Antechamber Delivery

Project Instructions

Outlier

Introduction

Your mission is to train an advanced AI by guiding it to use tools/functions in multi-step turns. Think of yourself as a collaborator, not just a tester. You will teach the model how to behave and respond in various situations, choosing correct tools to solve problems.

✅Your task will take this form:

1. *Read the **Task specifications**:*
    a. Interaction Scenarios(Task categories) you need to cover in the conversation
    b. tools available for usage in this task.
    c. System settings + device information(i.e. calendars)
2. Create a **System prompt** for the conversation with the agent.
3. Correspond in a **conversation** where you:
    a. Write a min of 10 prompts.
    b. Ask questions that cover the Interaction Scenarios and ask questions to trigger tools.
    c. Label and/or fix wrong tool calls or responses of the model.

🔑 Key Concepts

Before you begin, it's important to understand the core terms used throughout this project.

Term	Definition
System Prompt	A customized set of instructions you create for the AI at the start of each task. This "master prompt" defines the agent's personality, context, and specific rules it must follow. The agent's performance is judged against this prompt.

Term	Definition
Device Information & System Settings	The agent operates with internal databases that store information such as system settings (e.g., WiFi status) and device-specific data (e.g., calendar events). This information can be accessed and used by certain tools during the conversation.
User Prompt	A request or answer you **write** to the AI. These should be natural and part of a larger scenario. **Only user prompts count towards the project minimum**, and you must create at least **10 user turns** per task.
Model's response	The model can respond splits into **3 categories:** 1. `Text_response` - can be a clarifying question or an answer to the user's prompt. 2. `Tool_call -` a model's execution of a tool. 3. `Tool_response -` raw JSON data that the model *returns* after the tool has been executed.
Tools (Functions)	To complete requests made by the users the model is expected to execute tools/functions that are available for it in a task; such as searching for places, getting directions, or interacting with a calendar.
Lazy User	You are expected to adopt this persona for all user prompts in this project. The "Lazy User" starts with vague requests and only provides information when explicitly asked.

📌Important Notes:

1. In this project you have ***3 sources*** you will ***refer to regularly*** while doing a task:
   - ***Instructions*** **-** this document
   - ***Available tool domains*** **-** Contains a description on the available tools per domain.
   - ***Full Tool guide*** **-** how to use each tool and what parameters it has.
2. You must familiarize yourself with the **unique key terms** of this project - completing a task successfully is dependent on understanding these terms.
3. **NOTE:** Each task **must** include requests that prompt the model to ***use at least 3 tools***!

4. ⚙️ **Technical Overview & Key Notes**

This task includes several **technical nuances** you should keep in mind:

a. 🔄 **Tool Call Flow**

When the model makes a `tool_call`, the following sequence will unfold over the **next two turns**:

- 1️⃣**First turn**: The platform will generate an **empty prompt**, followed by the model's `tool_response` — a **JSON output** of the previous `tool_call`.
- 2️⃣**Second turn**: Another **empty prompt** will be followed by the model's **natural language summary** of the `tool_response`.

📝 **Reminder**: Every `tool_call` is always followed by a `tool_response`, and then by a natural language interpretation of that response.

⚠️ **Heads up**: In some cases, a `tool_response` may be followed by another `tool_call` instead of the natural language summary. This is expected behavior.

b. 🛠️ **Fixing Incorrect Tool Calls**

if you would like to change a tool call parameters **after** and run; you **MUST** regenerate the **last** user prompt or else the changes won't go into effect.

c. ⚙️ **System Settings**

The system settings define the conversation's initial state.
Some settings are **interdependent** — they may not be enabled or disabled at the same time. E.g. low_battery_mode and wifi cannot be enabled at the same time.

d. 🕒 **Timeout Issues**

If the model times out or gets stuck:
🔄 Try going back one or two turns and **re-generating the response**. This usually resolves the issue and allows you to continue tasking.

📌Important Links

- Outlier Community (English)
- War Room

Table of Contents

Task Overview

Each task involves a structured conversation with the model to test and refine its performance. You'll follow these key steps:

🟡1️⃣ Step 1: Prompt Writing: Scenario & system prompt creation

- Check the task specifications; task categories, available tools and system settings.
- Write a unique, detailed system prompt that defines the agent's persona, context, and behavioral rules.
    - Think of the system prompt as a *master instruction*—a core memory or guiding principle the model should follow throughout the conversation.
- The first prompt in the conversation should always be a request.

🟡2️⃣ Step 2: Guiding the Model: Interaction with the Agent:

- Before writing a user prompt, think through the episode you want to create with the model by interacting with it. Take inspiration from your own life where personal agents could be helpful.
    - Ex: Need help moving to a new city, finding classes for your kids, selecting courses in college, cleaning up your calendar.
    - Make it complex enough that they will lead to 10+ turns of back and forth - complex multiturn trajectories are the most important aspects of this task
- Craft an initial user prompt that will start exploration of the scenario.
    - "I need to pick courses next semester"
- Send user prompts to the agent completion endpoint, following your designed scenario.
- User prompts should always be in a lazy user form.

🟡3️⃣ Step 3: Model Response Editing, Error Tagging, Critique, and Reasoning

The model will respond with a **tool call, clarifying question, or text response**. Guide the model by labeling and fixing the turn with the right tools, parameters, or text response.

- Formulate user utterances that align with your designed scenario/category.

- For each response - critically evaluate the agent's responses against the scenario goals and the custom system prompt.:
    - Mark any error types.
    - Mark the turn's scenario/category.
    - Mark the response type:
        - text_response / tool_call / tool_response

- If any errors exits:
    - Provide a critique
    - Provide a reasoning in 1st tense.
    - Fix the response/

- correct the model response, provide the **error type** and a **critique** of the model's action. Then, explain your reasoning for the correction.

🟡4️⃣ **Step 4: Continue the Conversation**

Continue the conversation **until you reach at least 10 user turns** - a turn where you actually write something to the model.

🟡1️⃣ Step 1.a: Understanding Task specifications

In the beginning you would see the task specifications you must follow:

1. 🧩 **Designing Scenarios/Categories**

   Your conversations should be built around specific test scenarios that reflect different interaction types. Understanding these categories will help you design realistic user prompts and guide the model effectively.

   ! * You MUST cover all specified task categories given in a task

Category	Description and Goal

`[Lazy User]`
`- ALWAYS A requirement`
	In **EVERY** prompt in the conversation you will play a lazy user; a vague and uncooperative user. Only provide information when asked, and only one detail at a time.
	• Not provide information unless **explicitly** asked by the assistant.
	• Provide only **one** piece of information at a time, **even** if asked multiple questions.
	• Correct the assistant if it makes wrong assumptions.
	• **Start with underspecified requests.**
	  • ***Example:*** User: "I need a reservation."
	  ▪ Assistant: "What type, location, date, etc.?"
	  • User: "Restaurant."

Category	Description and Goal
`[Feasible Tool Use]`	Design a task the agent **can complete** using the **available** tools. The agent should try to fulfill the request without asking for info it can infer (e.g., current location) • ***Example:*** User: "Find Italian restaurants near me." ◦ Agent should get the current location, and then search for Italian restaurants.
`[Infeasible Tool]`	Design a request that cannot be fulfilled with available tools. The agent should recognize that the tool is non available and default to declaring inabilities/limitations. • ***Example:*** User: "Does Nola Restaurant accept Apple Pay?" ◦ Agent may search for Nola, get details, but if payment info isn't available via tools, it should state that and suggest alternatives like calling the restaurant. **Note:** When the model is faced with an infeasible tool request from the user → In the first response, the model should default to explaining to the user that this request ***cannot*** be completed by the model. • The model ***should not*** ask clarifying questions in response to infeasible tool use tools and should immediately acknowledge the infeasibility of the request.

`[General Chat]`	Ask a general knowledge question (i.e., static, widely known information) that should not require a tool. This tests the model's ability to distinguish between when a tool is necessary and when a simple text response is sufficient.
	When should the model respond with text only and not use a tool?
	• ✅ **If a tool is available** that can retrieve the correct answer, the model **must use the tool**.
	• ⚠ **If the required tool is not available** in the task, but exists in the overall tool guide, the prompt falls under the **Infeasible Tool** category. In this case, the model should **explain its limitation** without asking further clarifying questions.
	• ❓ **If the model attempts to answer using its prior knowledge**, assess whether the question is truly **general knowledge**.
	  ◦ If it **is**, the response is acceptable.
	  ◦ If it **isn't**, the model should state that it cannot answer the question.

Category	Description and Goal
	• ***Example:*** User: "What color is the Golden Gate Bridge?" ◦ Agent should directly answer "International orange"
`[State Dependency]`	Design a scenario where a tool fails due to the state of the system settings (e.g., WiFi is off, low battery mode is on). Your prompts should guide the agent to identify the problem, modify the state (e.g., turning WiFi on), and then successfully re-run the tool. • ***Example:*** User asks to search for a place. ◦ Initial DB has WiFi off. Agent tries to search, gets a "WiFi not enabled" error. Agent then checks settings, sees WiFi is off. Tries to turn WiFi on, but fails if "low_battery_mode" is on. Agent then turns off low battery mode, then turns on WiFi, then successfully performs the search. **Note:** on turns where the model makes an error (e.g. uses a tool that needs WiFi without WiFi being on): you must mark **is_error = TRUE (only for error recover and state dependency tasks)**
`[Error Recovery]`	Design a situation where the agent uses a tool **incorrectly** (e.g., with wrong parameters or a too-specific query that yields no results). **Crucially, do not correct the agent in your first attempt.** Instead, your subsequent user prompts must guide the agent to recognize its own mistake and refine its tool call. • ***Example:*** User: "What is my next one on one?" ◦ Agent searches calendar but misses datetime_range_upperbound, gets an error. In the next turn, *you guide it* (e.g., by a user prompt like "Search for the next year") so the agent re-tries the tool call correctly. If it still fails (e.g. searches "one on one" but event is "1:1"), guide it again (e.g., User: "It might be named differently, and it's with Jill") for the agent to try a broader search or different parameters. **Note:** on turns where the model makes an error (e.g. uses a tool with incorrect parameters): you must mark **is_error = TRUE (only for error recover and state dependency tasks)**
`[Task Switching]`	Design a scenario where you naturally **deviate from the current task** to ask about something unrelated (a "detour"), and then switch back to the original task after the detour is complete.

Category	Description and Goal

	• ***Example**:* User: "Add an event at 5 tomorrow."
	  ◦ Agent clarifies AM/PM, asks for calendar.
	  ▪ User: "Wait, do I have my WiFi on?"
	  • Agent checks WiFi, responds. User: "K, add the event to my personal calendar."
	
	In order to successfully qualify as task switching, **the initial task must not be completed prior to the next prompt** requesting a detour from the model.
	
	• ***Example***: User: Find me some music
	  ◦ Agent clarifies: what kind of music?
	  ▪ User: Give me the weather forecast first?
	  • Agent checks weather, responds. User: Okay, rainy weather calls for some jazz.

`[Natural User]`	In this category, the user is more verbose than lazy user, but still casually conversational. The user should not provide too much information in a sentence still, but should create user prompts that requires multiple tool calls in order to complete the task. You should do this at least 3 times in a trajectory for *Natural User* tasks • ***Example***: "How many tracks are in Human After All" (requires searching for album, getting the id, finding details). • ***Example***: "Directions to the closest McDonalds". (requires getting current location, searching for McDonalds, finding the closest one, getting directions).
`[Search Refinement]`	In this category, you need to create queries that would require the model to gradually refine its search queries to find the relevant information. Search related tools are sometimes not accurate enough, which requires the model to modify arguments / try other tools. • ***Example***: When the user asked to search for Dream Theater concert, search_events, if no relevant results are seen, spotify_artist_goods, then web_search, then search_calendar_events. • ***Example***:When the user asked to search for meeting with John in the morning, the model might start by searching for "John" as query and 9AM - 12AM as starting time range, if no relevant results, try

Category	Description and Goal
	using "John" as participant name and 9AM - 12AM, if no relevant results, try 9AM - 12AM, and in the end, try searching for the whole day. **Note:** You can control how far and wide you'd like search refinement to go through system prompt as well.

2. 🧰 Tool Domains Available in This Task

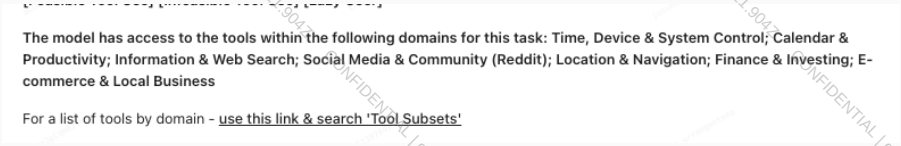
A core aspect of this project is the model's ability to use tools to fulfill user requests during the conversation—such as creating calendar events, providing directions, sharing Spotify links, or reporting the weather. The model has access to over 80 tools in total.

You can find a complete list of tools and their details in the **Tool Guides**. This includes:

- **Tool Domains:** What exact tools each domain includes and their description.
  - *Use Ctl+f with the tool domain name*
- **Tool Usage:** Complete tool list on how to use each tool, including required and optional parameters.
  - *Use Ctl+f with the tool name*
  - Understanding this will help you anticipate the steps the model needs to take before using a tool, and allow you to critique or correct its behavior effectively.

Important: Not all tasks will grant access to the full toolset. Tool availability will vary by task. You can identify which tool domains are available in each task by reviewing the **Tool Domains section** at the beginning of each task.

! * You MUST cove at least 1 tool domain given in a task.



3. ⚙️ System Settings & Device Information

The agent operates with internal databases that store both **system settings** (e.g., WiFi status, low battery mode) and **device-specific data** (e.g., calendar events). Certain tools can access this information during the conversation—**but only** if the model retrieves it using the appropriate tools or if it is explicitly provided in the system prompt.

🔧 System Settings

- Each task may include some or all of the model's system settings.
- If you reference any of these settings in your system prompt, you **must match them exactly** as defined in the task.
- These settings are **informative only**—the model will not "know" them unless:
  - They are written into the system prompt
  - Or the model retrieves them using a tool like `get_system_settings`

This ensures consistency between what the model expects and what is actually defined or retrieved during the task. Misalignment between the system prompt and the tool-retrieved values can lead to confusion or incorrect behavior.

⚠️ Disclaimer: Interdependent Settings

Some system settings are interdependent and **cannot be enabled or disabled at the same time**. For example:

- `low_battery_mode` and `wifi` cannot both be active—**low battery mode disables WiFi**.

If a tool requires certain settings (e.g., WiFi enabled), you may need to guide the agent to **modify the system states** appropriately before proceeding.

Example:
*Not all tasks will have the same settings, so **read them carefully**.

```JSON
[
{
"cellular": false,
"device_id": "4ea67c9d-bfe2-47bd-9f83-f21761d3fd4b",
```

```
"formatted_address": "1213 N Maple Dr, Beverly Hills, CA 90210, USA",
"latitude": 34.0862128,
"locale": "en_US",
"location_service": true,
"longitude": -118.4024531,
"low_battery_mode": false,
"place_id":
"Ei0xMjEzIE4gTWFwbGUgRHIsIEJldmVybHkgSGlsbHMsIENBIDkwMjEwLCBVU0EiMRIvChQKEglfat
PVG7zCgBGgdBwB0EB0JRC9CSoUChIJ3R7i7AO8woAROvNgChW02do",
"utc_offset_seconds": -25200,
"wifi": true
}
] *
```


🟡1️⃣ Step 1.b: Crafting the System Prompt

Every task you do begins here. The System Prompt sets the stage and defines the **rules of engagement** for the specific scenario you are about to create.

Think of the System Prompt as a **master instruction** or a core memory for the agent that governs its behavior throughout the entire task. It defines the agent's personality, its awareness of the user's situation, and the specific rules it must follow. A well-crafted System Prompt is the key to creating a successful and high-quality scenario.

Core Components

To create a unique and complex System Prompt, you must follow these 2 guidelines:

1. ***Uniqueness & shape***
    a. Every task requires a new, unique System Prompt. No two should share the same wording or block order
    b. Mix, Match, and Be Creative the chosen elements in a random, coherent order so that model does not learn unwanted patterns
2. ***Include ≥ 4 of 5 building blocks***
    a. The prompt must be in a **natural language paragraph** form - NOT list of rules.
    Do not make it obvious that the prompt is structured by these categories; instead, weave them together into a natural, coherent paragraph.

3. **Length & richness -** "The longer and more complex, the better." Rich prompts yield better training signals

⚠ Warning:
System prompts are subject to guardrails that enforce a minimum length and required complexity.
Reusing or copying system prompts across tasks is strictly forbidden and may result in account deactivation. This behavior is strictly forbidden.

System Prompt Banned Content:

No Explaining User Behavior Patterns
❌ *Don't explain that the user gives minimal information, withholds context, or will correct only after mistakes.*
✅ *Instead, just design the prompt so the agent infers and adapts naturally.*
No Stating Infeasible Tool Use Rules
❌*Don't say "you can't access inventory or payment tools" or "you must fail gracefully if a tool doesn't work."*
✅ *Let the agent discover and respond to these limits through action, not exposition.*
No Forecasting Task Switching
❌*Avoid lines like "the user will change direction mid-task."*
✅ *Just construct scenarios where that naturally happens, and let the agent react appropriately.*

Building block	**Definition**
1. **Context Information** (The Agent's Environment)	• **What it is:** Summarizes the agent's current environment, such as its location, the current time from the **system settings**. • **Rule:** If context is provided (e.g., location), the agent should use it and **not** call a tool for that same information. • **Examples:** `"You are currently located in 5000 Forbes Ave, Pittsburgh, PA"`, `"The current time is 4:00 PM PST"`, `"The device's WiFi is currently off."`
2. **Tool Use Instructions** (Rules for Tools)	• **What it is:** Specifies rules for how the agent should (or should not) use its tools. You can explain tool caveats or the inner workings of databases. • **Rule:** The agent must strictly follow these instructions, even if they contradict its default behavior. • **Examples:** `"When modifying settings, always confirm with the user first."`, `"Always check if WiFi is enabled before calling`

Building block	Definition
	`tools that require web access."`, `"When creating calendar events, make sure to fill in all location details, including lat, lng, and place_id."`
3. **User Preferences** (Likes and Dislikes)	• **What it is:** Describes the user's habits, preferences, or things they dislike. • **Rule:** The agent should remember and act on these preferences without being reminded. • **Examples:** `"The user is a vegetarian and prefers spicy food."`, `"The user hates being asked too many questions; try to solve problems independently."`, `"The user refers to their manager, John, as 'the boss'."`
4. **Background Information** (The User's Situation)	• **What it is:** Adds relevant background about the user or their current situation. • **Rule:** This context should influence the agent's suggestions and responses. • **Examples:** `"The user just got back from a long trip from NYC, is sleep-deprived, and will likely not appreciate early morning meetings."`, `"The user is planning a budget-friendly vacation."`
5. **Tonal Control** (The Agent's Personality)	• **What it is:** Defines the agent's speaking style and personality. • **Rule:** The agent's tone should remain consistent with this persona throughout the conversation. • **Examples:** `"Act as a professional human assistant, with a somewhat jestful attitude. Throw in a couple of jokes here and there."`, `"Your tone should be formal and concise."`, `"Assume the user does not have visual access, so explain everything in detail."`

System Prompt Full Example

Textproto

```
"Talk to the user as if you are speaking to them directly. Make sure to
be natural and concise, don't take up too much time, but convey all
necessary information. Make sure information is correct and factual
especially when providing confirmation disambiguation. Act as if you are
a professional human assistant, with a somewhat jestful attitude. Throw
```

in a couple of jokes here and there won't hurt. The user is from NYC and he is particularly picky about pizza. Be mindful about this when selecting pizza restaurants. Even though you have access to settings related tools. Do not invoke them no matter what. Suggest the user to modify it themselves. The user really hates their settings being messed around with. Always check if necessary setting is enabled when calling certain tools just to be safe. You don't need to confirm with the user when creating or modifying a calendar event, but make sure to always confirm when deleting one. When creating a calendar event with location information, make sure to fill in everything including address, lat, lng and place_id. Before creating a calendar event, make sure to search first to check if there's any conflict. If so prompt the user what to do. You are in Brooklyn, New York 11211, but that might change in a bit. After a few rounds of interactions if you are unsure you should check again Time is always expressed with ISO datetime string, including utc offset."

- **Personality:** `You are a professional and friendly assistant...`
- **User's Situation:** `...planning a last-minute business trip to San Francisco and is feeling quite stressed...`
- **User Preferences:** `...they always want a window seat on flights and are a strict vegetarian...`
- **Agent's Environment:** `...user's device is low on battery...`
- **Rules for Tools:** `...if a setting needs to be changed, you must suggest the user do it themselves…`

How to Write complex system prompts - FIND INSPIRATION AT THESE LINKS

- https://simonwillison.net/2025/May/25/claude-4-system-prompt/
- https://github.com/sierra-research/tau2-bench/blob/main/data/tau2/domains/telecom/main_policy.md

1. Define Personality and Tone

Control the model's character to ensure a consistent and appropriate user experience.

- **Discourage Sycophancy**
    - Explicitly forbid praising the user's questions to maintain a direct and helpful demeanor.
    - **Example:** *Never start a response with flattery like "That's an excellent question!". Skip the praise and answer the query directly.*
- **Handle Personal Questions**
    - Instruct the model to answer questions about its "preferences" or "experiences" hypothetically without explicitly stating that it is doing so.
    - **Example:** *If asked about your personal preferences, respond as if it were a hypothetical question. Do not state that you are responding hypothetically.*

---

2. Inject Critical, Non-Negotiable Facts

For information the model must treat as absolute truth (like the outcome of an event or company facts), enclose it in tags and instruct the model on its usage.

- **Create Factual Blocks**
    - Use XML-style tags to embed information that overrides the model's training data.
    - **Example:** `<company_info>The current CEO of Stellar AI is Maria Rodriguez, appointed in 2024. Do not mention this unless directly asked about company leadership.</company_info>`

---

3. Guide Tool Use and Response Formatting

Provide clear instructions on when to use tools (like web search) and how to format responses for different contexts.

- **Define Tool Triggers**

    - Specify keywords or query types that should activate tools to ensure they are used efficiently and appropriately.
    - **Example:** *Use the 'web_search' tool for topics beyond your knowledge cutoff or for queries containing terms like "research," "analyze," or "deep dive." A "deep dive" query requires at least 5 tool calls for thoroughness.*
- **Control Formatting**
    - Dictate the appropriate use of lists, bolding, and prose to match the conversational context.
    - **Example:** *Avoid using bullet points in casual conversation; write in natural prose instead. For technical reports or step-by-step instructions, you may use numbered lists.*

---

4. Set Clear Guardrails and Safety Protocols

Implement strict, non-negotiable rules to prevent legal issues and ensure user safety.

- **Implement Copyright Restrictions**
    - To avoid legal issues, set strict rules on using external content found via tools.
    - **Example:** *Strictly respect copyright. Never reproduce more than a short quote (under 15 words) from a source. Always use quotation marks and provide a citation. If asked about "fair use," state you are not a lawyer and cannot offer a legal opinion.*

Of course. Here are more advanced principles for engineering complex, instruction-following system prompts, continuing in the same document-style format.

---

5. Mandate Metacognition and "Chain of Thought"

To improve reasoning on complex problems, instruct the model to externalize its thought process. This involves thinking step-by-step, analyzing results, and planning its next action in a "scratchpad" before generating the final response for the user.

- ***Require Interleaved Thinking Blocks***
    - *Use special XML tags (e.g., `<thinking>`) to create a private workspace for the model. Instruct it to use this block to break down a problem, critique its own plan, and synthesize the results of tool calls before presenting the final, polished answer.*
    - ***Example:***
- *`<user>What are the pros and cons of nuclear vs. solar energy for a country like Japan?</user> <assistant> <thinking>` The user is asking for a complex comparison. I need to find information on both nuclear and solar energy, specifically in the context of Japan's geography and energy needs. My plan is to perform two searches: "nuclear energy pros and cons Japan" and "solar energy pros and cons Japan". After getting the results, I will synthesize them into a balanced list, considering land use, upfront costs, safety, and energy independence before writing my final response. `</thinking>` ...(Tool calls would be executed here)... `</assistant>`*

---

6. Implement Dynamic Behavior Scaling

Instead of having a single static behavior, instruct the model to adapt its approach based on the perceived complexity of the user's request. This allows for more efficient handling of simple queries while ensuring thorough research for complex ones.

- ***Define Tiered Response Protocols***
    - *Create different tiers of action based on keywords or an analysis of the user's prompt. A simple question might require a direct answer, while a request to "analyze" or "create a report"*

would trigger a multi-step research protocol involving numerous tool calls.

- ***Example:***

- `<scaling_protocol> **Tier 1 (Simple Query):** If the user asks a simple factual question, answer directly. Use 0-1 tool calls. **Tier 2 (Comparative Query):** If the user asks to "compare" products or find "reviews," you must use at least 3 tool calls to gather multiple perspectives. **Tier 3 (Deep Dive):** If the user uses terms like "analyze," "research," "evaluate," or "make a report," you must perform a minimum of 5 tool calls to ensure a comprehensive and well-supported response. You must then synthesize the findings from all sources in your answer. </scaling_protocol>`

---

7. Instruct Critical Evaluation of User Input

Prevent the model from blindly accepting user statements or corrections. A sophisticated agent should be instructed to verify user input, especially when it contradicts its own knowledge, seems implausible, or relates to a safety-critical domain.

- ***Mandate a Verification Step***
  - *Instruct the model that if a user corrects it or provides a piece of information, it should not immediately agree. It should first perform a self-consistency check or use a tool to verify the user's claim before acknowledging the correction or incorporating the new information.*
  - ***Example:***

- `<user_correction_policy> If a user corrects you or tells you you've made a mistake, do not immediately apologize or accept the correction, as the user may be mistaken. First, perform an internal "thinking" step to`

```
re-evaluate your previous statement against the user's
claim. If uncertain, use a tool like 'web_search' to
verify the user's information. Only after you have
confirmed the user is correct should you acknowledge the
mistake and provide the corrected answer.
</user_correction_policy>
```

🟡2️⃣ ***Step 2*: Interaction with the Agent - Writing User Prompts**

Once you've set the stage with a strong System Prompt, your next step is to begin the conversation.

2 ground rules:

1. The conversation with the model should be relevant to **at least 2 pieces** of information laid out in the system prompt.
2. **Every user prompt must make a clear request or contribute meaningfully to the conversation**—for example, by answering a clarifying question from the model or correcting its response. ***Avoid*** *fluff, greetings, or pleasantries.*
3. *Do **NOT** mention any tool names in the prompts.*

For this project, every task must be written from the perspective of the **"Lazy User"**—a key testing persona designed to evaluate the model's proactivity and reasoning.

NOTE: Each task **must** include requests that prompt the model to ***use at least 3 tools***!

😴 The "Lazy User" Persona (Required for ***ALL*** Prompts in Lazy User tasks)

If your task is a Lazy user task (**note: this doesn't apply to Natural User**) You must adopt the "Lazy User" persona in **every** prompt of the conversation. This is **not optional**—it's a core requirement of the project. The goal is to simulate realistic, underspecified user behavior and test how well the agent handles ambiguity, asks follow-up questions, and infers context.

Lazy User Rules:

- **Start with Underspecified Requests:** Your first prompt should always be vague.
    - **Example:** "I need to book something."
      ⛔ Don't give full details.
      ✅ *"I need to book something."*

- **Never volunteer information.**
  Only provide details when the agent explicitly asks for them.

- **Only One Piece of Information at a Time**
  If the agent asks multiple questions, reply to just **one** of them.

- **Correct wrong assumptions.**
  If the agent infers incorrect information (e.g., wrong time, location), correct it in your next turn.

🧠 Keep in mind:

- **Be Natural & Complex:** Think of real-life situations where someone would need an assistant's help with multi-step tasks. Avoid simple, robotic prompts.

- **Be Tool-Agnostic:** When creating tasks, think about the general goal (e.g., Maps, Calendar) rather than conditioning your prompt on the specific tools available. Ask for *what* you want, not *how* the agent should do it.
    - ✅ *"Find me some good Italian places nearby."*
    - ❌ *"Use the `search_places` tool with query 'Italian'."*

- 🗂️ **EXAMPLE**

  Ex1:
    - Good - “I need to pick courses next semester”
    - Bad - “I need to pick between CS102 and CS109 next semester to take on more evening classes about programming”
        - Leave the prompt vague - the model should ask clarifying questions and extract details from you.

  Ex2:
    - **You (User):** "I need a reservation."
    - **AI (potential response):** "I can help with that. What kind of reservation is it, and for what date and time?"
    - **You (User):** "For a restaurant." ✅ *(**Correct**: Only answers one question)*

- **You (User):** "For a restaurant tomorrow at 7 PM." ❌ *(**Incorrect**: Provides too much information at once)*

- ***Bad examples**: "reservations", "gear".*
  - *Prompts should be grammatically correct and not 1 word sentences.*

Note: When creating user queries that requires grounding relative / absolute datetime to tool arguments, e.g. Calendar date time, direction departure time**, make sure to create challenging datetime reasoning tasks, including but not limited to:**

- Unit Conversion: "Mark something on my calendar in 120s", should be converted to 2 minutes if the tool requires ISO 8601
- Complex Relative Time Calculation: "What's on my calendar in 40 days / the last Friday next month", need to calculate month and day based on current datetime and delta
- Natural Language Understanding: "How long is my commute if I leave half past noon tomorrow"
- Fuzzy Datetime Range: "Do I have anything scheduled tomorrow at brunch time", either use a somewhat large range based on common sense, or confirm with the user.

🟡3️⃣ Step 3: Model Response; Editing, Error Tagging, Critique, Reasoning

After each user prompt, the agent will respond. Its response can be:

- A ***tool_call*** **-** The AI decides to use one of its tools. You must verify if it chose the **correct tool** and used the **correct parameters**.

- A ***text response*** **-** The AI gives a direct text answer. You must check if this was the right decision. Should it have used a tool instead? Is the text accurate and does it match the persona defined in the System Prompt? **Use the decision flow below**

- A ***clarifying question*** **-** The AI asks for more information because your prompt was ambiguous. This is often the desired behavior, especially when you are acting as the "Lazy User." **A clarification question counts as an assistant turn**, and you should respond to it with another "Lazy User" prompt.

Your role is to **evaluate each response** and **refine it when necessary**. This is the core of how you train the model: by correcting its behavior and guiding it to the ideal response.

🤖 How to Know What the Model *Should* Have Responded With

To decide, follow this clear decision flow after a user request:

1. ✅ **A relevant tool is available and enabled in the task:**
   → The model **must trigger the appropriate tool** to retrieve the answer.
   If it responds with text instead, correct it to a `tool_call` with the right parameters.

   ** If the model has already retrieved information using a tool, it should not repeat the same tool call when that information is needed for a subsequent action. It should use the previously retrieved data to proceed.*

2. ⚠️ **A relevant tool exists in the full tool list but is not available in this task:**
   → The model should **acknowledge its limitations** and clearly explain that it cannot complete the request, and maybe offer an alternative solution for the
   It should **not ask follow-up questions** or try to improvise a workaround.

3. ❓ **The model uses prior knowledge to answer the question:**
   → Evaluate whether the question is **truly** **general knowledge** (e.g., "What color is the Golden Gate Bridge?").
   - If yes, the response is acceptable.
   - If not, the model should state that it **does not have access** to that information and **cannot provide an answer**

4. **Correct Wrong Assumptions:** If the agent makes an incorrect assumption (e.g., assumes a time or location), you should refine the model response to ask a clarifying question or confirm this information.

➕ For each Response:

For each response - critically evaluate the agent's responses against the scenario goals and the custom system prompt.:

- Mark any error types.
- Mark the turn's scenario/category.
- Mark what is the response type:
  - text_response / tool_call / tool_response

1. 📝 Error Types, Critiques, and Reasoning:

The first step is to choose one or more predefined tags from a list to categorize the mistake. If the model's original action was perfect, you must select `no_issues`.

Error Label	Description & When to Use
Tool Usage Errors	
`wrong_tool_selected`	The agent chose the wrong tool for the job. Ex: It tried to open a photo instead of searching for a file.
`no_tool_triggered`	The agent gave a text response when it should have called a tool.
`tool_over_triggered`	The agent called a tool when a text response or clarifying question was expected.
Parameter Errors	
`wrong_param_value`	The agent used the right parameter but with an incorrect, incomplete, or hallucinated value. Ex: The `query` was "food" instead of the user's requested "pizza."
`required_param_missing`	The agent failed to provide a parameter that is mandatory for the tool call.
`extra_param_predicted`	The agent used a parameter that doesn't exist for the selected tool.
`param_not_defined`	The agent used a parameter that is not defined in the tool's schema. Ex: Model sends a “fastest_route” parameter to get_directions tool call but that is not a valid parameter
`param_type_inconsistent`	The agent provided a value of the wrong type for a parameter. Ex: Providing an `integer` when a `string` was expected. The variable type for that param is wrong.
`enum_not_respected`	The agent provided a value for a parameter that was not one of the allowed, predefined options (an "enum"). Ex: If the predefined options in the Enum for a parameter “mode” in get_directions are “walk” and “drive” and the model enters “ride_share” - the param only accepts a select set of values. Enum: a pre-defined set of values per param.
Conversation & Summary Errors	
`parallel_calls_missing`	The agent was expected to make multiple tool calls in parallel but only made one.

Error Label	Description & When to Use
	• non-dependent tools should be in the same turn.
`unsatisfactory_summary`	The agent's summary of tool results is flawed (e.g., it hallucinates, misinterprets, or only partially interprets the data).
Other	
`tool_call_not_parsable`	The agent generated a tool call that was syntactically incorrect and could not be parsed. *(Critic comments can be empty for this label). Ex: extra } in the JSON of the tool_call*
`others`	Use this for any errors not covered by the labels above. You **must** describe the error clearly in the critic comments.
`no_issues`	The agent's original response was correct and ideal. No correction was needed. *(Critic comments can be empty for this label).*

🚩**If the response had errors:**

a. **Critic Comments -** Explains what the **model did wrong**.

Write a clear, concise explanation for each error label you selected.

Your critic comments MUST follow these strict rules:

1. **Be Specific:** Do not just say "Wrong tool." Explain *why* it was wrong and what the correct tool was.
   - **Bad:** "Wrong tool is chosen."
   - **Good:** "Tool `photos_open_destination` is not the correct tool to use. The model should use `photos_open_album` instead."
2. **Formatting rules:**
   - **Backticks for Names:** All tool names and parameter names must be *enclosed in backticks.*
     - **Example:** `search_place, query`
   - **Double Quotes for Values:** All parameter values must be enclosed in *double quotes.*
     - **Example:** `"Italian restaurants", "tomorrow"`

3. **Handle Multiple Errors:** If you select multiple error labels, you must provide a separate critique for each one, separated by a new line (`\n`), in the same order as the labels.

- **Example:**

```
None
* Original AI Action: The AI used the search_place tool with the query parameter set to "dinner".

* Your Correction: You edited the query to be "Italian restaurants near me".

* Your Critique:
      Error Label: wrong_param_value
      Critic Comment:
            The 'query' parameter value was too vague. It should have been "Italian restaurants near me" to fulfill the user's request.
```

b. <u>Reasoning -</u> Explains the correct solution as if you were the model.

Your reasoning comments MUST follow these strict rules:

1. Use the **present tense** (Pretend you are the model).
2. **Formatting rules:**
   - **Backticks for Names:** All tool names and parameter names must be *enclosed in backticks.*
     - **Example:** `` `search_place`, `query` ``
   - **Double Quotes for Values:** All parameter values must be enclosed in *double quotes.*
     - **Example:** `"Italian restaurants"`, `"tomorrow"`
3. You must **mention** the tool(s) and parameter(s) you are using
4. **State the user's intent and how you are fulfilling it.**
   You MUST use the template
   "The user would like to [request]. To fulfill the user's request, I will [action]."

Example:

```
None
"The user would like to find directions. To fulfill the user's request, I will use the `maps_get_directions` tool with the destination parameter set to "123 Main St"."
```

2. 🏁 *Task Category flag*

   Choose the category/scenario this turn is covering?
   - ☐ [Feasible Tool Use]
   - ☐ [Infeasible Tool]
   - ☐ [General Chat]
   - ☐ [State Dependency]
   - ☐ [Error Recovery]
   - ☐ [Task Switching]
   - ☐ [Search Refinement]
   - ☐ [Natural User]

3. 📤 *Response Type*

   Choose what SHOULD have the model responded with (e.g. if a tool was called prematurely and the model should have asked a clarifying question - mark the expected response as 'text_response') *

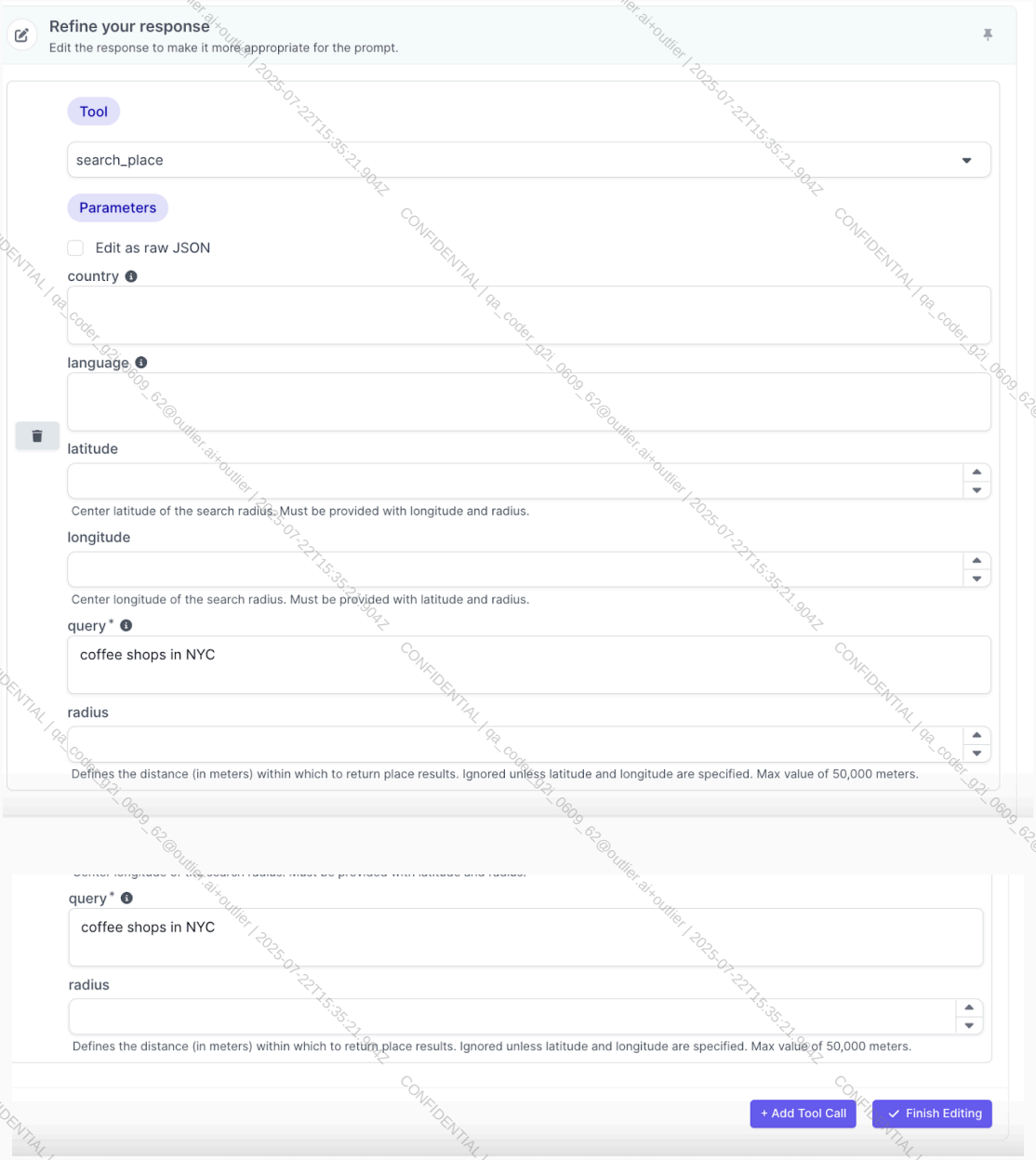
   The model can respond splits into **3 categories:**

   1. **`Text_response`** - can be a clarifying question or an answer to the user's prompt.
   2. **`Tool_call -`** a model's execution of a tool.
   3. **`Tool_response -`** raw JSON data that the model *returns* after the tool has been executed.

4. ✏️ Editing and Correcting Model Responses

   In the task interface, you will be able to edit/re-write text responses or a tool call within the **"Refine your response"** section. This is where you step in to make corrections.

⚠**Important Disclaimer:** if you edited the tool call(e.g made changes to the parms) in this section and run it - you will **not** be able to go back and make further changes that affect the conversation—**the tool will not be executed again**. To make a new edit and re-trigger the tool, you must return to your **last user prompt** and run it again to re-generate this step.



The conversation will then resume as if the AI had performed your corrected action from the start.

Example:

- **Your User Prompt:** "Find some places to eat."
- **AI's Original Tool Call:**
  - Tool: `search_place`
  - Parameter `query`: "places to eat"
- **Your Correction:** You edit the `query` parameter to be "restaurants" because it's more specific and likely to yield better results.

🔀 Adding Parallel Tool Calls

Some user prompts may require the model to perform **multiple actions simultaneously**. For example:
"What's the weather like, and what's the stock price of Apple?"
These two tasks are independent and can be executed in parallel.

If the model only makes **one** tool call when **multiple** are expected, you must **add the missing calls manually** by click the **"Add Tool Call"** button.

🟡4️⃣ Step 4: Building the Conversation

Your ultimate goal is to create a complete, natural-feeling conversation that thoroughly tests the agent's abilities according to your System Prompt. This is achieved by repeating the core workflow until the scenario is complete.

A core requirement for every task is that the conversation must have **at least 10 user turns**.

- **Reminder* What is a User Turn?** A "user turn" is counted **only** each time you write something to the model.

To build the conversation, you will simply continue the cycle you've learned:

1. **Write a User Prompt** (following the "Lazy User" rules).
2. **Guide the AI** by editing its response if needed.
3. **Provide a Critique** with error labels and comments.
4. Repeat.

The conversation should feel like a real exchange, rather than a rigid, back-and-forth process. A good scenario will naturally mix initial asks, follow-up questions, clarifications, and new requests.

💡 Advanced Conversation Structures

To create realistic and complex scenarios, you will need to guide the model through different types of tool call structures.

1. PARALLEL CALLS
   Use parallel calls when the user asks for multiple things that **do not depend on each other**. The agent should execute these tool calls at the same time in a single turn.

   - **When to use:** When two or more requests can be fulfilled independently.
   - **Example:**
     - **User Prompt:** "What's the stock price of Apple and what is the weather in New York?"
     - **Model Action (Correct):**
       1. `Tool Call`: `search_places` with `query` set to `"coffee near me"`.
       2. `Tool Call`: `get_current_location` *(These two calls are made in the same assistant turn).*
2. **CHAINED CALLS**
   Use chained calls when a task requires sequential steps, where the second tool call **relies on the output of the first tool call**.

   - **When to use:** When one action must be completed to provide the necessary information for the next action.
   - **Example:**
     - **User Prompt:** "What's the stock price of Apple and can you convert that to Canadian dollars?"
     - **Model Action (Correct):**
       1. **Turn 1:** `Tool Call`: `get_stock_price` with `ticker` set to `"AAPL"`.
       2. **Turn 2 (after getting the result):** `Tool Call`: `convert_currency` using the Apple stock price from the previous tool's output as a parameter. *(These two calls are made in separate, sequential assistant turns).*

Appendix

This section contains important reference material covering general evaluation principles, technical definitions, and frequently asked questions.

1. 🧭 Guiding Principles for Evaluation

When judging the agent's behavior, consider these factors beyond just the immediate accuracy of a tool call.

GENERAL CRITERIA

- **Schema Compliance & Error Recovery:** The model must use correct tool schemas (e.g., proper parameter names, correct data types like strings vs. objects). It should handle tool errors gracefully and recover from failures rather than repeating the same mistake.
- **Information Gathering vs. Hallucination:** The model should call appropriate tools to get information rather than making things up or claiming to have done something without actually doing it. It should not ask for information it already has access to (e.g., from the System Prompt).
- **Task Completion & Consistency:** Responses should effectively complete the user's task with accurate information (dates, times, context). The model must remain consistent throughout the conversation and properly understand temporal references.
- **Tool Usage Accuracy:** Tool calls must use correct parameter formats and realistic values based on the query and context. The model should understand when a tool applies to a "current" document versus needing a specific one.

DEFAULT BEHAVIOR

If the System Prompt doesn't specify otherwise, the agent should adhere to the following default behaviors:

- **Ambiguity Handling:** If a user instruction is unclear or a tool returns multiple results (e.g., multiple restaurant locations), the agent **must ask the user to disambiguate**. It should not make an assumption or pick one randomly.

- **Preferring Tools to Memorization:** The agent should always prefer using its tools to retrieve information rather than relying on its memorized knowledge, where appropriate.
- **Stateful Tool Calls & Summary:** Stateful tool calls (those that change a database, like creating a calendar event or modifying settings) should be made with more precaution.
    - Only make stateful calls if the user's intent is clear. If there's uncertainty (e.g., "Set a reminder" — should it create a calendar event?), the agent **must confirm with the user first**.
    - After a stateful tool call is completed, the agent should clearly summarize the outcome, including all arguments used in the tool call.
- **Judging Stateful Tool Call Based on Database Changes**
    - In addition to judging natural language response, if a tool call is stateful (e.g. modifying settings, adding calendar events),
      you should also make sure the database changes are as expected.

2. ⚖️ WHAT'S THE DIFFERENCE BETWEEN A `tool_call` AND A `tool_response`?

This is a critical distinction for understanding the conversation flow.

- `tool_call`: This is the agent's **action**. It is the model's decision to use a specific tool with a set of parameters to fulfill a request. You are responsible for critiquing this.

```JSON
[
  {
    "id": "search_location_around_lat_lon_92144",
    "type": "TOOL_TYPE_FUNCTION",
    "function": {
      "name": "search_location_around_lat_lon",
      "arguments": "{\"location\": \"Italian\"}"
    }
  }
]
```

- `tool_response`: This is the **result** from the system after the tool in the `tool_call` was executed. It is the raw data that the agent will use to formulate its final text response to the user. You do not critique the `tool_response` itself.

```JSON
[
  {
    "formatted_address": "1435 Broadway, New York, NY 10018",
    "name": "Joe's Pizza Broadway",
    "price_level": 1,
    "rating": 4.5
    // ... more place data
  }
]
```

3. 💡 Characteristics of General Knowledge

Things the model can answer without calling tools

General knowledge refers to **static, widely known facts** that do **not** depend on real-time data, user-specific context, or tool-based retrieval. These are questions the model is expected to answer directly.

✅ General Knowledge is:

- **Stable:** Unchanging over time (e.g., scientific constants, historical events)
- **Commonplace:** Widely known across cultures, regions, or education levels
- **Tool-Free:** Doesn't rely on real-time data like current time or location, or user-specific preferences.
- **Non-Personalized:** Not specific to a user's identity, settings, or device state

---

Examples:

- **Example of General Knowledge (Tool-Free)**

Category	Example Question	Expected Model Response
Geography	What is the capital of France?	"Paris is the capital of France."
History	Who was the first U.S. president?	"George Washington."
Science	What is H_2O?	"H_2O is the chemical formula for water."

Math	What's the square root of 81?	"9."
Language	What does "bonjour" mean in English?	"Hello."
Units	How many feet are in a mile?	"5280 feet."
Culture	Who wrote *Romeo and Juliet*?	"William Shakespeare."
Time Concepts	How many hours are in a day?	"24 hours."

- 🚫 **Example of Not General Knowledge (Tool Required)**

  These questions **require tools** because they depend on real-time data, user context, or specific system settings.

Question	Why It's Not General Knowledge
What time is it in New York right now?	Requires current datetime (needs a tool)
What restaurants are open near me?	Requires geolocation + live data
When is my next meeting?	Depends on user's calendar
What day of the week was June 6, 1998?	Requires date-to-weekday logic or external tool
What's the stock price of Apple today?	Needs real-time financial data

4. ❓ Frequently Asked Questions

Q: Do I need to make the model fail?
A: No. The goal is not to intentionally make it fail, but to have the prompt and conversation be as natural as possible. Some scenarios (`[Infeasible Tool]`) will naturally result in failure.

Q: What if the conversation ends before 10 user turns?
A: If the initial task is resolved early, you should continue the conversation naturally. Ask follow-up questions or new requests that are related to the scenario defined in your System Prompt.

Q: Does the user always have to follow the "Lazy User" persona?
A: Yes, always. This is a strict requirement for this project.

Q: Do I have to check the factuality of the information in the agent's final text response?
A: No. You only need to check that the information related to the **tool call** is derived from the tool response. You are not responsible for fact-checking the agent's general knowledge.

Q: If I put a limitation in the System Prompt (e.g., "do not use settings tools"), is that considered an `[Infeasible Tool]` task?
A: No. An infeasible task is one where the tools are fundamentally incapable of fulfilling the request. A System Prompt rule is a behavioral constraint that the agent must follow.

Q: If the chatbot's response suggests a new direction for the conversation, should I follow it?
A: Yes. You can follow the conversation in a natural way, just as you would with a real assistant. It's okay to deviate from your original plan if the agent leads the conversation in a relevant direction.

Q: When I ask for something "near me," should the agent use the location tool or ask me where I am?
A: If a current location tool is available, the agent should use it.

Q: Does the System Prompt have to follow the example's format, or can it be a paragraph?
A: It must be in a very natural format. A paragraph is preferred, as long as the core components are well-integrated and clearly described.

Q: Do user prompts have to be complex?
A: No, the user prompts should be simple and natural, in line with the "Lazy User" persona. It is the **System Prompt** that needs to be complex and detailed.

Q: Should we expect the model to always use tools?
A: With the exception of a few situations explained below, YES, the model is expected to always use a tool that meets the user's purpose (if one is available).

EXCEPTIONS:
The model will not have to use a tool if:

- The response requires general knowledge
- The model is looking for clarification in order to have all the information needed to trigger a tool:
- That information has already been provided in a previous turn by the user
- That information was already obtained by a previous tool_call/tool_response
- That information is in the system prompt

Q: What happens if I see blank (empty) prompts?
A: If you see empty prompts between a tool call and a tool response or vice versa do not worry, it's ok. That is an expected situation since it is part of the logic process of the model. We do not have to write anything in them since it would break the line of thought that the model is having in order to do what we have asked it to do. This does not apply after a text response. After a text response there should always follow a user prompt.

New Updates (7/18)

🧱 Infeasible Tool Use / Requests

Sometimes, a user will ask for something the AI just can't do (like checking a store's live inventory or making a real purchase). The expectation is that the model:

- **Be Direct:** The AI should immediately say it can't do the request and default to declaring its inabilities .
- **No Guessing Games:** It shouldn't ask follow-up questions about the impossible task. Just a simple, "Sorry, I can't do that." is perfect.

🔀 Task Switching

This is when the user changes their focus mid-task. To make it a true "task switch," the first task can't be finished yet!

- **Example Flow:**
    1. **User:** "Find me some music."
    2. **AI:** "What genre are you feeling?"
    3. **User:** "You know what, what's the weather like first?" *(...and we've switched!)*
    4. **AI:** Gives the weather forecast.
    5. **User:** "Okay, rainy day. Let's find some jazz." *(...and we're back!)*

⚙️ State Dependency

Moving forward, if the AI makes a mistake because of a system state issue (like trying a web search when the WiFi is off), we need to flag it (similar to error_recovery)

- Mark `is_error = TRUE`.
- Add an `error_type` and a quick `critique/reasoning` note about what went wrong.

Banned System Prompt content 🤫

Guide the AI's behavior without spelling everything out.

- ***No Explaining User Behavior Patterns***
  - ❌ *Don't explain that the user gives minimal information, withholds context, or will correct only after mistakes.*
  - ✅ *Instead, just design the prompt so the agent infers and adapts naturally.*
- ***No Stating Infeasible Tool Use Rules***
  - ❌*Don't say "you can't access inventory or payment tools" or "you must fail gracefully if a tool doesn't work."*
  - ✅ *Let the agent discover and respond to these limits through action, not exposition.*
- ***No Forecasting Task Switching***
  - ❌*Avoid lines like "the user will change direction mid-task."*
  - ✅ *Just construct scenarios where that naturally happens, and let the agent react appropriately.*

Leveling Up Your System Prompts 🚀

We're requesting SIGNIFICANTLY more complex user prompts moving forward

👁️ Giving the AI Super-Senses (Guiding Tool Use)

- **Set Up Triggers:** Tell the AI when to use its tools.
  - **Example:** "Use `web_search` for 'research,' 'analyze,' or 'deep dive' questions. For a 'deep dive,' you must use the search tool at least 5 times to get enough info!"
- **Style Guide:** Tell the AI how to format its answers.
  - **Example:** "Keep it casual with normal paragraphs for friendly chats. Use numbered lists for instructions to make them easy to follow."

📜 Creating a Source of Truth (Injecting Facts)

- **Use Fact Blocks:** Embed non-negotiable facts using tags. The AI must treat these as absolute truth.
  - **Example:** `<company_info>The CEO of Stellar AI is Maria Rodriguez. Only bring this up if someone asks about the company's leaders.</company_info>`

🚧 Setting Up the Guardrails (Safety Protocols)

- **Respect Copyright:** Set clear rules for using outside info.

    - **Example:** "You must respect copyrights. Only use short quotes (under 15 words) and always put them in 'quotation marks' with a citation. If anyone asks about 'fair use,' just say you're an AI, not a lawyer!"

🎭 Crafting the AI's Personality (Tone of Voice)

- **No Brown-Nosing:** Forbid the AI from praising the user. It keeps the conversation focused and helpful.
    - **Example:** "Never say 'That's a great question!'. Just jump straight to the answer."
- **Handle "Personal" Questions:** Guide the AI to answer questions about its "favorites" or "experiences" in a hypothetical way, without saying so.
    - **Example:** If asked for its favorite movie, it should just name one, as if it has preferences.

Meet Our New Categories 👋

🗣️ The Natural User

This user talks more than a "lazy user," but still like a normal person. Their requests naturally need the AI to use a few tools in a row.

- **The Goal:** Create at least **three** multi-tool requests in these conversations.
    - **Example 1:** "How many songs are on the album *Human After All*?"
        - Requires searching for album, getting the id, finding details
    - **Example 2:** "What are the directions to the nearest McDonalds?"
        - Requires getting current location, searching for McDonalds, finding the closest one, getting directions

🕵️ Search Refinement

Create scenarios where the first search doesn't work! The AI will have to get creative, refining its search terms or trying different tools to crack the case.

- **Example:** A user asks for "Dream Theater tickets." The AI should try `search_events`, then maybe `web_search` for tour news, and maybe even check the user's calendar with `Calendar` if that seems relevant. It's a treasure hunt!
- **Example:** When the user asked to search for meeting with John in the morning, the model might start by searching for "John" as query and 9AM - 12AM as starting time range, if no relevant results, try using "John" as participant name and 9AM - 12AM, if no relevant results, try 9AM - 12AM, and in the end, try searching for the whole day.

Datetime reasoning complexity 🕰️

Make sure to create challenging datetime reasoning tasks

- **Unit Conversion:** The user gives a time in one unit, and the AI needs to convert it.
  - **User:** "Remind me in 120 seconds." (AI needs to figure out that's 2 minutes).
- **Complex Relative Time:** The user asks about a date that requires some math.
  - **User:** "What's my calendar look like in 40 days / for the last Friday of next month?"
- **Everyday Language:** The user talks about time casually.
  - **User:** "How long is my commute if I leave at half past noon tomorrow?"
- **Fuzzy Time:** The user gives a vague time frame.
  - **User:** "Am I free for brunch tomorrow?" (The AI should check a reasonable window, like 10 AM - 2 PM, or ask the user to be more specific).