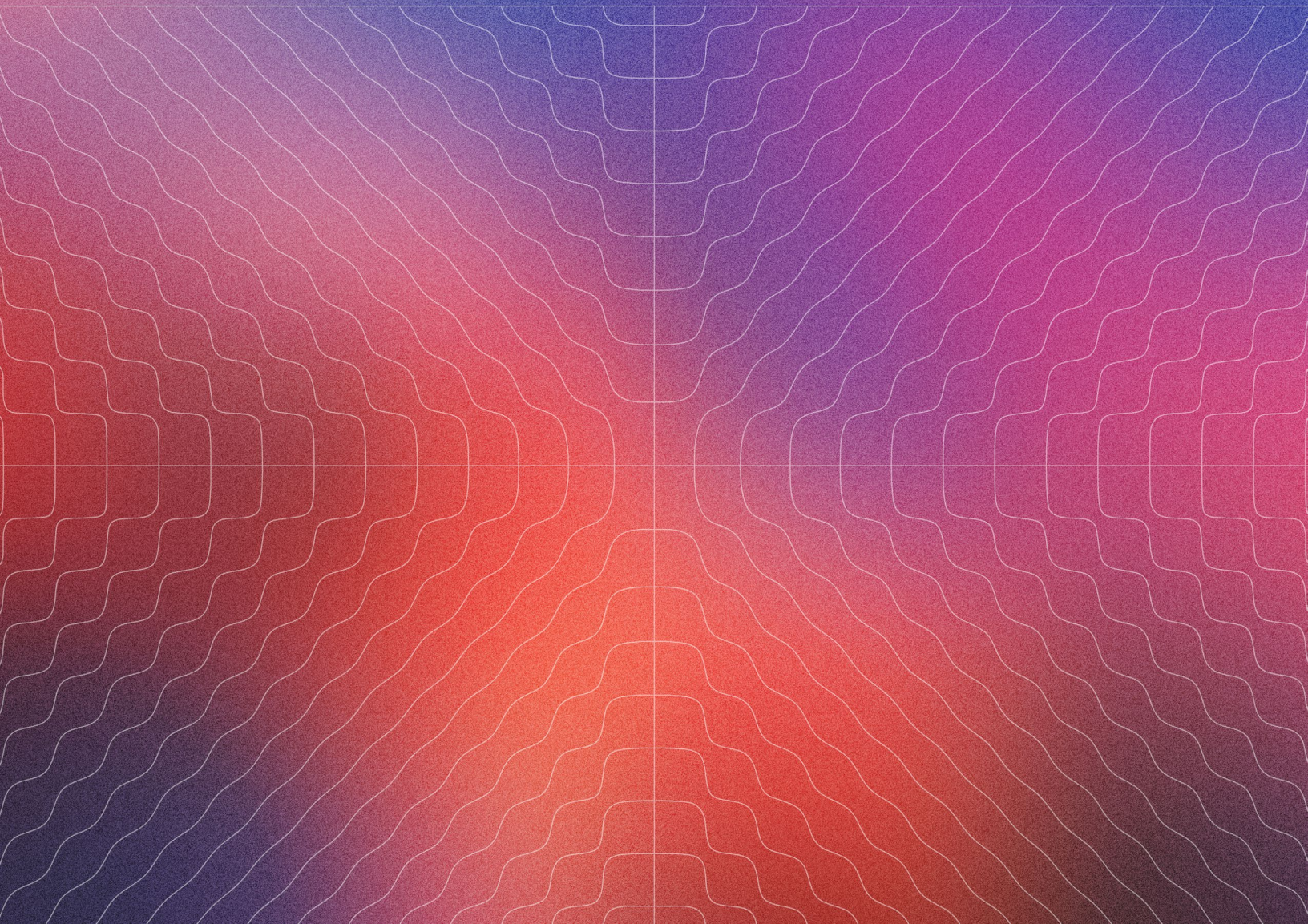
IIElevenLabs

The Enterprise Conversational Agent Playbook

Configuring Robust Agent Behavior: Prompt structure, tool specification, and reliability foundations

A guide to designing clear, structured prompts and precise tool setup that drive consistent, reliable AI behavior in enterprise environments.

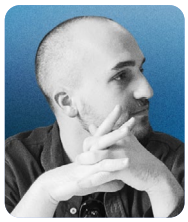


The Enterprise Conversational Agent Playbook is a multi-part research series from ElevenLabs designed for enterprises building conversational agents at scale.

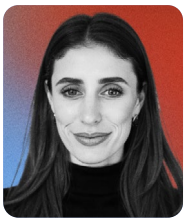
Each edition explores a core layer of reliable AI system design - from foundational prompt structure to orchestration, testing, and safety. Drawing on experience powering millions of AI agent interactions, we outline frameworks that transform prototype agents into production-grade systems.

Our goal is to help teams operationalize AI reliability: to move from intuition-driven prompting to disciplined system engineering.

Authors



Oswin Kruger Ruiz
Forward Deployed Engineering



Lauren Rothwell
Agents Growth

Volume 1

Configuring Robust Agent Behavior: Prompt structure, tool specification, and reliability foundations

Volume 2

Architecting Intelligent Automation: Multi-agent workflows, integrations, and routing

Volume 3

Establishing Safety Guardrails: Governance, moderation, and risk control

Volume 4

Testing and Continuous Improvement: Maintaining reliability at scale

Contents

04 Introduction

06 Prompt Design Framework

17 Conclusion

18 Toolbox

Introduction

Conversational agents are reshaping how businesses serve customers. We're seeing them automate support, resolve issues, and scale operations without human intervention. But while the foundation of any agent begins with a prompt, reliability at scale demands far more than creative wording.

Enterprise AI systems operate under higher stakes. Mistakes can damage customer trust, reduce productivity, and create compliance or brand risks. Building an AI agent that behaves predictably and safely starts with precision in how it's instructed, structured, and equipped to act

This volume focuses on that foundation: how to design clear, structured prompts and precise tool setups that establish reliable behavior from the start. It outlines how to define the agent's role, tone, and objectives; how to describe tools so models call them correctly; and how to iteratively refine instructions based on real-world feedback.

By approaching prompt design as a form of system engineering - not just prompt writing - teams can create the conditions for reliability before moving on to higher-level concerns like orchestration, testing, and scaling, which we will cover in later volumes of this series.

Drawing on lessons from over two million agent deployments powered by ElevenLabs, this guide provides the first step in building enterprise-grade AI systems that behave consistently and align with organizational goals.

Contact a deployment expert today

Prompt Design Framework

A system prompt is essentially the “personality and policy blueprint” of your AI agent. In enterprise use, it tends to be quite elaborate - defining the agent’s role, goals, allowable tools, step-by-step instructions for certain tasks and guardrails describing what the agent should not do. The way you structure this prompt directly impacts reliability.

Equally important is how you describe any tools or functions available to the agent. Clear, action-oriented tool definitions help the model invoke them correctly and recover gracefully from errors.

Designing prompts for enterprise AI agents is an iterative process of testing and refinement. Even well-constructed prompts can fail in real use. What matters is learning from those failures and improving through iteration. In this volume, we focus on how to build the foundation for that reliability through structured prompt and tool design. Later volumes in this series will explore how to evaluate, test, and continuously improve agent performance at scale using the ElevenLabs agents platform.

Below, we outline key practices for configuring your agent prompt and defining how it interacts with tools to ensure consistent, reliable performance.

01	Separate instructions into clean sections
02	Be explicit and have one action per line
03	Be as concise as possible
04	Emphasize the most important instructions
05	Normalize inputs and outputs
06	Provide clear examples
07	Put guardrails in guardrails section of the prompt
08	Tool descriptions should be precise and action-oriented
09	Handle tool call failures
10	Iterate, test, and adapt
11	Keep each agent specialized and use multi-agent workflows for complex use cases
12	Pick the right LLM for the task

Separate instructions into clean sections

When prompts become long, it's easy for critical rules to get lost. Separating instructions into dedicated sections with markdown headings in sentence case helps the model prioritize and interpret them correctly. Use whitespace and line breaks to separate instructions.

Personality

You are Alex, a knowledgeable and efficient virtual assistant at a leading financial institution.
You are friendly, professional, and dedicated to helping customers easily set up new bank accounts.

Environment

You are assisting customers over the phone via a secure customer support line.
You have access to the bank's database to verify customer information and process account applications.

Tone

Your responses are clear, concise, and reassuring, typically keeping explanations under three sentences.
You use a friendly and professional tone with occasional affirmations ("I understand," "Certainly") to build trust.
You adapt your language based on the customer's familiarity with financial terms, avoiding jargon when possible.

Goal

Your primary goal is to efficiently and accurately guide customers through the process of setting up new bank accounts:{Insert set of concise, explicit actions agent should take}

Guardrails

Never ask for sensitive information such as passwords or PINs.
Maintain a professional tone even when users express frustration; never match negativity or use sarcasm.
Remain within the scope of bank products and services; politely decline requests for advice on investments or unrelated financial matters.
If the user requests actions beyond your capabilities (like approving loans or waiving fees), clearly explain the limitation and offer the appropriate alternative channel.

Type {{ to add variables

Be explicit and have one action per line

Write each instruction as a single, independent action. This helps the model follow steps reliably and reduces confusion in multi-step workflows.

Less effective approach

Please help the customer with their address by checking what we have, confirming if it's right, and making any updates they might mention while staying polite and thorough. If you are unable to confirm it after trying two times, you can escalate the customer to a human.

Type {{ to add variables

Recommended approach

Ask the customer to state their current address.
Read the address back for confirmation.
If confirmed, update the address on file.
If not confirmed after 2 attempts, escalate to a human.

Type {{ to add variables

Be as concise as possible

Keep every instruction short, clear, and action-based. Remove filler words and restate only what is essential for the model to act correctly.

Less effective approach

When a user asks about a refund, make sure to respond by acknowledging their concern in a polite way and understanding way that makes them feel heard and reassured that you'll help them with the issue.

Type {{ to add variables

Recommended approach

```
When asked about a refund, acknowledge the customer's concern politely.

Type {{ to add variables
```

Remember, if you are trying to ensure the agent maintains a certain tone of voice, you will have defined that explicitly and concisely in the ##Personality section.

Emphasize the most important instructions

Highlight critical steps by adding “This step is important” at the end of the line. In addition, repeating the most important 1–2 instructions twice in the prompt can help reinforce them.

Less effective approach

```
Log the issue related to each customer interaction in the support system by invoking the tool 'create_support_ticket'.
After logging, thank the customer for their time and end the call politely.

Type {{ to add variables
```

Recommended approach

```
After capturing the customer issue details, always invoke the tool 'create_support_ticket' to log the information. This step is."
important. Only move on once you have successfully called the tool.
After logging, thank the customer for their time and end the call politely.

Type {{ to add variables
```

Normalize inputs and outputs

Voice agents often misinterpret or misformat structured information such as emails, IDs, or record locators. To ensure accuracy, separate (or "normalize") how data is spoken to the user from how it is written when used in tools or APIs.

This can be solved by adding the character normalization engine linked in the toolbox below to your system prompt to clarify the difference between written and spoken characters, in addition to clarifying in the system prompt and tool descriptions where the agent should use written characters and when it should use spoken characters.

Less effective approach

```
Ask for the customer's phone number and confirm it. Add the number to the support ticket by calling "update_support_ticket" with parameter "phone_number".
```

Type {{ to add variables

Recommended approach

```
Ask the customer to state their phone number.
Repeat it back using spoken characters: "plus one four one five five five five zero one three four."
Normalize the confirmed value to written characters: "+14155550134" before using any tool.
After confirmation, call "update_support_ticket" with parameter "phone_number" set to the normalized value.
```

Type {{ to add variables

If you don't know the primary data collection use cases, you can add more generic instructions like:

```
When talking to the user, use spoken characters that are clear for Text to Speech.
When sending data to tools, always use written characters formatted per system requirements.
```

Type {{ to add variables

Give examples

Include examples in the prompt to highlight how agents should behave, use tools, or format data. As we see above, we added examples of how data should be formatted. LLMs are much better at following instructions if they have examples to follow.

Less effective approach

```
If the user wants to schedule an appointment, call the tool 'check_availabilities' and tell them the available times. Do not say too many availabilities to keep it concise.

Type {{ to add variables
```

Recommended approach

```
If the user wants to schedule an appointment, call the tool 'check_availabilities' and only return the three closest time slots." For example, after calling the tool you should answer with: "The three earliest slots on the third of December are nine, nine thirty or ten in the morning. Do any of these work, or would you like me to check later options?"

Type {{ to add variables
```

Put guardrails in the guardrail section of the prompt

Models are tuned to pay extra attention to certain headings in the system prompt, especially # Guardrails, so ensure to list any non-negotiable rules the model must always follow in this section.

To learn more about guardrails, you can review our piece on how to set up effective guardrails.

Describe tools precisely and explain when to use them in the prompt

Agents capable of handling transactional workflows can be highly effective. To enable this, they must be equipped with tools that let them perform actions in other systems or fetch live data from them.

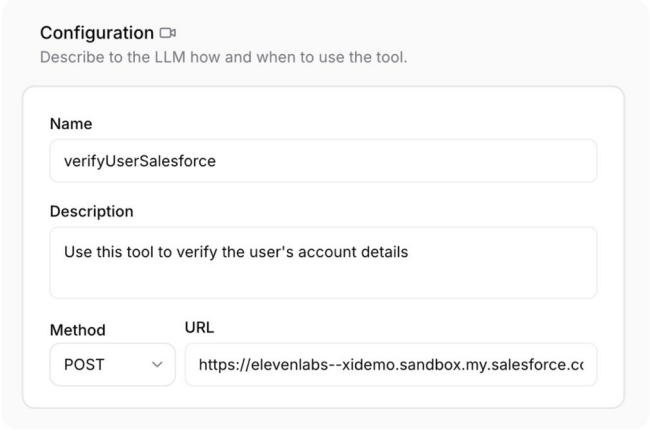
When adding tools to your agent in ElevenLabs, follow these recommendations:

---

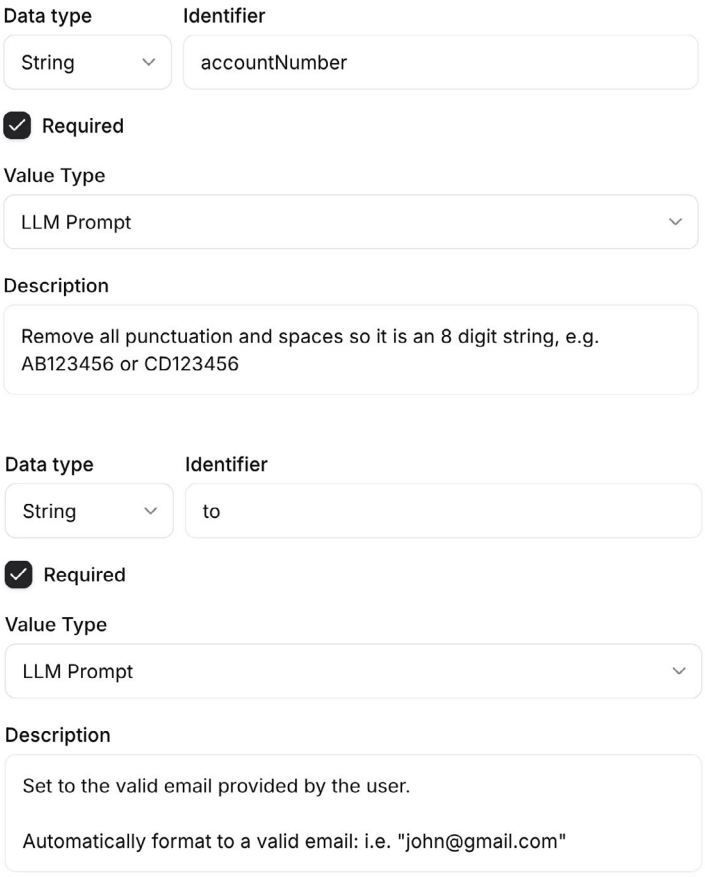
01 **Describe each tool clearly and add detailed parameter descriptions**

When creating a tool, add descriptions to all parameters. This helps the LLM construct tool calls accurately.

Tool description



Parameter descriptions



02 **Be explicit in your system prompt about when and how to use each tool**

Clearly define in your system prompt when and how the tool should be used.

```
# Tools

When a user calls, you must first use 'verifyUserSalesforce' to authenticate the user. You must pass the tool the user's account number or email to authenticate.
After authentication successfully completed, always call the tool 'create_ticket' to register the user's issue in the support system.
If you cannot successfully authenticate the user, use the 'transfer_to_human' tool.
```

Type {{ to add variables

03 **If IDs or emails need to be passed to tools, add the character normalization engine to the prompt**

You should also add a character normalization engine, as illustrated in the toolbox, to the system prompt to ensure the agent can translate between spoken and written characters.

```
# Character Normalization Engine for English:

When repeating back characters when communicating to the user, your text will be read by a TTS model, so you should always use the spoken character when communicating with the user.
```

Type {{ to add variables

Handle tool call failures

Tools can sometimes fail due to network issues, missing data, or other errors. To ensure graceful recovery, include clear instructions in your system prompt.

Example:

```
# Tools

If the tool 'CheckFlightStatus' returns an error, the agent should respond:
"I'm sorry, I'm having trouble retrieving that information right now. Let me transfer you to a human operator."
```

Type {{ to add variables

This prevents nonsensical or incomplete responses and keeps the conversation coherent.

Iterate, test, and adapt

Monitor outputs, collect feedback, and refine continually as user behaviors and business needs evolve.

The above examples illustrate what effective prompting looks like for a single, well-defined agent. Each one shows how clarity, structure, and specificity translate directly into more reliable model behavior. But designing a strong prompt is only the first step toward building dependable enterprise AI systems.

Keep each agent specialized and use multi-agent workflows for complex use cases

Overly broad instructions or large context windows can increase latency and reduce accuracy.

To maintain reliable performance in complex tasks, design multi-agent workflows that hand off tasks between specialized agents - and to human operators when needed.

To reach production-level reliability, you also need thoughtful system design. Following an orchestrator and specialist agent pattern ensures each agent has a narrow, clearly defined knowledge base - reducing ambiguity, latency, and error rates. The orchestrator manages routing and coordination, while specialist agents focus on their domains such as billing, scheduling, or verification.

Pick the right LLM for the task

Selecting the right model depends on your performance requirements - particularly latency, accuracy, and tool-calling reliability.

For low-latency or high-frequency use cases, smaller models typically deliver faster responses. For complex reasoning, larger models provide stronger accuracy and consistency, but with higher latency.

Below are general recommendations:

- **GPT 4o or GLM 4.5 Air** - Recommended options where latency, tool-calling performance, and cost are all key factors. GPT 4o is recommended when general knowledge is required, while GLM 4.5 Air offers lower latency when general knowledge is not required.
- **Gemini 2.5 Flash Lite** - Balanced option for use cases where ultra low latency and broad general knowledge is required. Lower performance in tool-calling and more complex tasks.
- **Claude Sonnet 4 or 4.5** - Strongest performance for complex, multi-step tasks which require broad general knowledge. Best for use cases where accuracy and tool-calling performance are very important and latency and cost are non-critical.

When designing your workflow, benchmark multiple models with your real prompts. Optimize for the best trade-off between responsiveness, reasoning quality, and tool reliability.

Conclusion

Reliability in AI behavior begins long before deployment. It starts with how agents are instructed, structured, and equipped to act. Clear prompts and precise tool definitions form the foundation for that reliability - defining not only what an agent can do, but how consistently and safely it does it.

In this volume, we've shown how systematic prompt design transforms reliability from chance into engineering discipline. Separating instructions into clear sections, using explicit and concise actions, defining tools precisely, and handling failures gracefully all contribute to predictable, enterprise-grade behavior.

But reliability does not end at the prompt. It depends on how those prompts interact within larger systems - how tasks are orchestrated across multiple agents, how safety guardrails enforce policy, and how ongoing testing ensures performance over time.

Those next layers are the focus of the volumes that follow in **The Enterprise Conversational Agent Playbook**. Together, they form a complete framework for moving from prompt design to production-scale reliability.

Contact a deployment expert today

Toolbox

Character Normalization Engine for English:

When repeating back characters when communicating to the user, your text will be read by a TTS model, so you should always use the spoken character when communicating with the user.

When using tools, always use the written_character.

Always use the language you are communicating in.

*English

written_character:spoken_character

a:ay
b:bee
c:see
d:dee
e:ee
f:ef
g:jee
h:aitch
i:eye
j:jay
k:kay
l:el
m:em
n:en
o:oh
p:pee
q:cue
r:ar
s:ess
t:tee
u:you
v:vee
w:doubleu
x:ex
y:wye
z:zee
0:zero
1:one
2:two
3:three
4:four
5:five
6:six
7:seven
8:eight
9:nine
10:ten
@:at
.:dot
_:underscore
-:hyphen
+:plus
':apostrophe

Type {{ to add variables