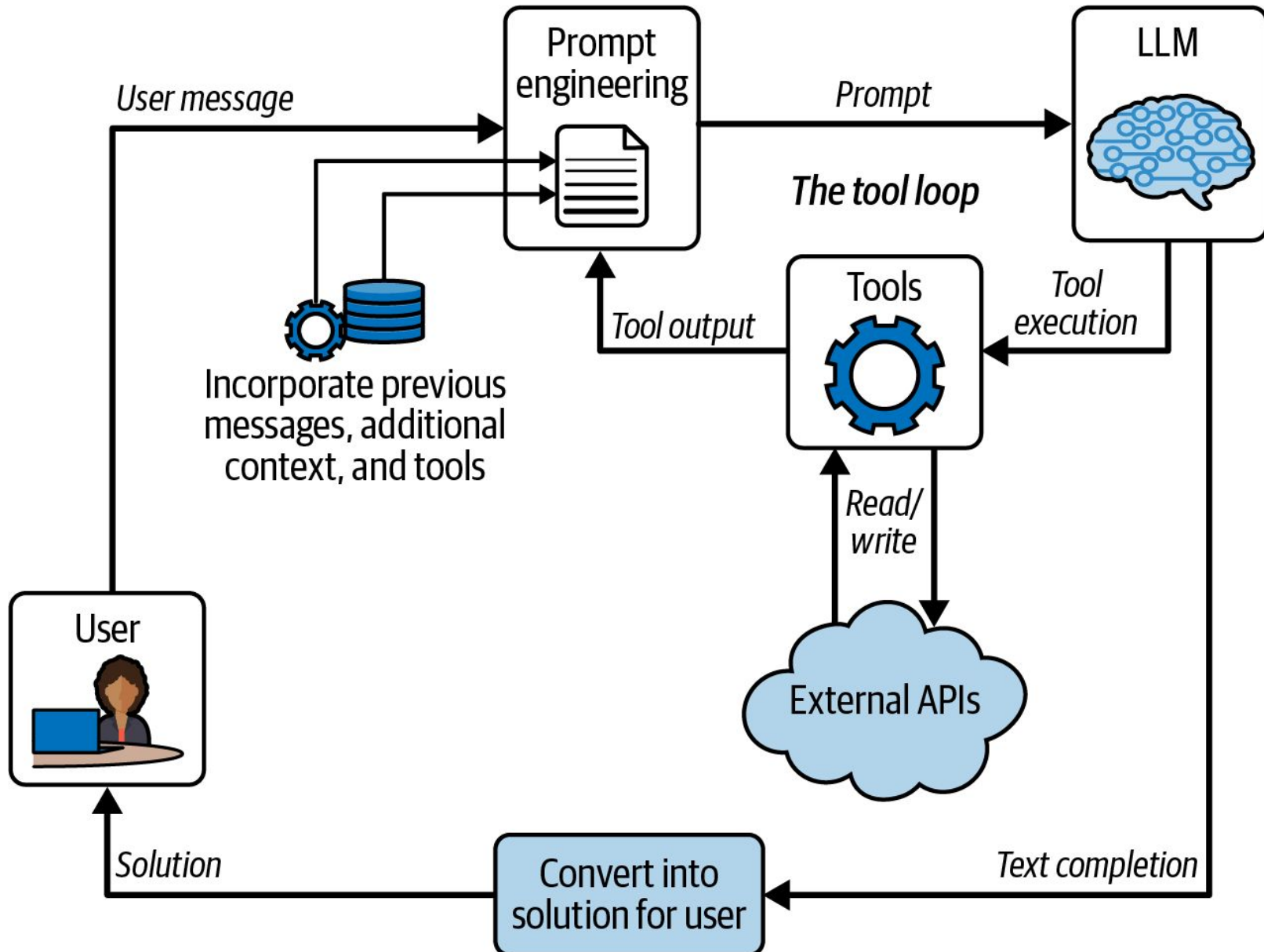


- The first step in building the feedforward pass is gathering or retrieving raw text that serves as context for the prompt.
- Context can be categorized based on its *directness* :
 - **Direct Context:**
 - Comes directly from the user describing their problem.
 - Example: Text entered by a user in a help box or code being edited (e.g., GitHub Copilot).
 - **Indirect Context:**
 - Derived from relevant nearby sources.
 - Example: Searching documentation for excerpts related to the user's issue or other open tabs in an IDE that may contain relevant snippets.
 - **Boilerplate Context (Least Direct):**
 - General framing text used to introduce the problem and shape the model's response.
 - Example: A message like, "This is an IT support request. We do whatever it takes to help users solve their problems."

Role of Boilerplate Text

- **Introduction:** Sets the stage for the problem (e.g., travel plans).
- **Connection:** Acts as glue to combine direct and indirect context.
- **Example:**
 - Non-bolded text introduces the travel problem.
 - Additional boilerplate later connects user-specific details with relevant news or government sources.

Tool usage



Required Reading

Required Reading

- Chapters 3 and 4 of “Prompt Engineering for LLMs”
- Lex Fridman on Deepseek, specially the first hour, which about the technical elements ([YT](#), [Spotify](#))
- [Blog](#) on prompt driven development

