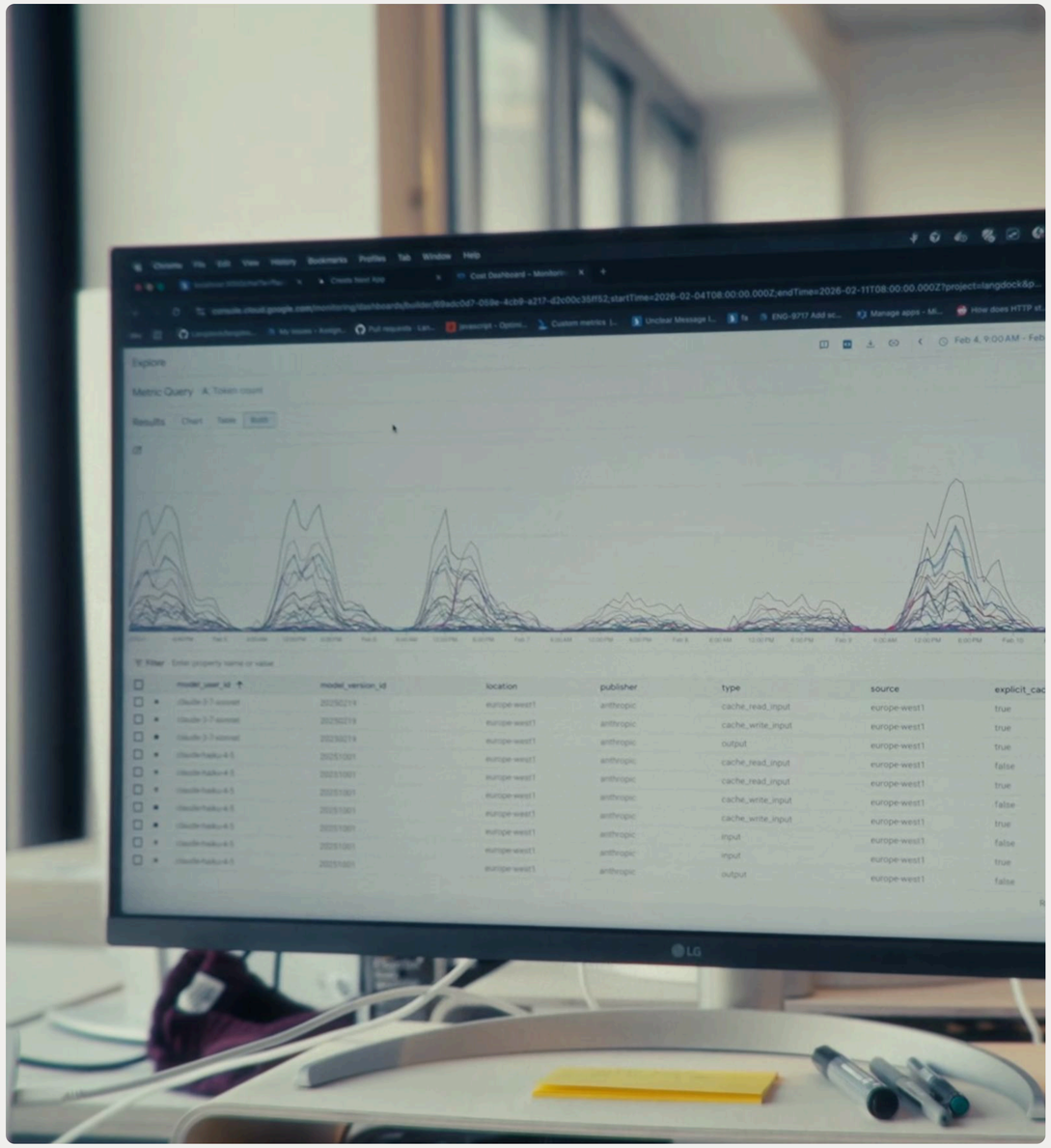
Cache Money

How Prompt Caching Cut Our LLM Bills in Half



Balázs Csomor
Engineering Lead @ Langdock

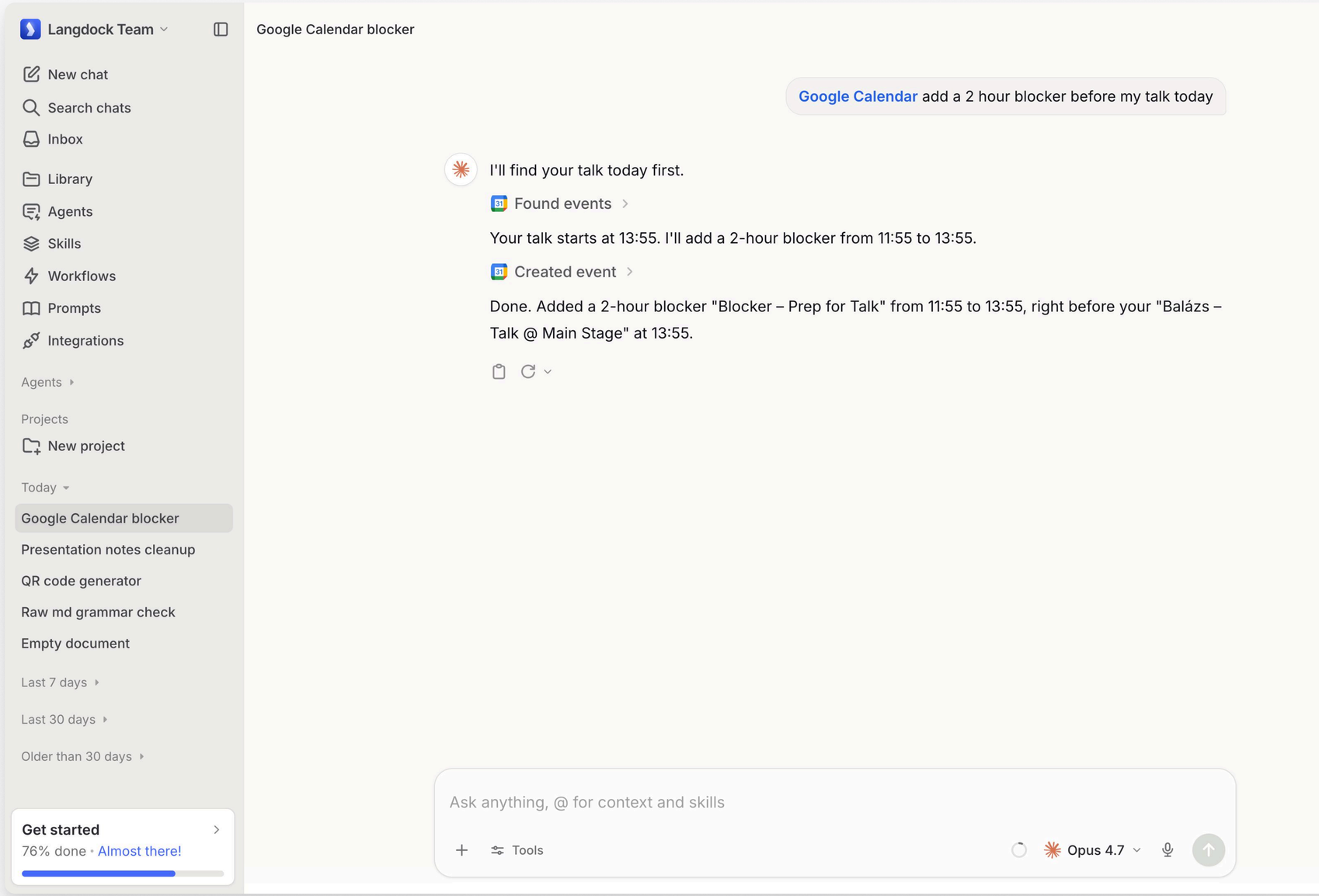
What is Langdock?

Product

- Enterprise-ready
- Security first
- Deployable anywhere
- Model agnostic

Company

- 150k+ users
- 7,000+ customers
- $30M ARR
- Team of 50

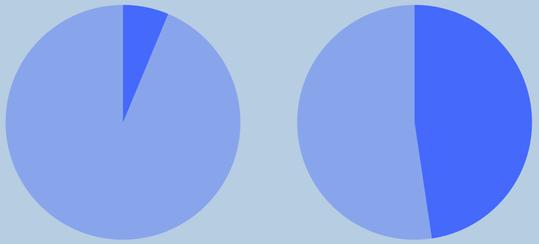


Rising LLM costs

Costs grew faster than the number of users, for two reasons

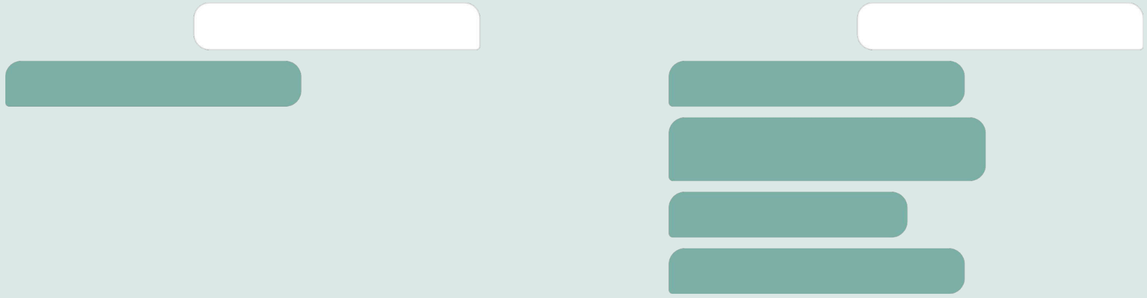
Model shift

More expensive models became more popular among Langdock users



Agentic workloads

More tool calls result in more API calls and more tokens used



What can you do about LLM costs?

....when your core business is providing access to LLMs?

~~Raise prices~~
~~Add stricter usage limits~~
~~Remove expensive models~~
~~Limit tool calling~~
~~Prompt compression~~
~~Smarter context window~~
Optimize prompt caching

Prompt caching
The basics

A provider-side feature that lets an LLM API reuse the prefix of a prompt across requests, making responses faster and cheaper.

Sending the same prompt twice is ~90% cheaper on the second request.

Request

```
{
  "model": "gpt-5.2",
  "input": [
    {
      "role": "user",
      "content": "Please translate this text to English: [...]"
    }
  ]
}
```

Response #1

```
{
  "usage": {
    "input_tokens": 1455,
    "input_tokens_details": {
      "cached_tokens": 0
    },
    "output_tokens": 102,
    "output_tokens_details": {
      "reasoning_tokens": 0
    },
    "total_tokens": 1557
  },
  ...
}
```

Response #2

```
{
  "usage": {
    "input_tokens": 1455,
    "input_tokens_details": {
      "cached_tokens": 1280
    },
    "output_tokens": 108,
    "output_tokens_details": {
      "reasoning_tokens": 0
    },
    "total_tokens": 1563
  },
  ...
}
```

Prompt caching
In conversational systems

The powerful part is the prefixable nature of conversational systems.

A follow-up message can use the cache from the the previous messages.

Input tokens	1536
Cached tokens	0
Output tokens	1236

Please translate this text to English: [...]

Sure, here is a translation of the text you provided: [...]

Input tokens	2774
Cached tokens	2560
Output tokens	1246

Please translate this text to English: [...]

Sure, here is a translation of the text you provided: [...]

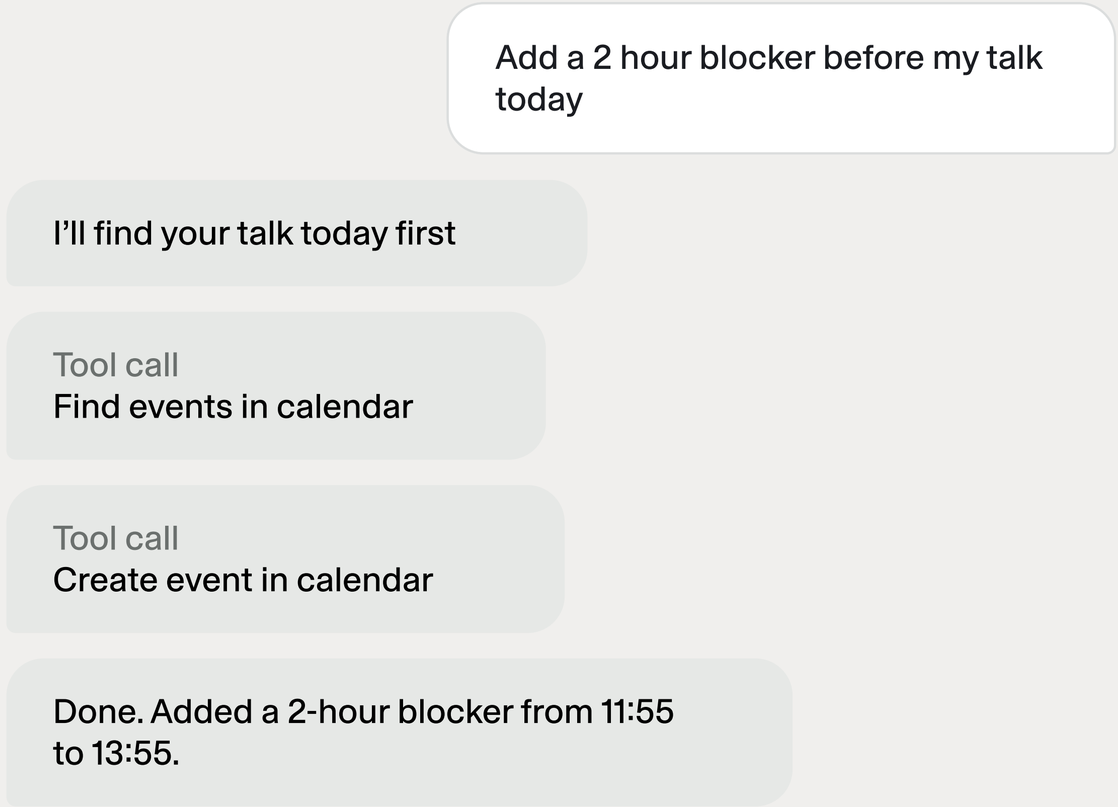
No, make it way less formal

Of course, here is a less formal translation of the text you provided: [...]

Prompt caching
In agentic systems

LLMs in agentic systems output a sequence of assistant messages and tool calls without user input.

In a well-designed system, steps inside turns are cheap.



Prompt caching
Breaking the cache

Any change in the prefix will result in a cache key miss and the entire request will be processed at full price.

Request #1

```
{
  "model": "gpt-5.2",
  "input": [
    {
      "role": "user",
      "content": "Please translate this text to English: [...]"
    }
  ]
}
```

Response #1

```
{
  "usage": {
    "input_tokens": 1455,
    "input_tokens_details": {
      "cached_tokens": 0
    },
    "output_tokens": 102,
    "output_tokens_details": {
      "reasoning_tokens": 0
    },
    "total_tokens": 1557
  },
  ...
}
```

Prompt caching
Breaking the cache

Any change in the prefix will result in a cache key miss and the entire request will be processed at full price.

Request #2

```
{
  "model": "gpt-5.2",
  "input": [
    {
      "role": "user",
      "content": "Can you please translate this text to English:"
    }
  ]
}
```

Response #1

```
{
  "usage": {
    "input_tokens": 1455,
    "input_tokens_details": {
      "cached_tokens": 0
    },
    "output_tokens": 102,
    "output_tokens_details": {
      "reasoning_tokens": 0
    },
    "total_tokens": 1557
  },
  ...
}
```

Response #2

```
{
  "usage": {
    "input_tokens": 1455,
    "input_tokens_details": {
      "cached_tokens": 0
    },
    "output_tokens": 102,
    "output_tokens_details": {
      "reasoning_tokens": 0
    },
    "total_tokens": 1557
  },
  ...
}
```

The *DO*s and *DON'T*s of prompt caching

By the time costs are a problem, you may have already made a few mistakes.

Turn on prompt caching

And override some basic configurations

OpenAI

Enabled by default

- Extend prompt cache retention to 24 hours as default is 5-10 minutes
  `prompt_cache_retention`
- Use cache key as default is first 256 tokens
  `prompt_cache_key`

Anthropic

Disabled by default

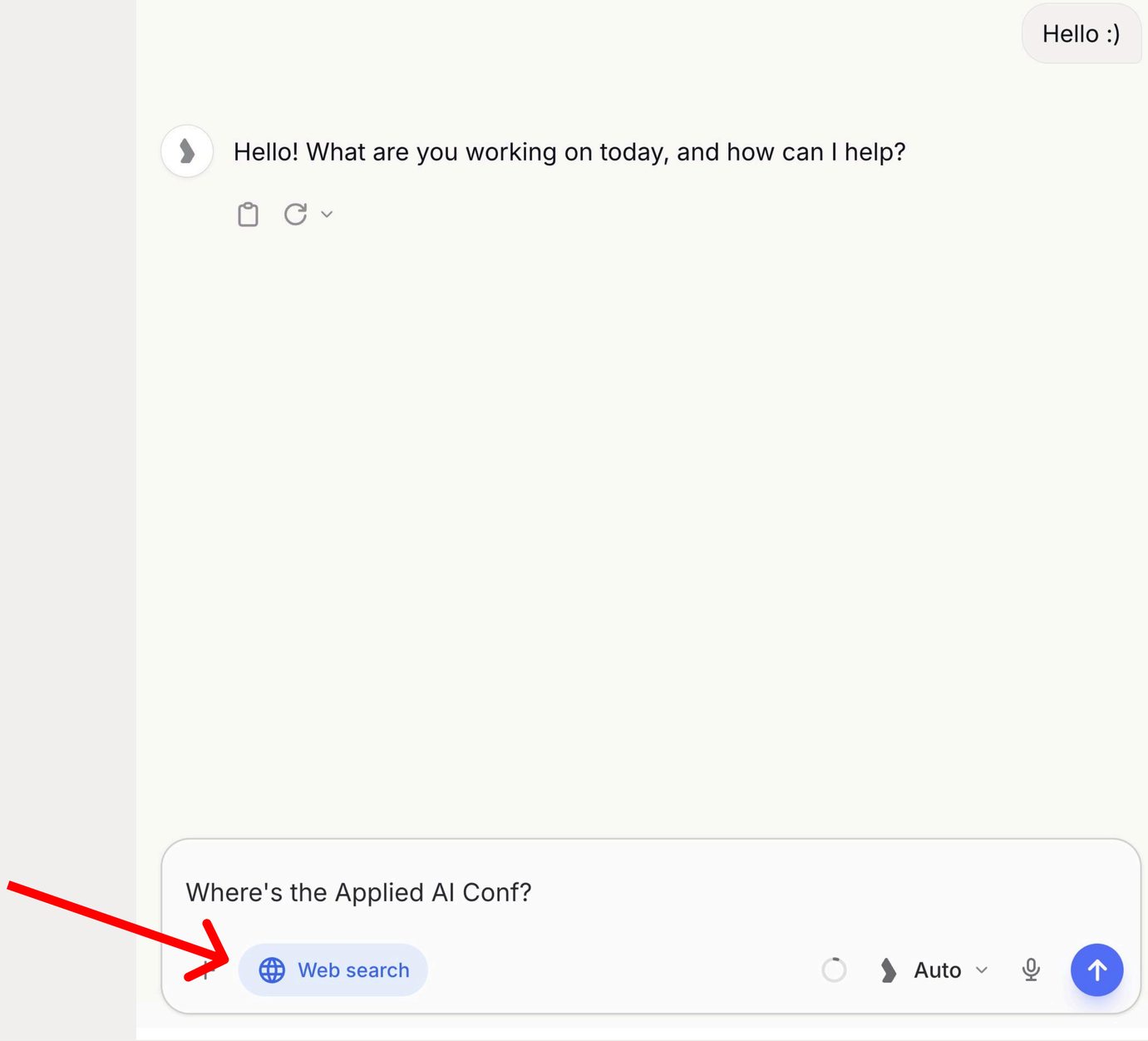
- Two modes
  - Automatic caching
  - Explicit cache breakpoints
- TTL
  - 5 minutes - 1.25x price
  - 1 hour - 2x price

Don’t let tool definitions change

The order of information sent to the LLM is

1. Tool definitions
2. System prompt
3. Messages

Any change in tool definitions busts your entire cache for that conversation.



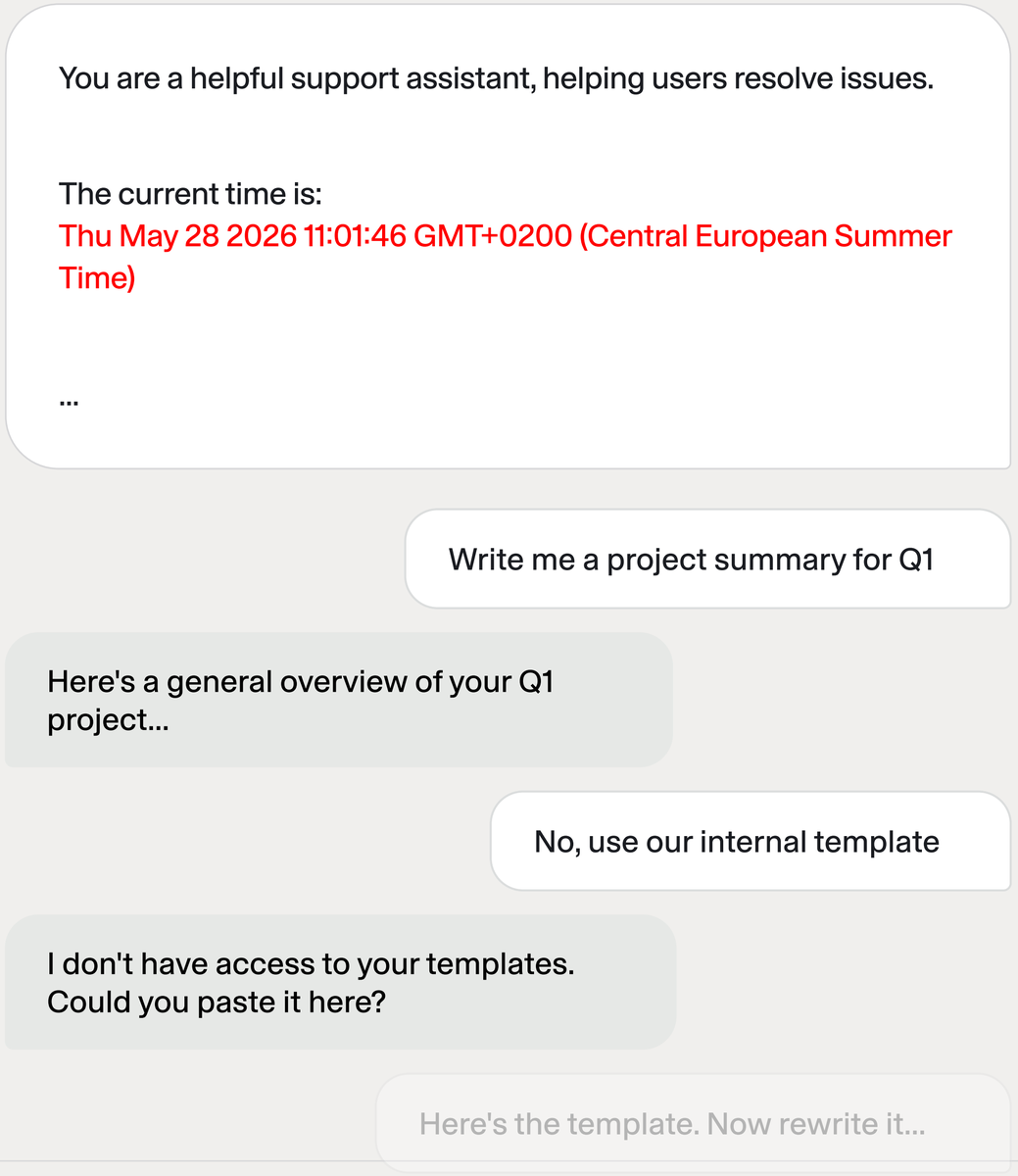
Don’t put dynamic content in the system prompt

Don’t add information that changes mid-conversation into the system prompt.

Use hidden assistant-user message pairs or tool calls instead instead.

Examples:

- Current timestamp
- Available documents



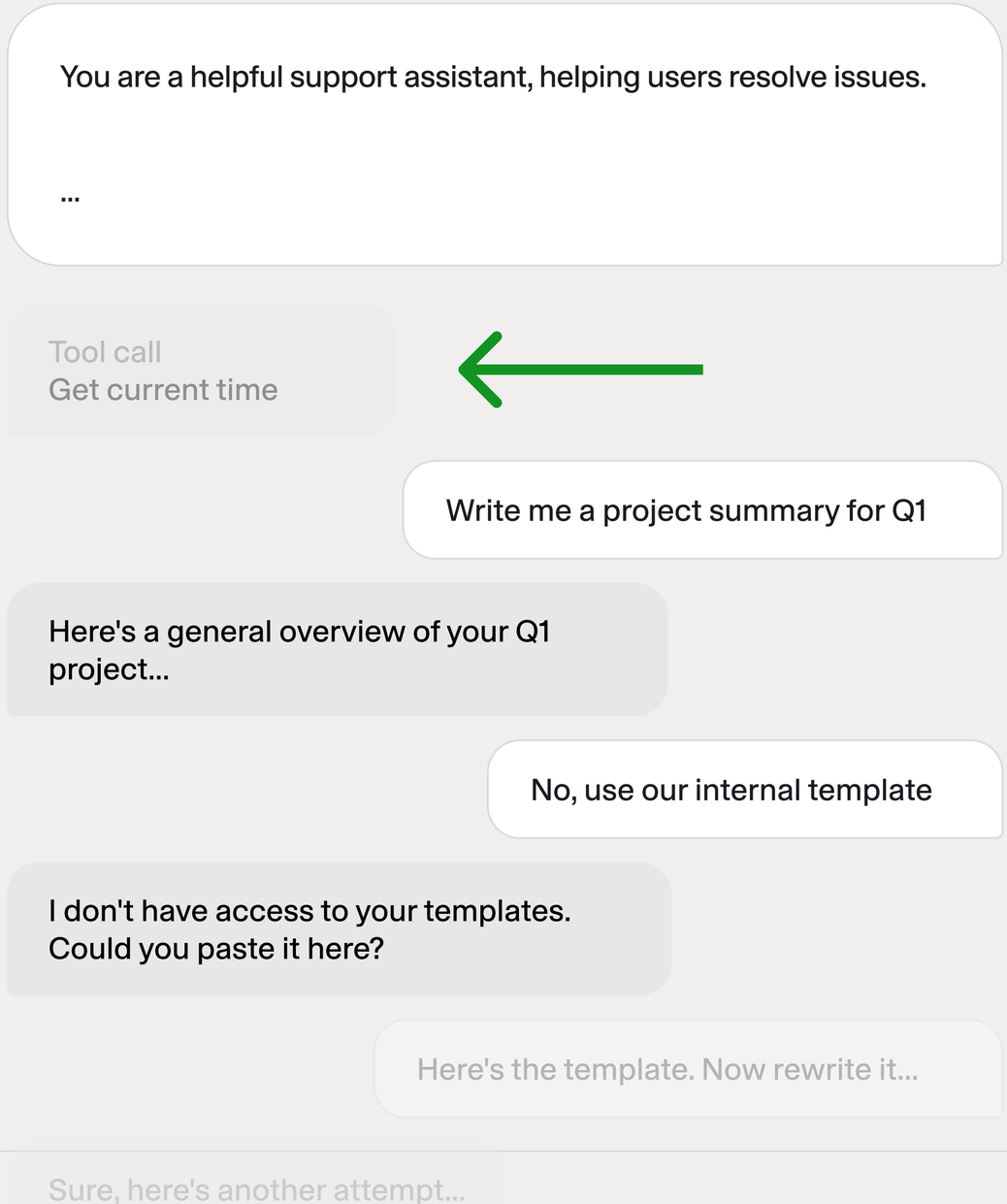
Don’t put dynamic content in the system prompt

Don’t add information that changes mid-conversation into the system prompt.

Use hidden assistant-user message pairs or tool calls instead instead.

Examples:

- Current timestamp
- Available documents



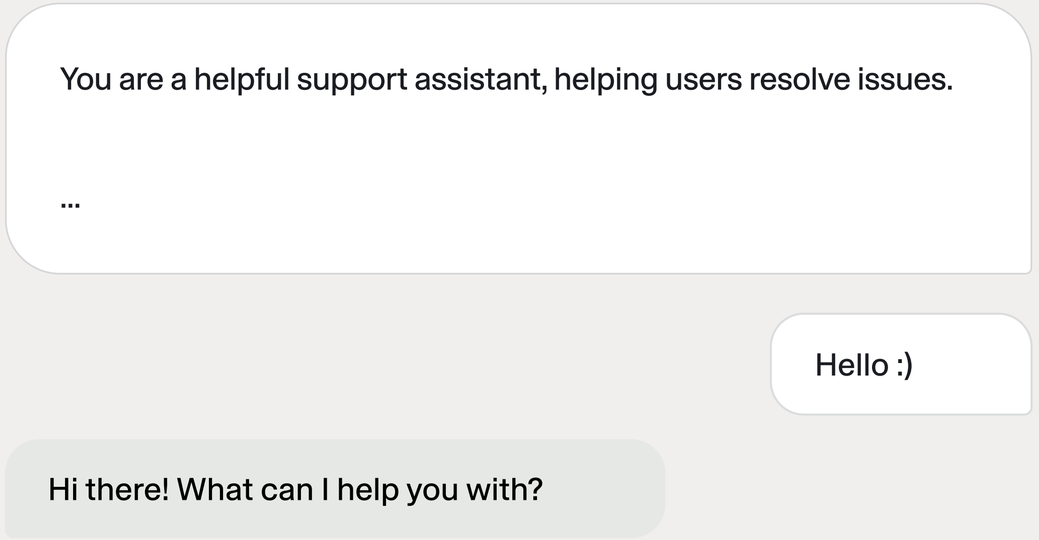
Don’t put dynamic content in the system prompt

Don’t add information that changes mid-conversation into the system prompt.

Use hidden assistant-user message pairs or tool calls instead instead.

Examples:

- Current timestamp
- Available documents



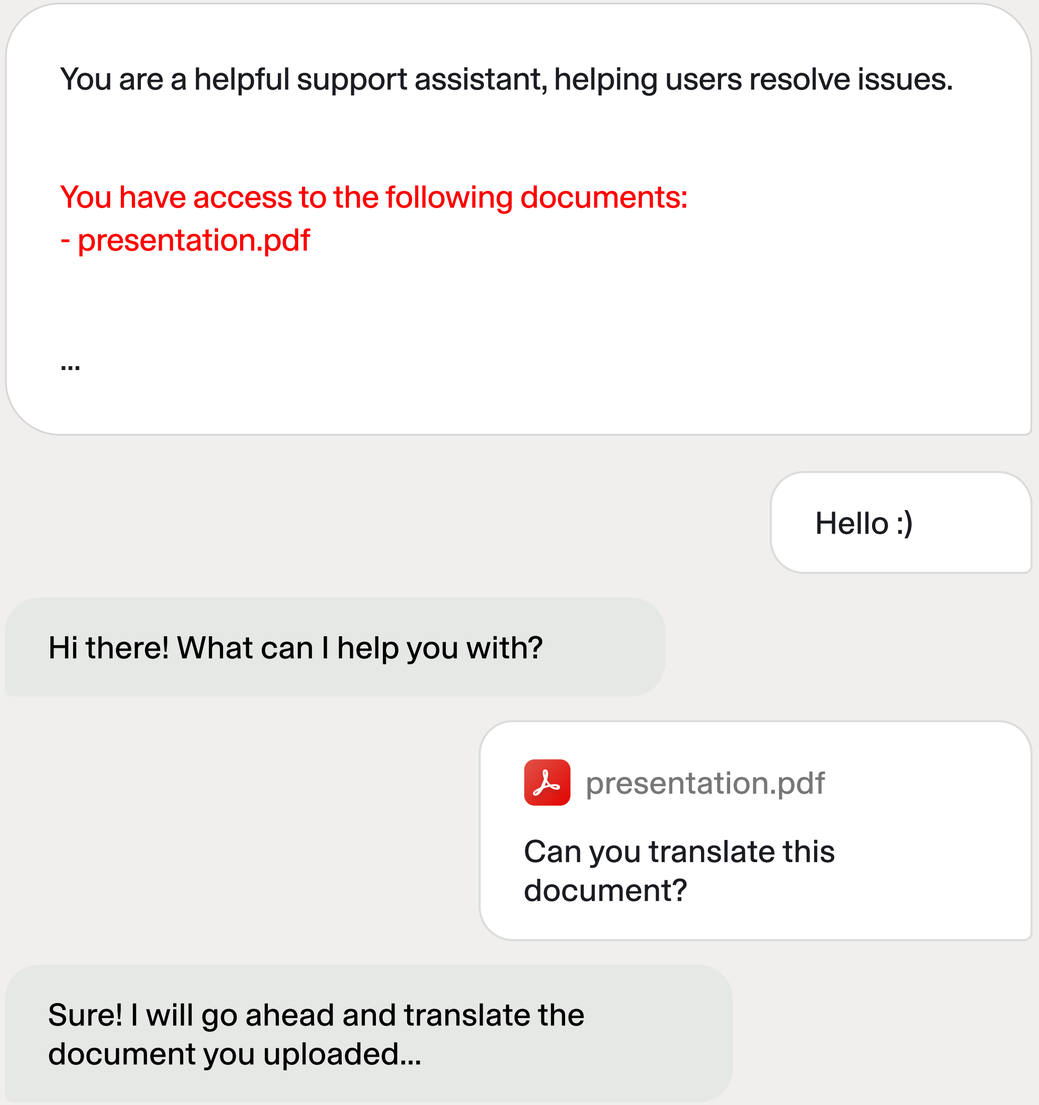
Don’t put dynamic content in the system prompt

Don’t add information that changes mid-conversation into the system prompt.

Use hidden assistant-user message pairs or tool calls instead instead.

Examples:

- Current timestamp
- Available documents



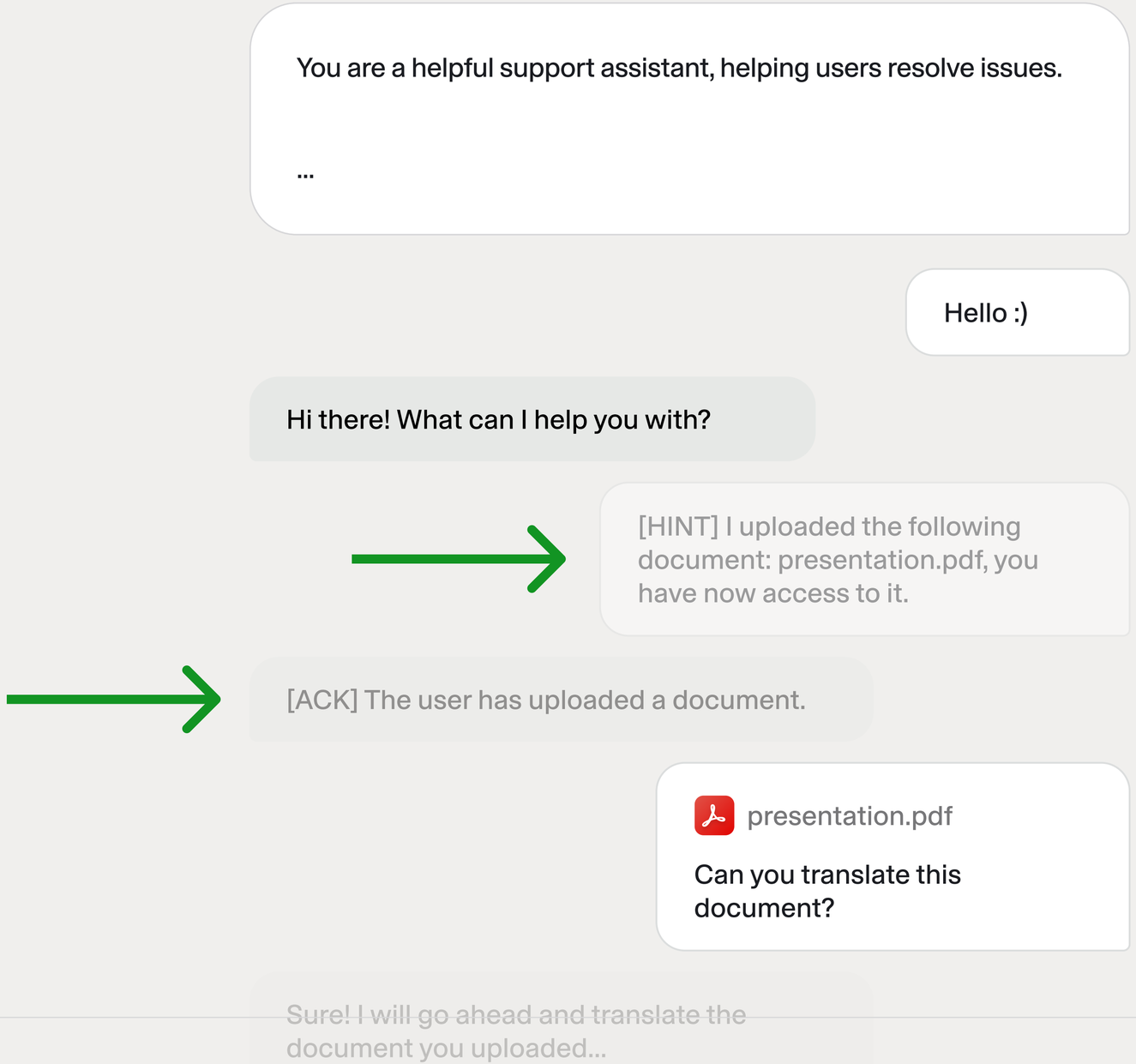
Don’t put dynamic content in the system prompt

Don’t add information that changes mid-conversation into the system prompt.

Use hidden assistant-user message pairs or tool calls instead instead.

Examples:

- Current timestamp
- Available documents



Make serializations deterministic

Semantically identical tool definitions, tool results, serialized JSON objects can break the cache.

Always sort JSON/YAML keys consistently (e.g. by alphabetical order).

```
{
  type: "function",
  name: "get_weather",
  description: "Get current temperature",
  parameters: {
    type: "object",
    properties: {
      location: {
        type: "string",
        description: "City and country",
      },
      unit: {
        type: "string",
        description: "Temperature unit",
        enum: ["celsius", "fahrenheit"],
      },
    },
    required: ["location"]
  }
}
```

```
{
  name: "get_weather",
  type: "function",
  description: "Get current temperature",
  parameters: {
    type: "object",
    properties: {
      location: {
        type: "string",
        description: "City and country",
      },
      unit: {
        type: "string",
        description: "Temperature unit",
        enum: ["celsius", "fahrenheit"],
      },
    },
    required: ["location"]
  }
}
```

Don’t use a rolling context window

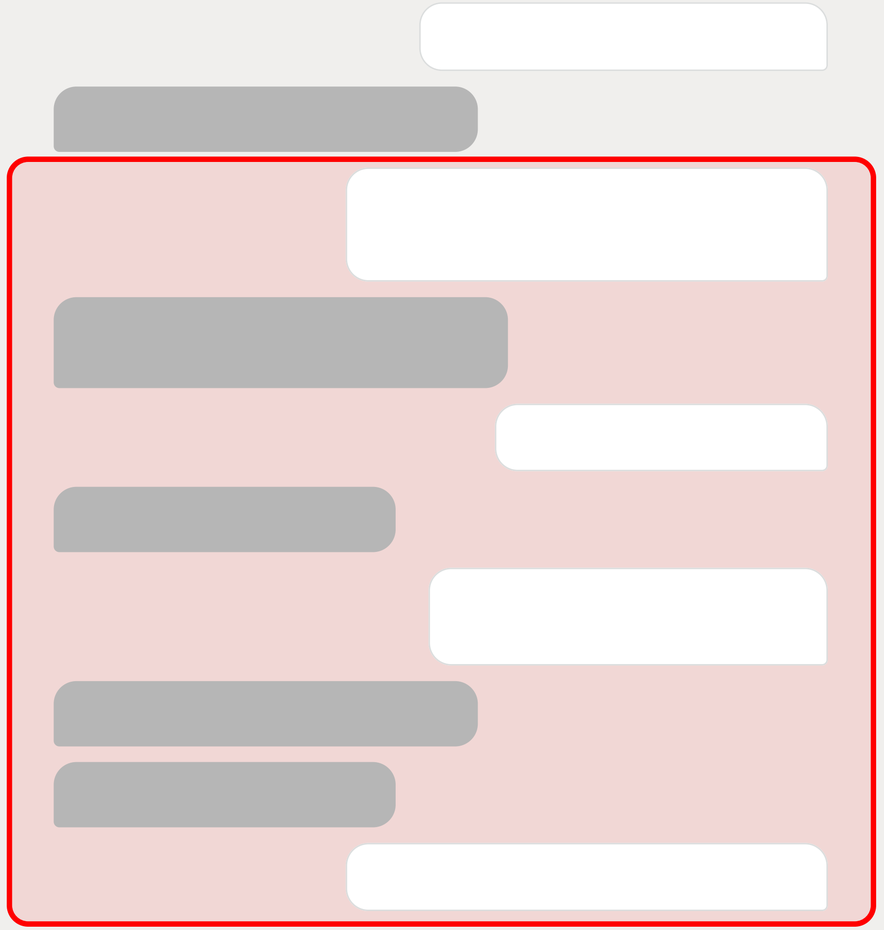
Dropping oldest messages that don’t fit into the context window will break the cache on every message.

Use summarization, compaction or another context window management strategy instead.

Don’t use a rolling context window

Dropping oldest messages that don’t fit into the context window will break the cache on every message.

Use summarization, compaction or another context window management strategy instead.



Don’t use a rolling context window

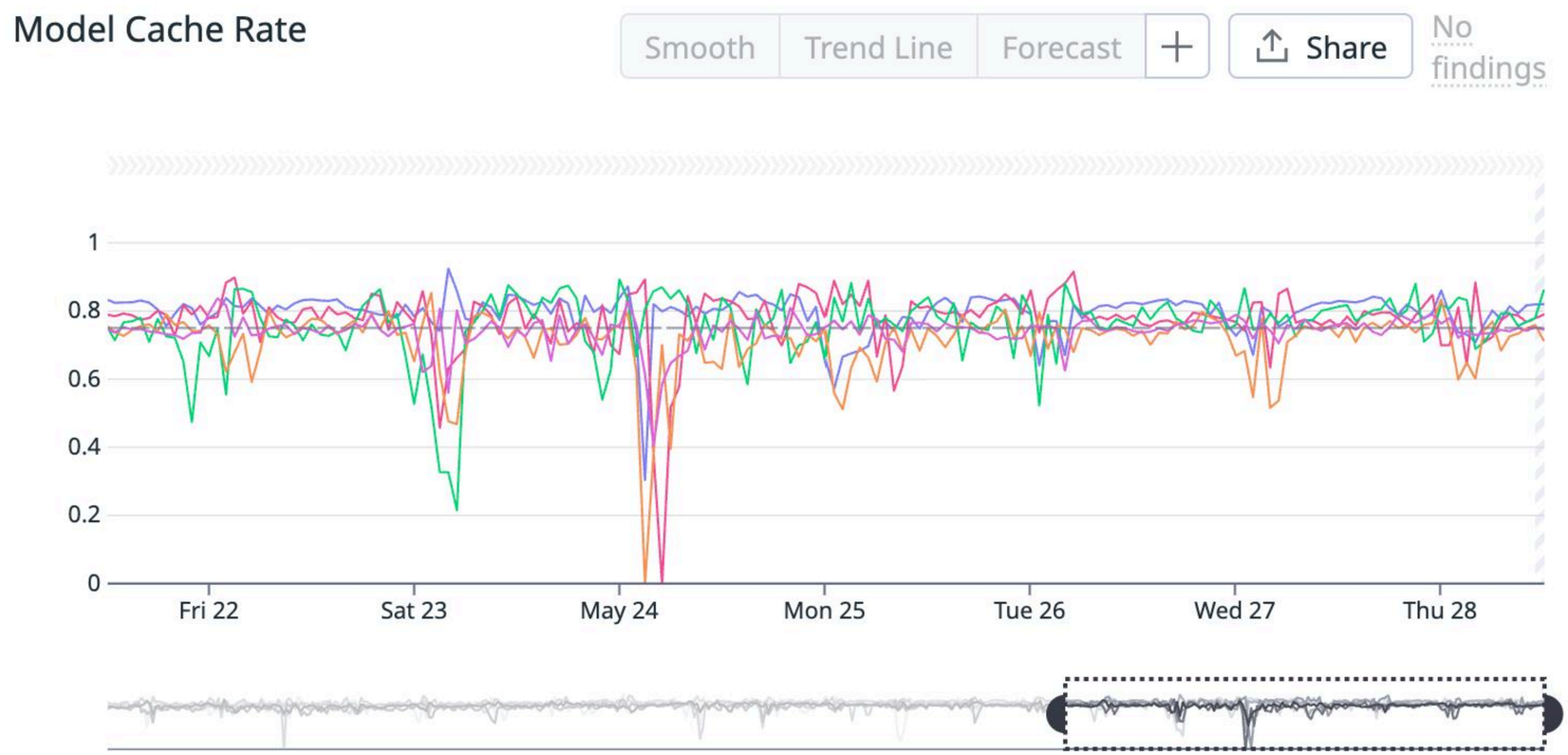
Dropping oldest messages that don’t fit into the context window will break the cache on every message.

Use summarization, compaction or another context window management strategy instead.

Build observability & developer tooling

Monitor cache hit rate (and costs) in production.

Build developer tooling and guardrails.

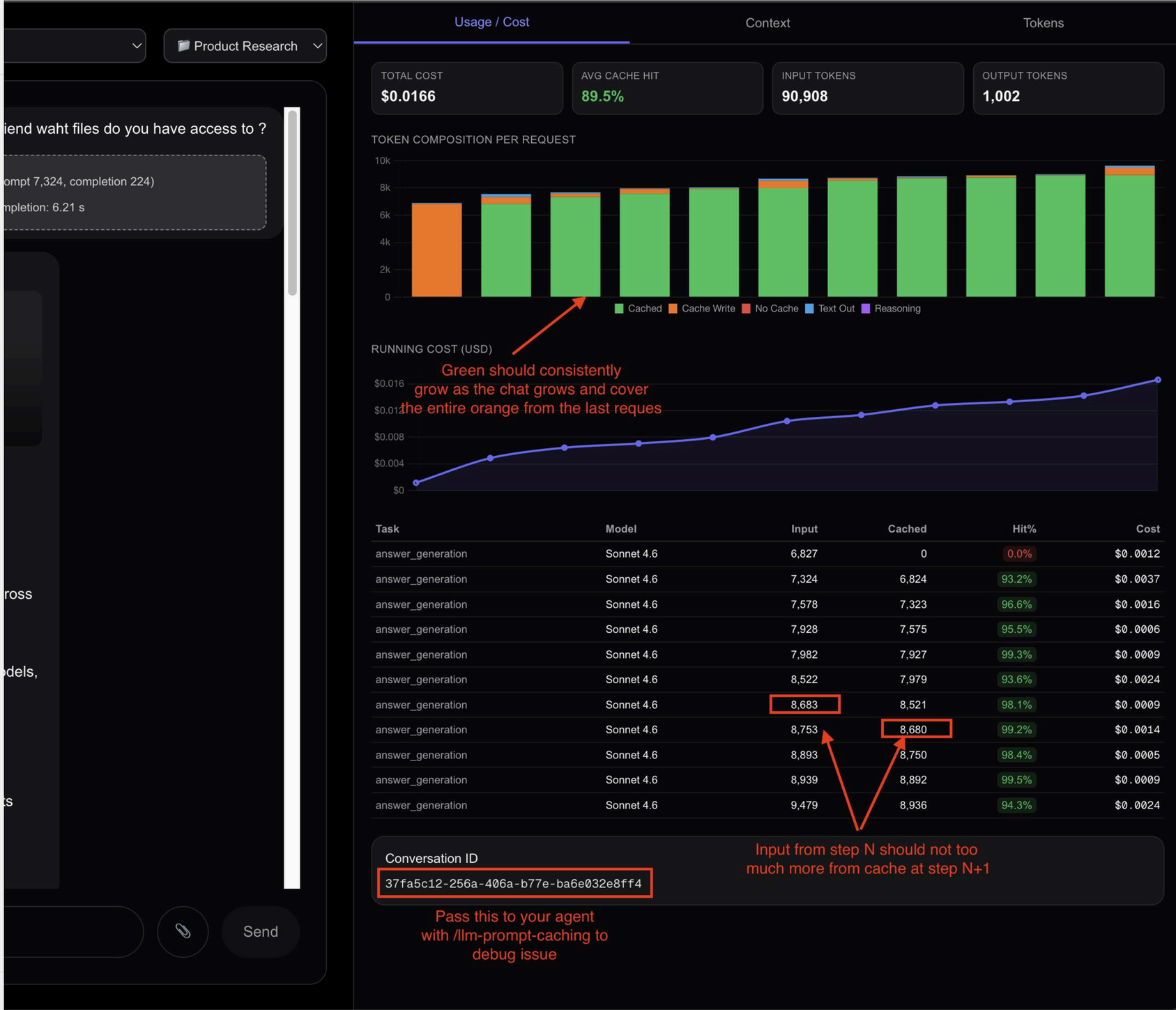


model in default_zero(sum:engine.llm_r...	FINDINGS	AVG	MIN	MAX	SUM	VALUE
gpt-5.2		0.80	0.30	0.93	135	0.82
claude-opus-4-6		0.78	0	0.92	131	0.79
gpt-5.4		0.76	0.22	0.89	128	0.86
claude-opus-4-7		0.71	0	0.85	121	0.71
claude-sonnet-4-6		0.74	0.41	0.84	125	0.75

Build observability & developer tooling

Monitor cache hit rate (and costs) in production.

Build developer tooling and guardrails.



Three open questions

How do we educate the user?

Stale conversations are more expensive.

Should we always summarize them? Should the user know?

What's our long-term caching observability stack?

Currently a mix of

- Provider dashboards
- Datadog
- Production database.

Where does model provider tooling evolve?

OpenAI’s 24-hour cache by default is new.

Anthropic’s automatic prompt caching at is new.

What should you do next?

Still experimenting with LLMs?

Keep this in mind, don’t optimize prematurely.

Already in production?

Go through the checklist.
Your margins (and your CV) will thank you.

Notes & prompt


We’re hiring!
langdock.com/careers
Make
something
people
want.