# Beyond Prompting: Context Engineering for Production-Grade AI

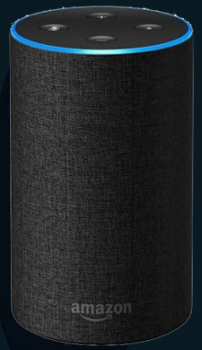
Ricardo Ferreira
Principal Developer Advocate @ Redis

@riferrei

© 2026 Redis Ltd. All rights reserved.

*“Alexa, can you explain what is context engineering”*

*“Hum, I don’t know that one.”*




Redis © 2026 Redis Ltd. All rights reserved.

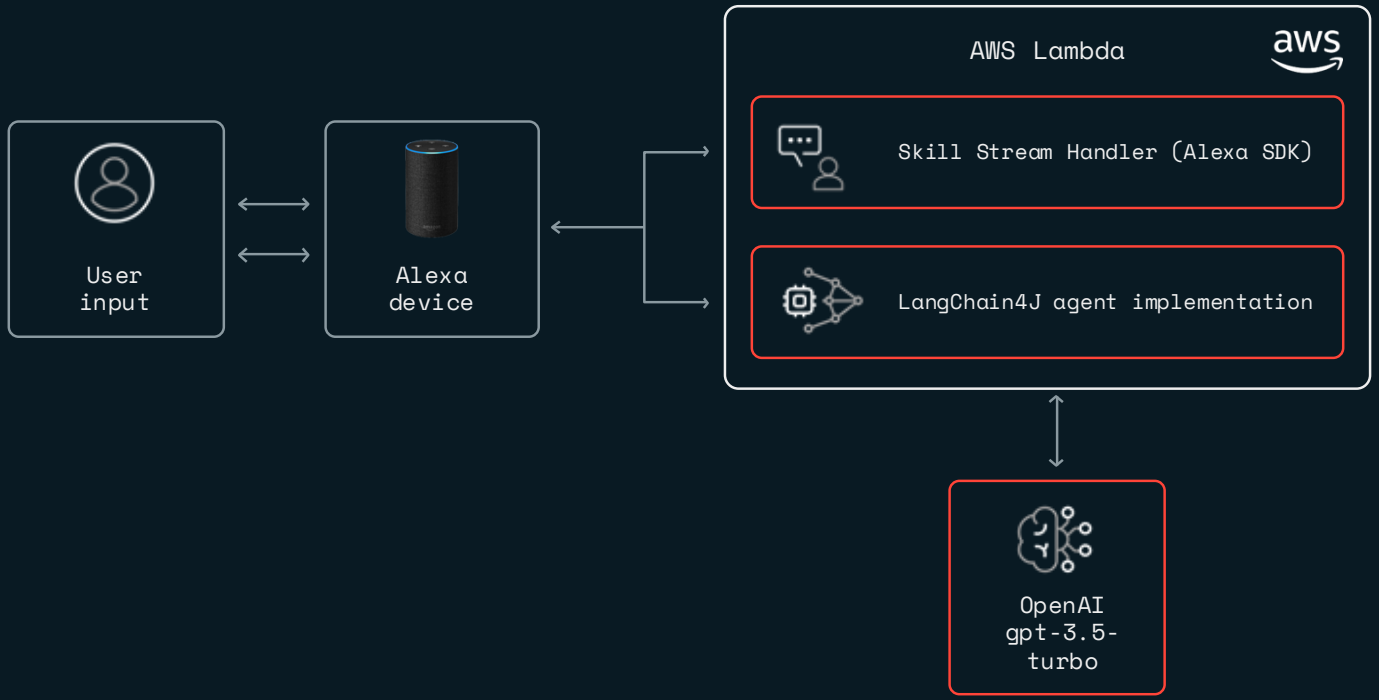
# My Jarvis initial implementation

Hoping the LLM will handle everything


Redis © 2026 Redis Ltd. All rights reserved.

# My naïve initial implementation

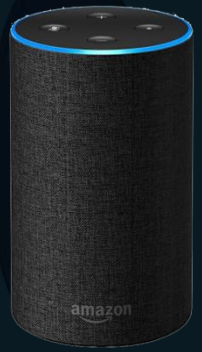
## Alexa skill backed by an AWS Lambda function using LangChain4J




Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to explain what is context engineering?*

*“Context engineering is the discipline of designing, shaping, and delivering the...”*



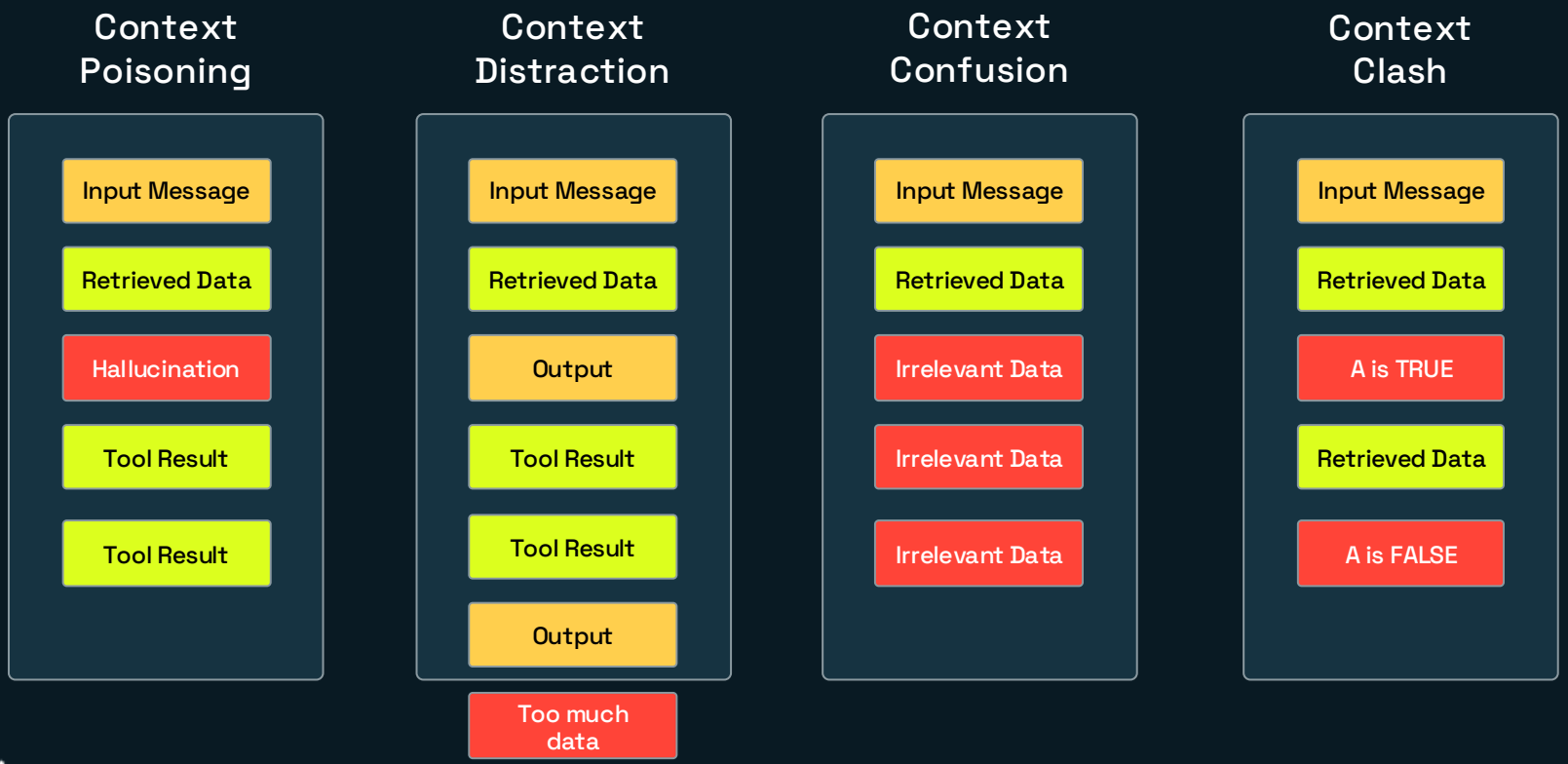

Redis © 2026 Redis Ltd. All rights reserved.


J.A.R.V.I.S, are you there?

Using the skill every day made me realize something was missing and the experience was odd.

Redis © 2026 Redis Ltd. All rights reserved.

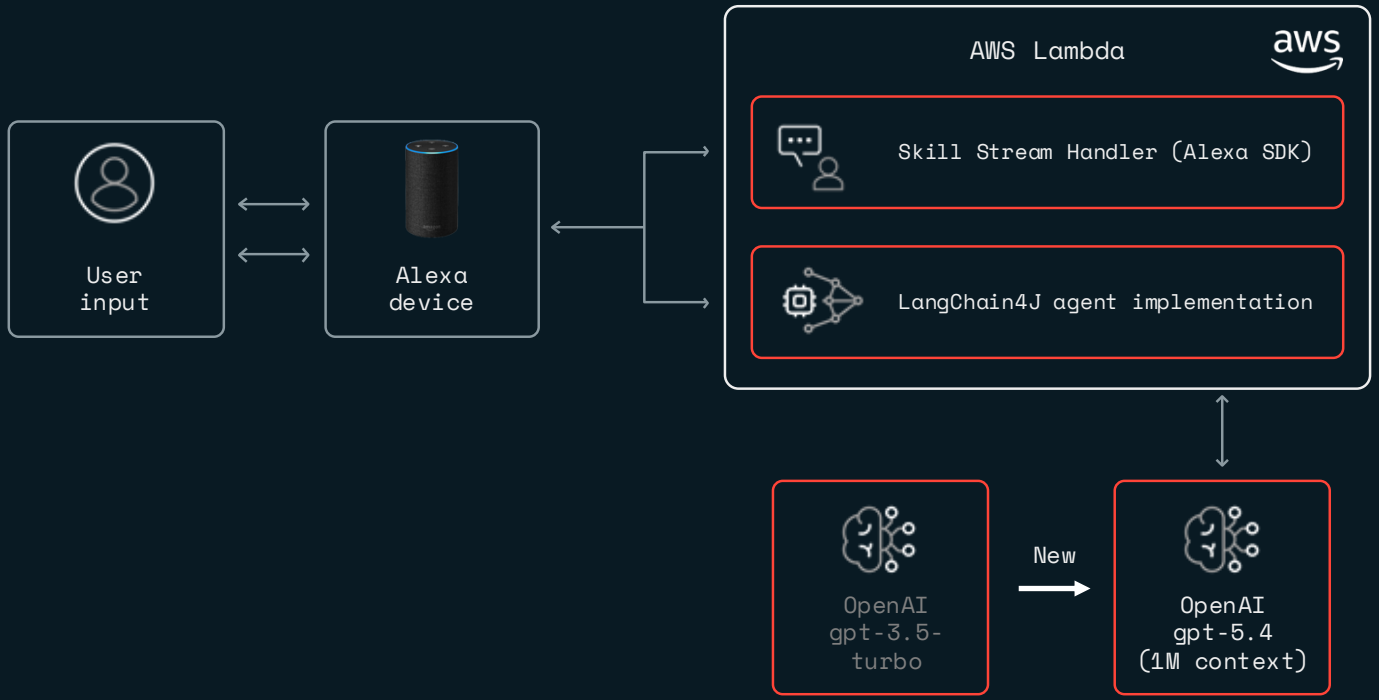
# Different types of context problems




Redis © 2026 Redis Ltd. All rights reserved.

# Using newer models as solution

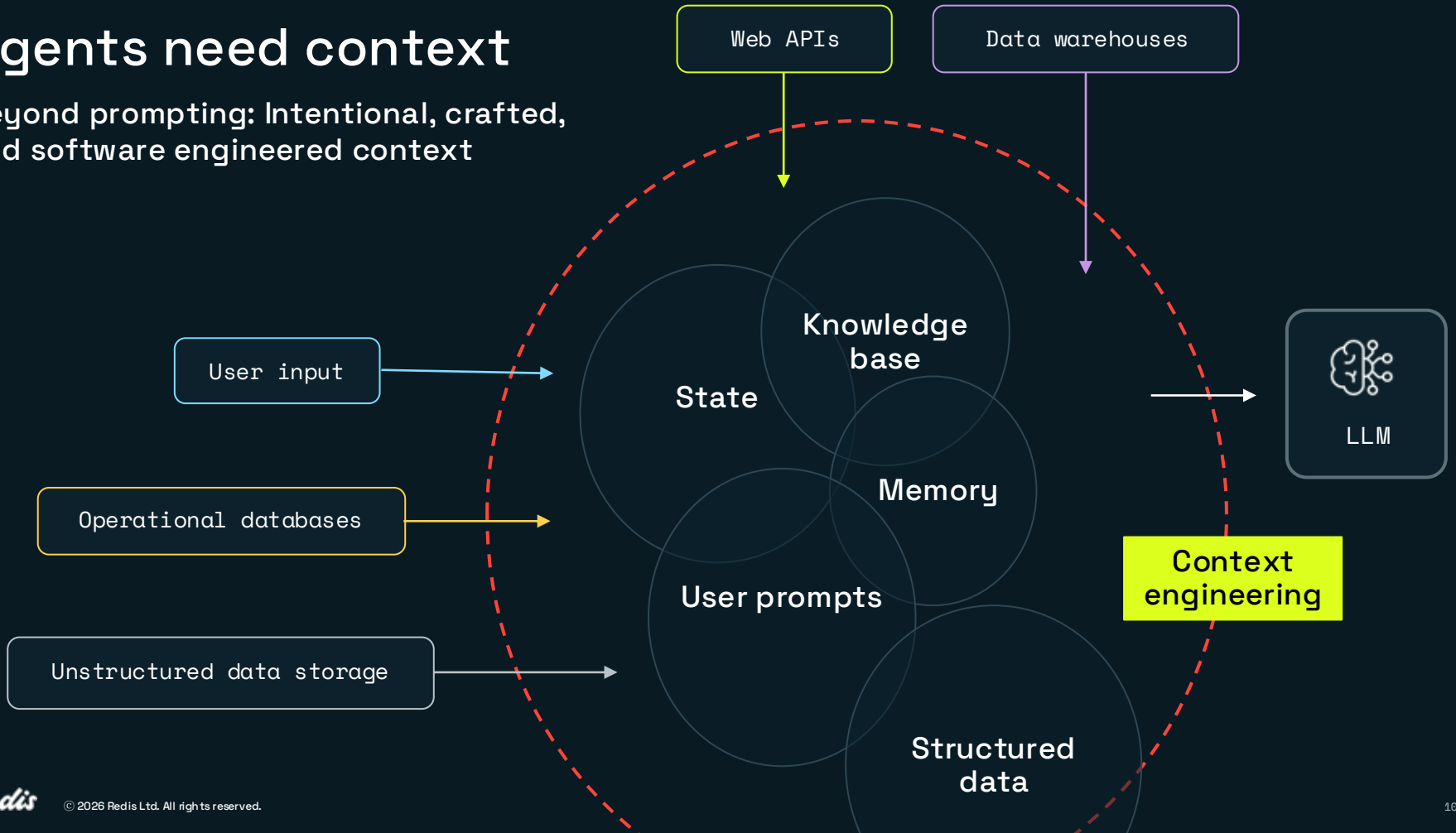
## Upgrading to newer models with larger context didn't improve




Redis © 2026 Redis Ltd. All rights reserved.

# Agents need context

Beyond prompting: Intentional, crafted, and software engineered context



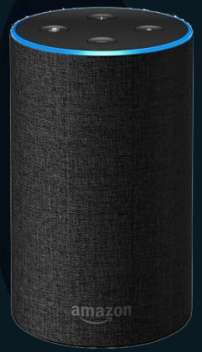

Redis © 2026 Redis Ltd. All rights reserved.

# Chapter 1: pragmatically solving each problem

© 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to remember about my dentist appointment tomorrow at 10:00am”*

*“Sure. I created a reminder for July 15, 2021, at 10:00am.”*




Redis © 2026 Redis Ltd. All rights reserved.

LLMs have no concept of what day it is. Every query needs temporal context.

© 2026 Redis Ltd. All rights reserved.

# Implementing essential tools

Solving the problem with date and time tracking

```
public class DateTimeTool {

    private static final ThreadLocal<String> userTimeZone =
            ThreadLocal.withInitial(() -> "America/New_York");

    @Tool("Set the user's timezone for this request")
    public String setUserTimeZone(String timeZone) {
        userTimeZone.set(timeZone);
        return "Timezone set to: " + timeZone;
    }

}
```

Redis © 2026 Redis Ltd. All rights reserved.

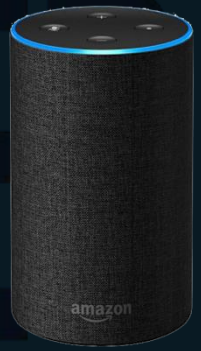
# Implementing essential tools

Solving the problem with date and time tracking

```
ChatAssistant chatAssistant =
            AiServices.builder(ChatAssistant.class)
                    .chatModel(chatModel)
                    .tools(List.of(dateTimeTool))
                    .build();

String response = chatAssistant.chat(systemPrompt, query);
```

Redis © 2026 Redis Ltd. All rights reserved.

Wakes up at 10:00am...




*“This is a reminder about your dentist appointment.”*

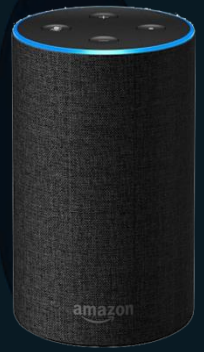
Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to confirm that my name is Ricardo.”*

*“Nice to meet you, Ricardo.”*

*“Alexa, ask my jarvis to recall what is my name?”*

*“Apologies, I don’t know you.”*




Redis © 2026 Redis Ltd. All rights reserved.

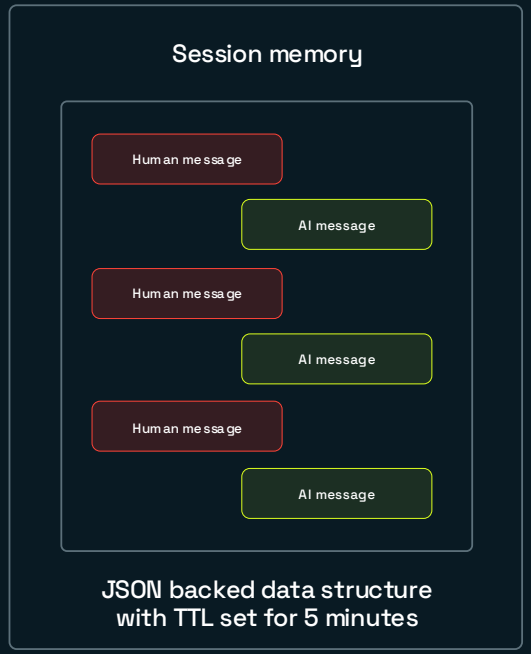
LLMs are stateless. Every request is a blank slate.

Redis © 2026 Redis Ltd. All rights reserved.

# Implementing short-term memory

## Implementing the memory backend persistence

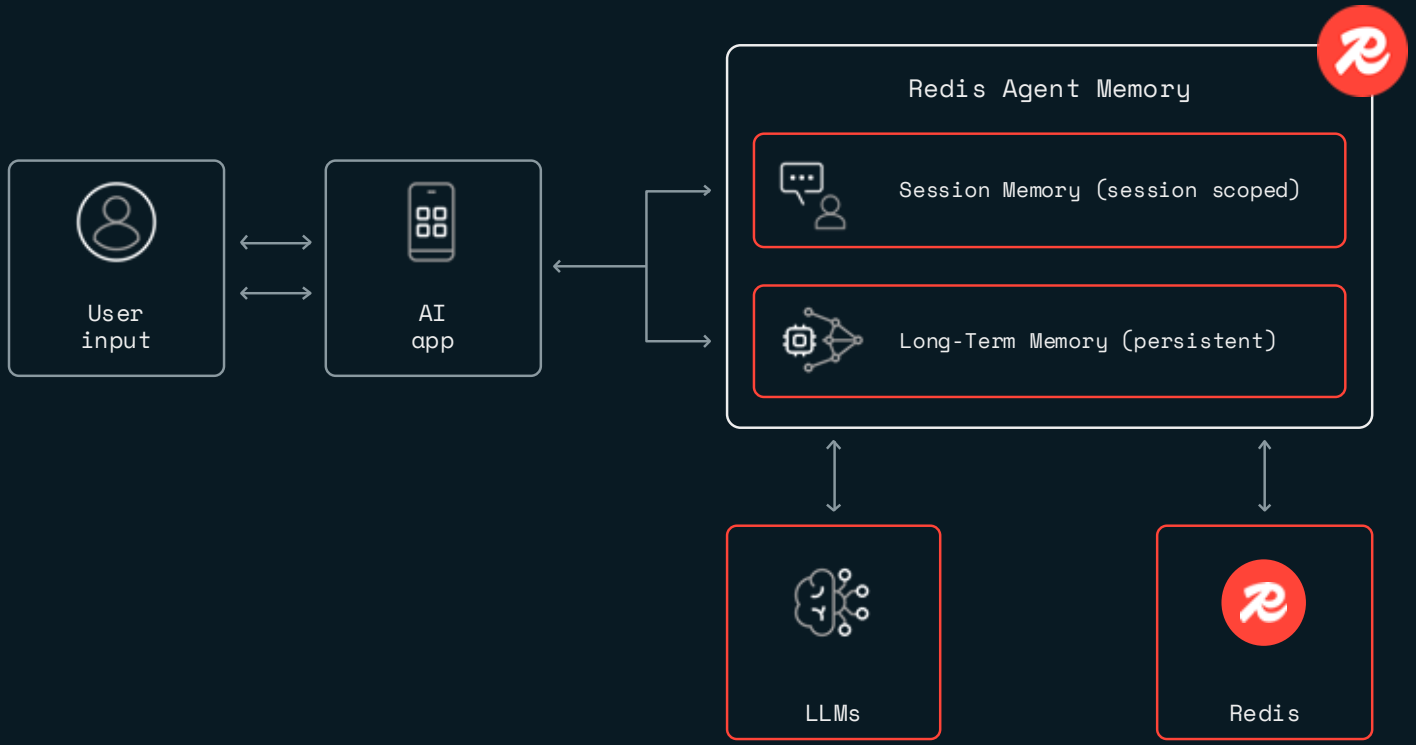
```
public class WorkingMemoryStore
    implements ChatMemoryStore {


    public List<Message> getMessages() {}


    public void updateMessages(
        Object memoryId,
        List<Message> messages
    ) {}
}
```




Redis © 2026 Redis Ltd. All rights reserved.

# Redis Agent Memory

## RESTful API and MCP server for managing dual-tiered agent memory




© 2026 Redis Ltd. All rights reserved.

# Implementing short-term memory

Using the sliding window chat memory implementation

```
private ChatMemory getChatMemory(String userId) {
    ChatMemoryStore chatMemoryStore = WorkingMemoryStore.builder()
            .agentMemoryServerUrl(REDIS_AGENT_MEMORY_SERVER_URL)
            .maxContextWindow(Integer.parseInt(OPENAI_CHAT_MAX_TOKENS))
            .build();

    return MessageWindowChatMemory.builder()
            .id(userId)
            .chatMemoryStore(chatMemoryStore)
            .maxMessages(10)
            .build();
}
```

Redis © 2026 Redis Ltd. All rights reserved.

# Implementing short-term memory

Using the chat memory system with the AI service

```
ChatAssistant chatAssistant =
            AiServices.builder(ChatAssistant.class)
                    .chatModel(chatModel)
                    .tools(tools)
                    .getChatMemory(getChatMemory(userId))
                    .build();

String response = chatAssistant.chat(systemPrompt, query);
```

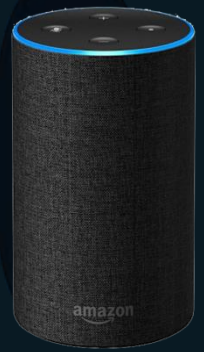
Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to confirm that my name is Ricardo.”*

*“Nice to meet you, Ricardo.”*

*“Alexa, ask my jarvis to recall what is my name?”*

*“Your name is Ricardo.”*




Redis © 2026 Redis Ltd. All rights reserved.

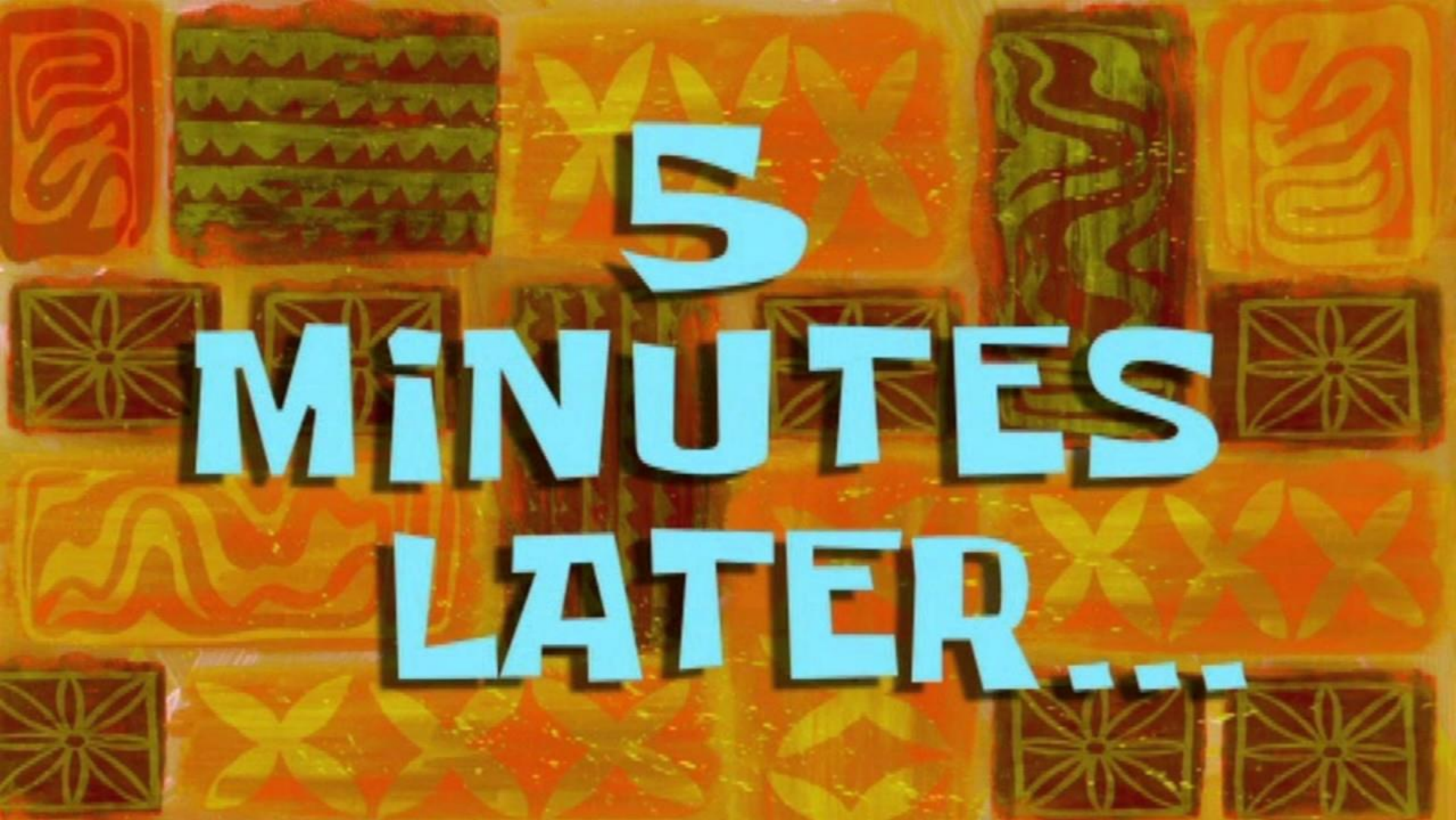
# Lessons learned

- Built-in implementation from LangChain4J had issues with tool calling. It call tools in a loop.
- Capping the number of messages with a fixed number sounded very arbitrary. Decided to keep all messages for better experiences.

```
return MessageWindowChatMemory.builder()
    .id(userId)
    .chatMemoryStore(chatMemoryStore)
    .maxMessages(10)
    .build();
```

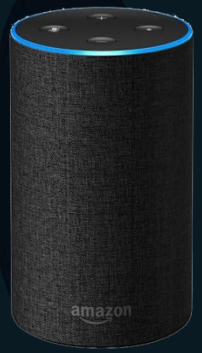
```
return WorkingMemoryChat.builder()
    .id(userId)
    .chatMemoryStore(chatMemoryStore)
    .build();
```

Redis © 2026 Redis Ltd. All rights reserved.


5
MINUTES
LATER...

*“Alexa, ask my jarvis to recall what is my name?”*

*“Apologies, I don’t know you.”*




Redis © 2026 Redis Ltd. All rights reserved.

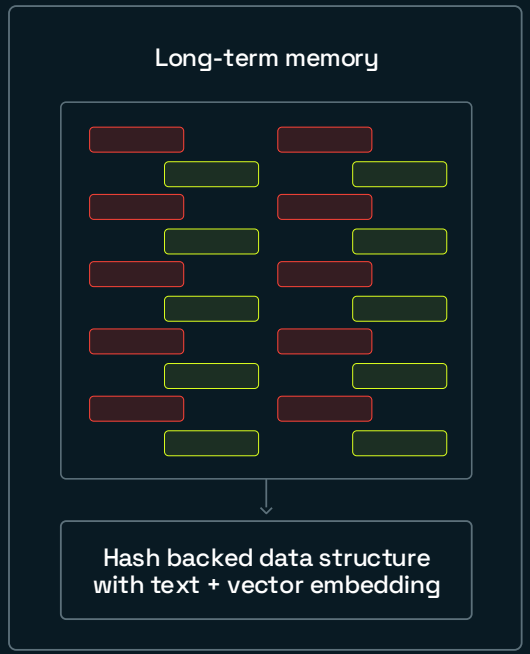
The context must use
long-term memories
from the user as well.

Redis © 2026 Redis Ltd. All rights reserved.

# Implementing long-term memory

## Building a content retriever for memories

```
public interface ContentRetriever {
    List<Content> retrieve(Query query);
}

private ContentRetriever getLongTermMemories(
   String userId) {
   return query -> memoryService.getMemories(
      userId, query.text())
         .stream()
         .map(Content::from)
         .toList();
}
```




Redis © 2026 Redis Ltd. All rights reserved.

# Implementing long-term memory

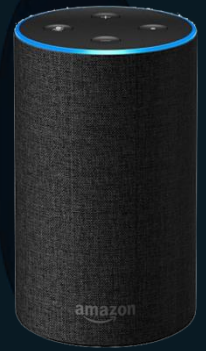
## Using the content retriever with the AI service

```
ChatAssistant chatAssistant =
          AiServices.builder(ChatAssistant.class)
                  .chatModel(chatModel)
                  .tools(tools)
                  .getChatMemory(getChatMemory(userId))
                  .contentRetriever(getLongTermMemories(userId))
                  .build();

String response = chatAssistant.chat(systemPrompt, query);
```

Redis © 2026 Redis Ltd. All rights reserved.

“Alexa, ask my jarvis to confirm my dentist appointment.”

“Very well. It is confirmed.”




“Alexa, ask my jarvis to verify what is for tomorrow?”

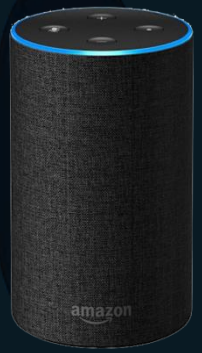
“You have dentist at 10 am.”

Redis © 2026 Redis Ltd. All rights reserved.


MANY, MANY
MINUTES LATER...

*“Alexa, ask my jarvis to verify what is for tomorrow?”*

*“You have dentist at 10 am.”*




Redis © 2026 Redis Ltd. All rights reserved.

# Lessons learned

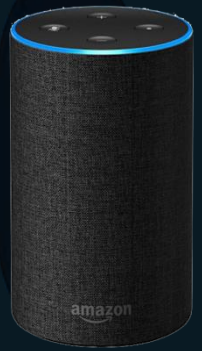
- The app's multi-tenancy required vector search with post-filtering. The concept of owner id.
- Deciding how many rows to return for the top-k vector search required many hours of testing.

```
public List<String> searchUserMemories(
   String userId, String memory) {
   var searchRequest = Map.of(
      "text", memory,
      "limit", Integer.parseInt(
         USER_MEMORIES_SEARCH_LIMIT),
      "filter", Map.of(
         "ownerId", Map.of("eq",
         sanitizeOwnerId(userId)),
         "namespace", Map.of("eq",
         LONG_TERM_MEMORY_NAMESPACE))
    );

    return extractTexts(executeSearch(
       searchRequest));
}
```

Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to confirm what is the code for the wireless touchpad in the garage?”*

*“I’m sorry, but I don’t have access to this information.”*




Redis © 2026 Redis Ltd. All rights reserved.

The context may require user memories or data from knowledge bases.

© 2026 Redis Ltd. All rights reserved.

# Dual layer long-term memory system

## Replacing the content retriever with a retrieval augmentator

```
private RetrievalAugmentor createRetrievalAugmentor(String userId) {
    Map<ContentRetriever, String> retrievers = Map.of(
            getLongTermMemories(userId), "User specific memories",
            getGeneralKnowledgeBase(), "General knowledge base"
    );

    LanguageModelQueryRouter router = LanguageModelQueryRouter.builder()
            .chatModel(chatModel)
            .retrieverToDescription(retrievers)
            .fallbackStrategy(FallbackStrategy.ROUTE_TO_ALL)
            .build();

    return DefaultRetrievalAugmentor.builder().queryRouter(router).build();
}
```

Redis © 2026 Redis Ltd. All rights reserved.

# Dual layer long-term memory system

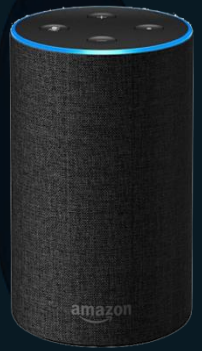
Replacing the content retriever with a retrieval augmentator

```
ChatAssistant chatAssistant =
        AiServices.builder(ChatAssistant.class)
            .chatModel(chatModel)
            .chatMemory(getChatMemory(userId))
            //.contentRetriever(getLongTermMemories(userId))
            .retrievalAugmentor(createRetrievalAugmentor(userId))
            .tools(tools)
            .build();

String response = chatAssistant.chat(systemPrompt, query);
```

Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to confirm what is the code for the wireless touchpad in the garage?*

*“You can unlock your garage door with the code 7121. Is there anything else?”*



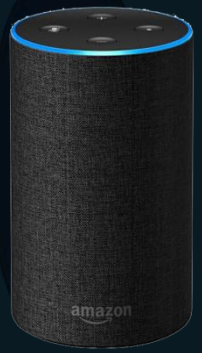

Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to recall who was the John Doe I have meet.*

*“John Doe was…”*

*“Alexa, ask my jarvis to recall where he lives?*

*“Where lives who?”*




Redis © 2026 Redis Ltd. All rights reserved.

# Improving query retrieval quality

Using a query transformer to implement compressing

```
QueryTransformer queryTransformer =
        new CompressingQueryTransformer(chatModel);


return DefaultRetrievalAugmentor.builder()
    .queryTransformer(queryTransformer) // Added
    .queryRouter(router)
    .build();
```

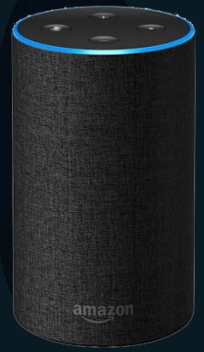
Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to recall who was the John Doe I have meet.”*

*“John Doe was…”*

*“Alexa, ask my jarvis to recall where [he → John Doe] lives?”*

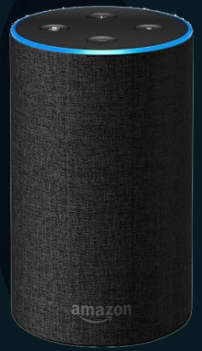
*“He lives in Boston, MA.”*




Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to confirm what is my favorite color?*

*“Your favorite color is black. Also, you love writing code using Java.”*




Redis © 2026 Redis Ltd. All rights reserved.

The model is dumping everything retrieved. It needs to be selective.

© 2026 Redis Ltd. All rights reserved.

# Prompt design for few-shots

Creating a template that prompts can leverage

```
ContentInjector contentInjector = DefaultContentInjector.builder()
    .promptTemplate(PromptTemplate.from(
        "{{userMessage}}\n\n[Context]\n{{contents}}"
    )).build();

return DefaultRetrievalAugmentor.builder()
    .contentInjector(contentInjector) // Added
    .queryTransformer(queryTransformer)
    .queryRouter(router)
    .build();
```

Redis © 2026 Redis Ltd. All rights reserved.

# Prompt design for few-shots

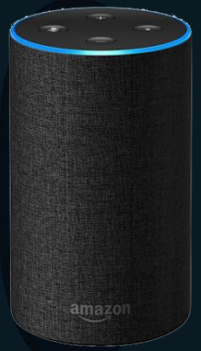
Using the template with structured prompts

[Example 1 - Ignoring irrelevant context]
User: ”What programming language do I use?”
Context: ”Favorite color is black”, ”Birthday is October 5th”, ”Enjoys Java”
Response: ”You enjoy coding in Java.”

Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to confirm what is my favorite color?*

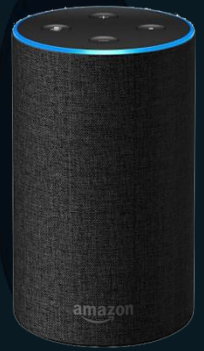
*“Your favorite color is black.”*




Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to recall which PR is a priority right now?”*

*“Adding support for JSON and going to dentist, are the priority for this week.”*




Redis © 2026 Redis Ltd. All rights reserved.

The retrieved context data was semantically correct but irrelevant to the query.

Redis © 2026 Redis Ltd. All rights reserved.

# Improving context quality with reranking

## Creating a new scoring model based on Cohere API

```
ScoringModel scoringModel = CohereScoringModel.builder()
        .apiKey(COHERE_API_KEY)
        .modelName(COHERE_MODEL_NAME)
        .build();
```

Redis © 2026 Redis Ltd. All rights reserved.

# Improving context quality with reranking

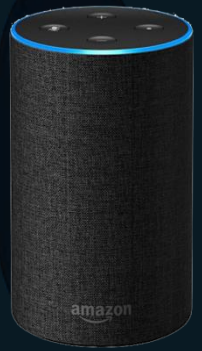
Using the content aggregator with a scoring model

```
ContentAggregator contentAggregator =
   ReRankingContentAggregator.builder()
      .scoringModel(scoringModel)
      .minScore(0.8)
      .build();

return DefaultRetrievalAugmentor.builder()
   .contentInjector(contentInjector)
   .queryTransformer(queryTransformer)
   .queryRouter(router)
   .contentAggregator(contentAggregator) // Added
   .build();
```

Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to recall which PR is a priority right now?”*

*“Adding support for JSON is the priority for this week.”*




Redis © 2026 Redis Ltd. All rights reserved.

# Lessons learned

- Using a local model like ms-marco-MiniLM-L-6 (ONNX format) is better, but deployment archive became too large for a AWS Lambda function.
- Changing models made me re-evaluate the min score model used. The precision changed with different values.

**JAR file maximum size (direct):** 50MB
**JAR file maximum size (with S3):** 250MB

**JAR file with local model:** 415MB

```
ContentAggregator contentAggregator =
    ReRankingContentAggregator.builder()
        .scoringModel(scoringModel)
        .minScore(0.8)
        .build();
```

Redis © 2026 Redis Ltd. All rights reserved.

# Chapter 2: Dealing with the monthly cost

© 2026 Redis Ltd. All rights reserved.

# How much a single Alexa interaction cost?

A single day costs $0.09 per user and $8.65 per month

## Cost with LLM

One Alexa query triggers 4–5 LLM calls to the OpenAI API. It includes query routing, query compression, and tool calling (at least 3 tools)

→ Single user
- 10 queries per day
- $4.20 per month

→ Three users
- 10 queries each per day
- $12.60 per month

## Cost with Cohere

**During the reranking process, each call from Alexa triggers 1 Cohere call for the multilingual-v3.0 model which costs $2 per 1,000 searches**

→ Single user
- 10 queries per day
- 10 rerank calls = $0.02

→ Three users
- 10 queries each per day
- 30 rerank calls = $0.06

## Context growth cost

**Context persist across all user sessions. Every call re-sends the full session history, making token usage grow linearly and not flat**

→ Single user
- 10 queries per day
- 22,700 tokens = $0.0708

→ Three users
- 10 queries each per day
- 67,800 tokens = $0.2123

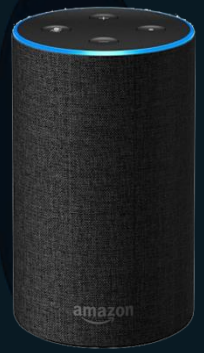
Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to recall…”*

[Adds question data to context]

*“Some elaborated answer…”*

[Adds answer data to context]




Redis © 2026 Redis Ltd. All rights reserved.

# Compressing the context for efficiency

Making the short-term memory efficient and costly effective

```
private ChatMemory getChatMemory(String userId) {
    ChatMemoryStore chatMemoryStore = WorkingMemoryStore.builder()
            .agentMemoryServerUrl(REDIS_AGENT_MEMORY_SERVER_URL)
            .maxContextWindow(OPENAI_CHAT_MAX_TOKENS)
            .build();

    return TokenWindowChatMemory.builder() //New implementation
            .id(userId)
            .maxTokens(OPENAI_CHAT_MAX_TOKENS,
                new OpenAiTokenCountEstimator(OPENAI_MODEL_NAME))
            .chatMemoryStore(chatMemoryStore)
            .build();
}
```

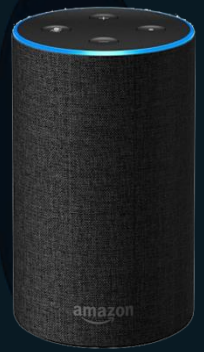
Redis © 2026 Redis Ltd. All rights reserved.

*“Alexa, ask my jarvis to recall who is coming for dinner tonight?”*

*“Both John and Susan are…”*

*“Alexa, ask my jarvis to recall am I having dinner with who tonight?”*

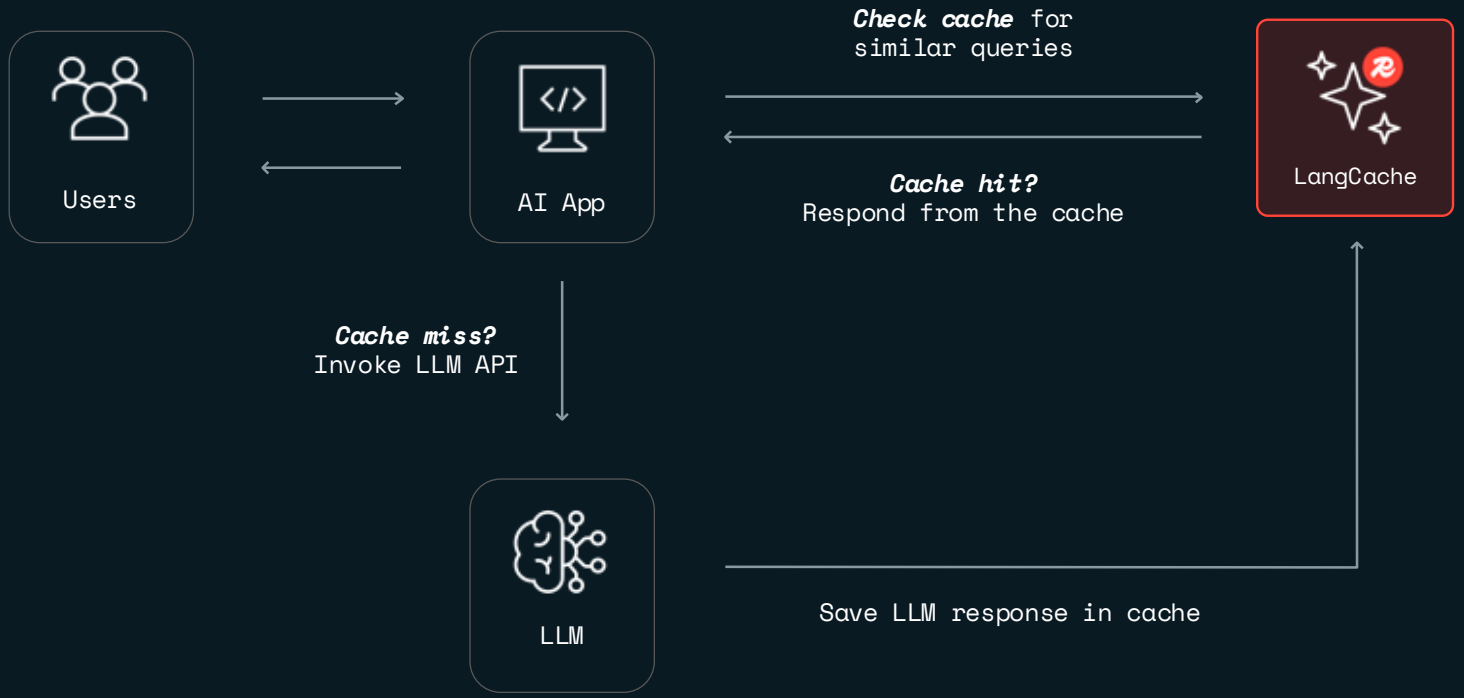
*“Both John and Susan are…”*




Redis © 2026 Redis Ltd. All rights reserved.

# Cutting LLM cost calls with semantic caching

## Reusing previously answered questions with Redis LangCache




Redis © 2026 Redis Ltd. All rights reserved.

# Cutting LLM cost calls with semantic caching

Reusing previously answered questions with Redis LangCache

```
String response = langCacheService.searchForResponse(query)
    .orElseGet(() -> {
        String response = chatAssistant.chat(systemPrompt, query);
        langCacheService.addNewResponse(query, response);
        return response;
    });
```

Redis © 2026 Redis Ltd. All rights reserved.

Context engineering isn't a feature, it's architecture. Get this right and everything else gets easier.

Redis © 2026 Redis Ltd. All rights reserved.

# Let’s stay connected

Get My Jarvis on GitHub

Let’s connect on LinkedIn

Redis © 2026 Redis Ltd. All rights reserved.