LangSmith

How to Improve LLM Applications in Production

Presented by

Greg Loughnane, Founder & CEO
Chris Alexiuk, Co-Founder & CTO

🎯 ALIGNING OUR AIM

lide

AI MAKERSPACE

BY THE END OF TODAY...

- **Baseline production RAG** apps w/ **evaluation** & **E2E testing**
- **Improve RAG** re; eval **metrics** and **feedback/annotation** on examples

OVERVIEW

- 🔗 LangChain RAG
- 🔎 Monitoring/Evaluation
- 👷 Baseline RAG

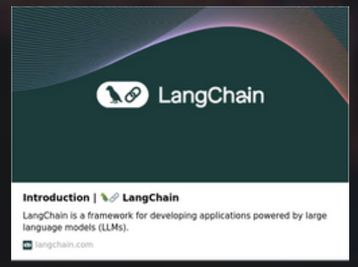
- 👍 Annotation/feedback
- 📈 Advanced retrieval
- 👷 Improved RAG

- ❓ Conclusions, QA

AI Engineering

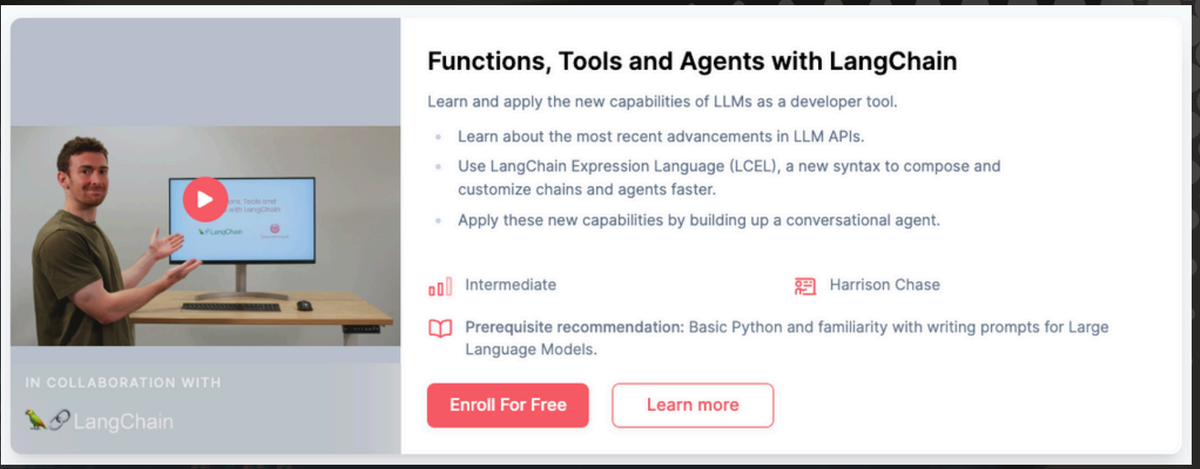
Data Science

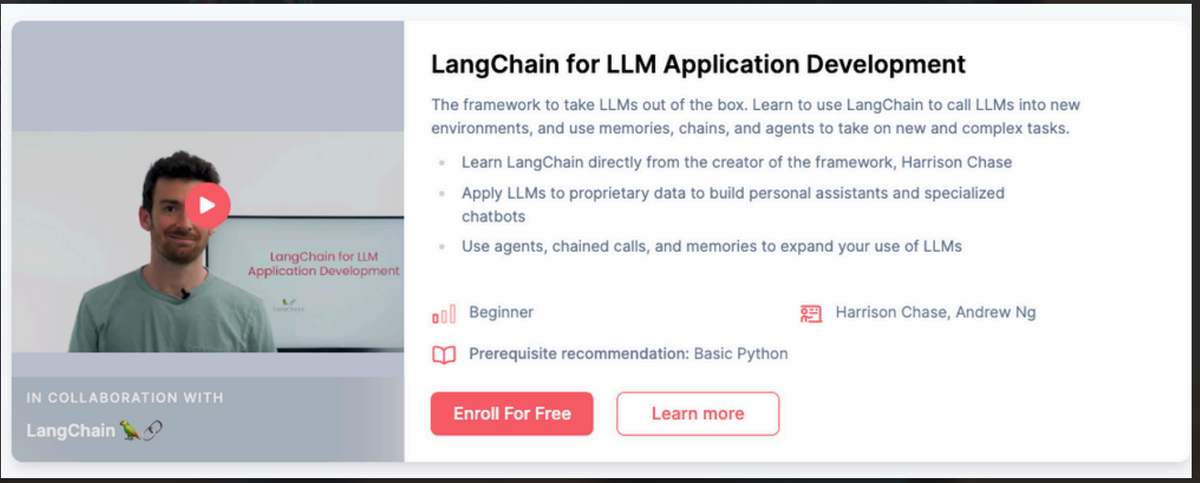
AI ENGINEERING

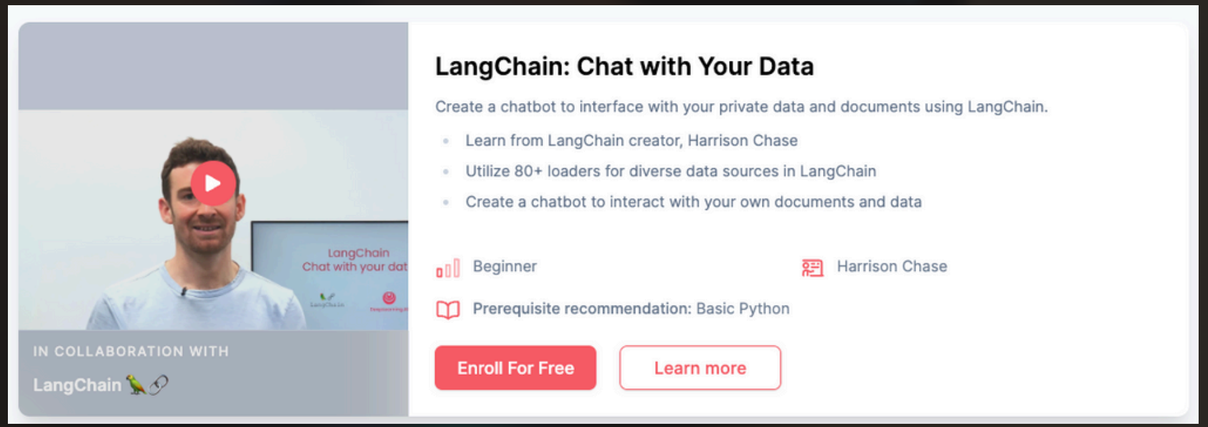


"The real power comes when you can **combine** them with **other sources** of **computation** or **knowledge.**"

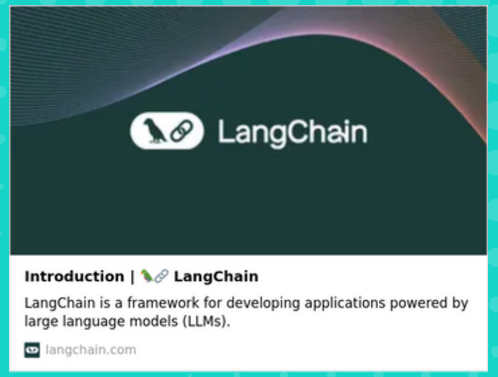
~ Harrison Chase
Creator of LangChain







WHAT IS IT?



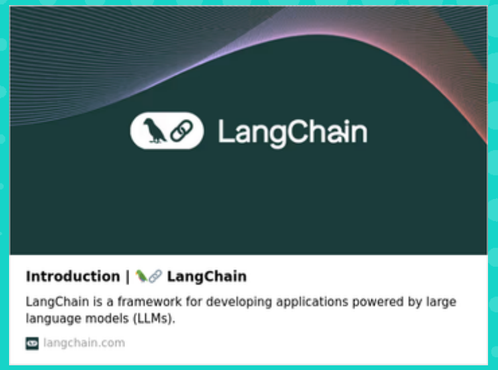
🦜🔗 LangChain

LangChain is a framework for developing applications powered by language models. We believe that the most powerful and differentiated applications will not only call out to a language model via an api, but will also:

1. Be data-aware: connect a language model to other sources of data
2. Be agentic: Allow a language model to interact with its environment

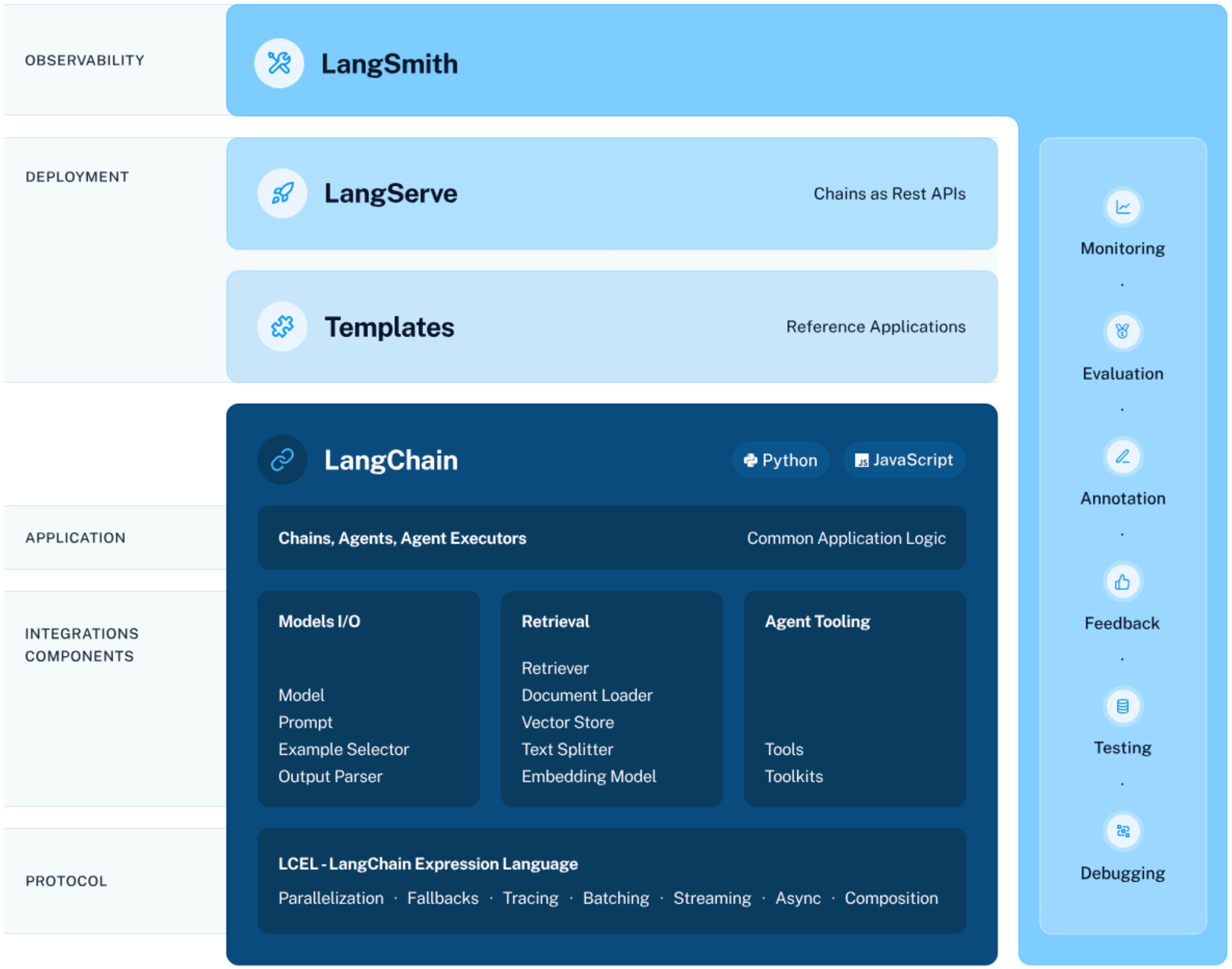
As such, the LangChain framework is designed with the objective in mind to enable those types of applications.

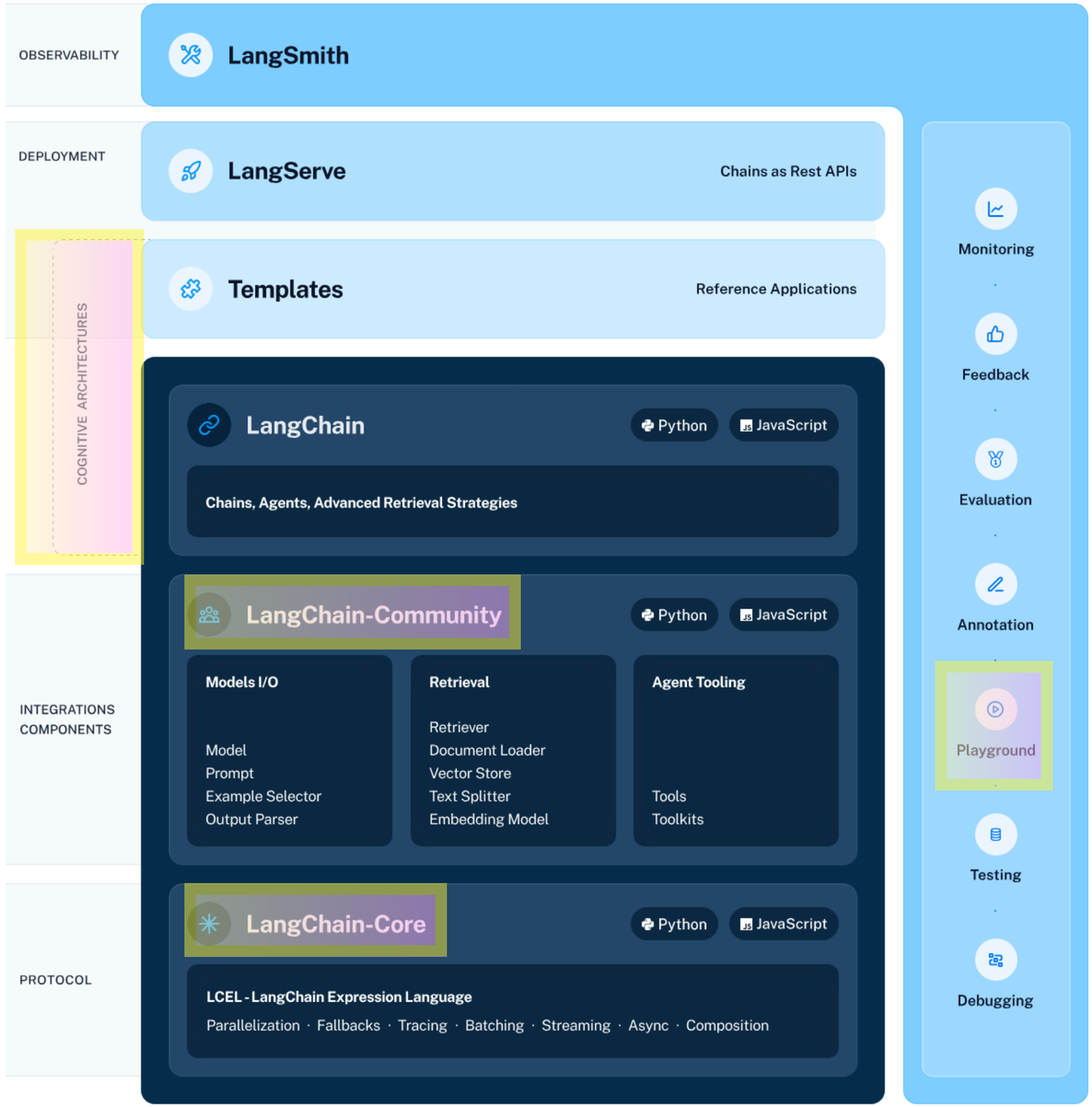
WHAT IS IT?

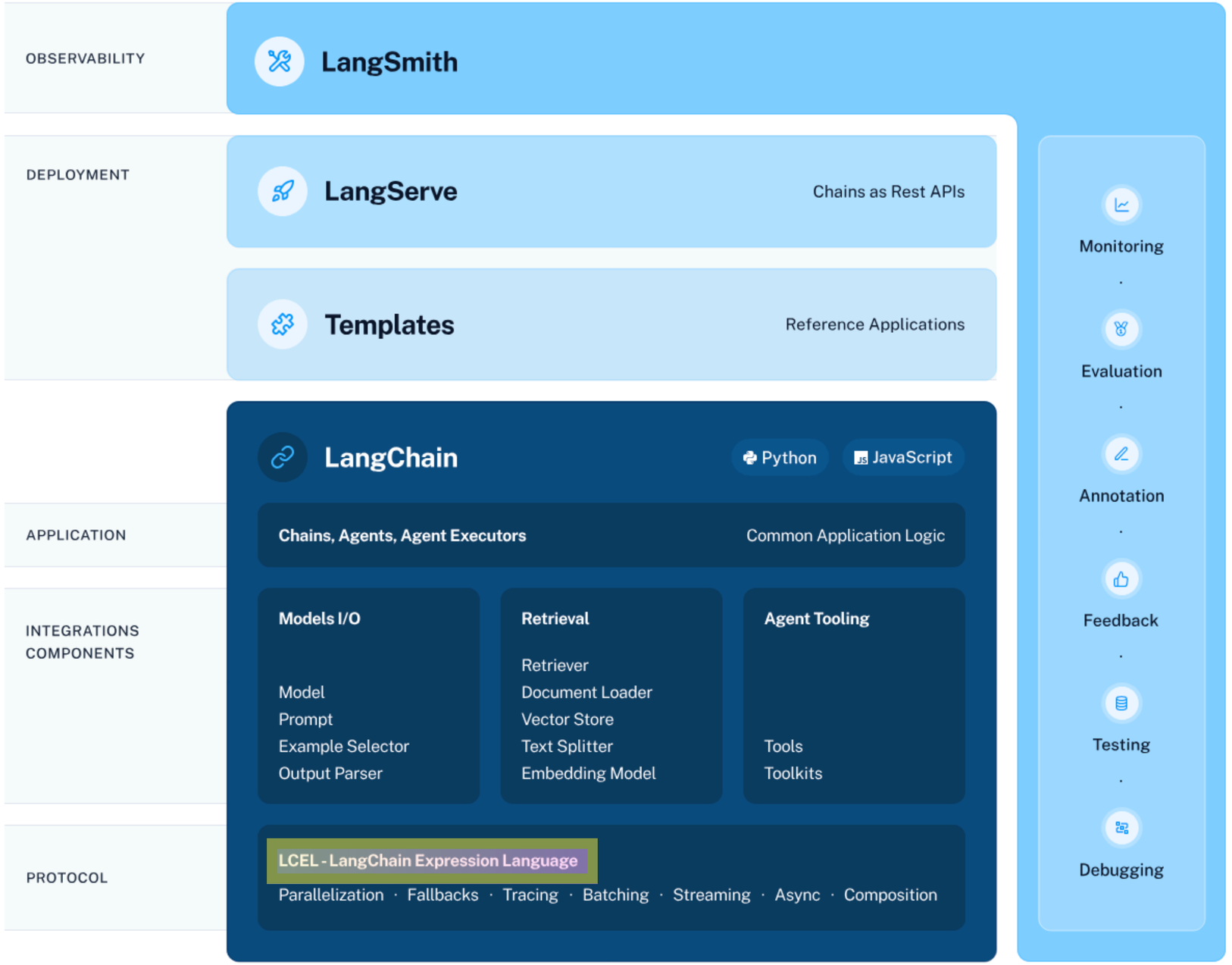


LangChain is a framework for developing applications powered by language models. It enables applications that:

- **Are context-aware**: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
- **Reason**: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
APPLICATION
LangChain
Python
JavaScript
Chains, Agents, Agent Executors
Common Application Logic
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
PROTOCOL
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Evaluation
Annotation
Feedback
Testing
Debugging


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
COGNITIVE ARCHITECTURES
LangChain
Python
JavaScript
Chains, Agents, Advanced Retrieval Strategies
LangChain-Community
Python
JavaScript
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
LangChain-Core
Python
JavaScript
PROTOCOL
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Feedback
Evaluation
Annotation
Playground
Testing
Debugging

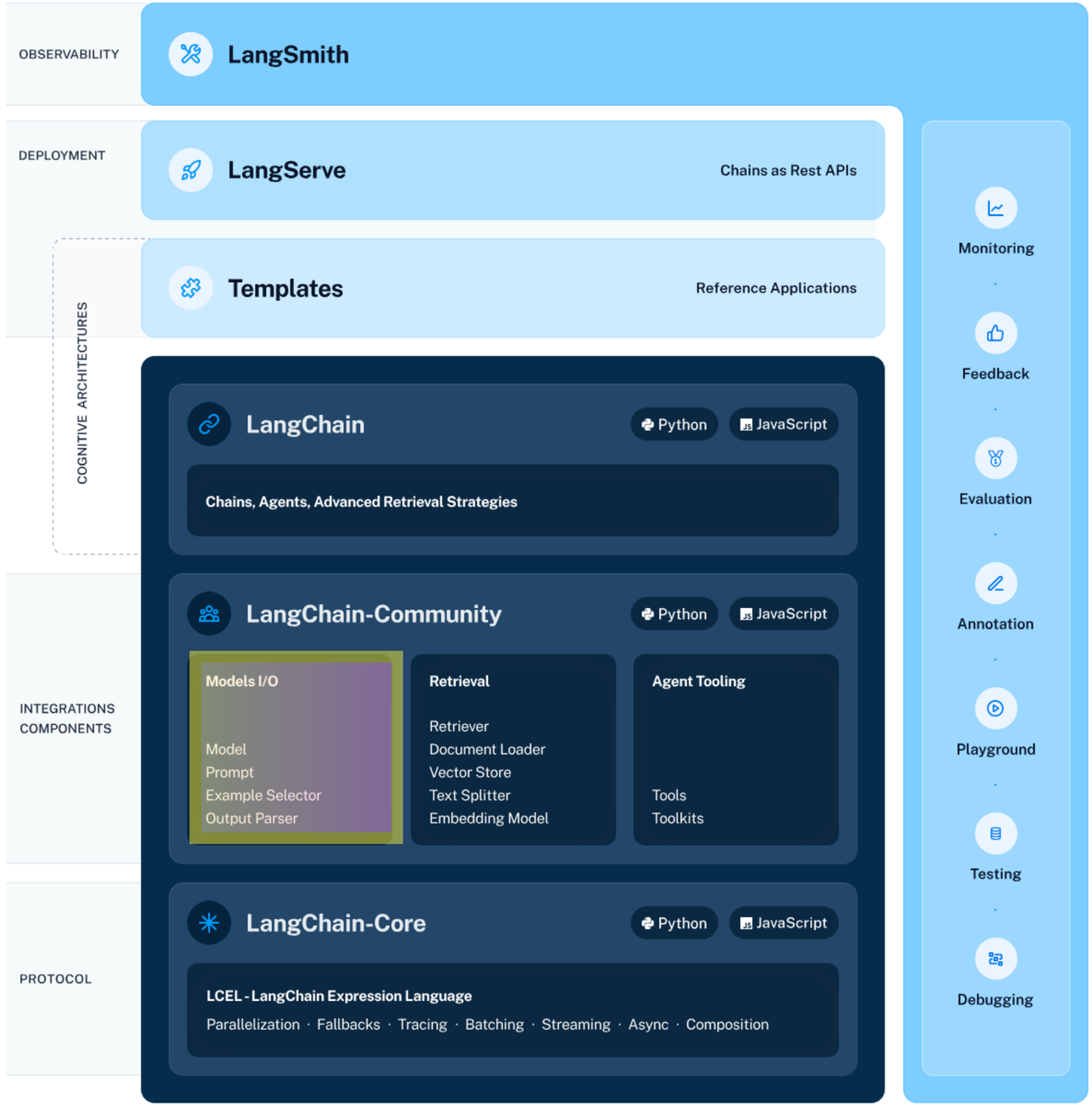

OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
LangChain
Python
JavaScript
APPLICATION
Chains, Agents, Agent Executors
Common Application Logic
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
PROTOCOL
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Evaluation
Annotation
Feedback
Testing
Debugging

LangChain Expression Language

- *Declarative way to **easily compose chains** with elegance*
- *Handles all the finicky bits*
  - *Streaming*
  - *Async*
  - *Parallel execution*
  - *Retries & fallbacks*
  - *Access intermediate results*
  - *I/O schemas*
  - ***LangSmith tracing***
  - *LangServe deployment*

```
llm_chain = LLMChain(prompt=prompt, llm=llm)
```

```
llm_chain = prompt | llm
```


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
COGNITIVE ARCHITECTURES
LangChain
Python
JavaScript
Chains, Agents, Advanced Retrieval Strategies
LangChain-Community
Python
JavaScript
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
INTEGRATIONS
COMPONENTS
PROTOCOL
LangChain-Core
Python
JavaScript
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Feedback
Evaluation
Annotation
Playground
Testing
Debugging

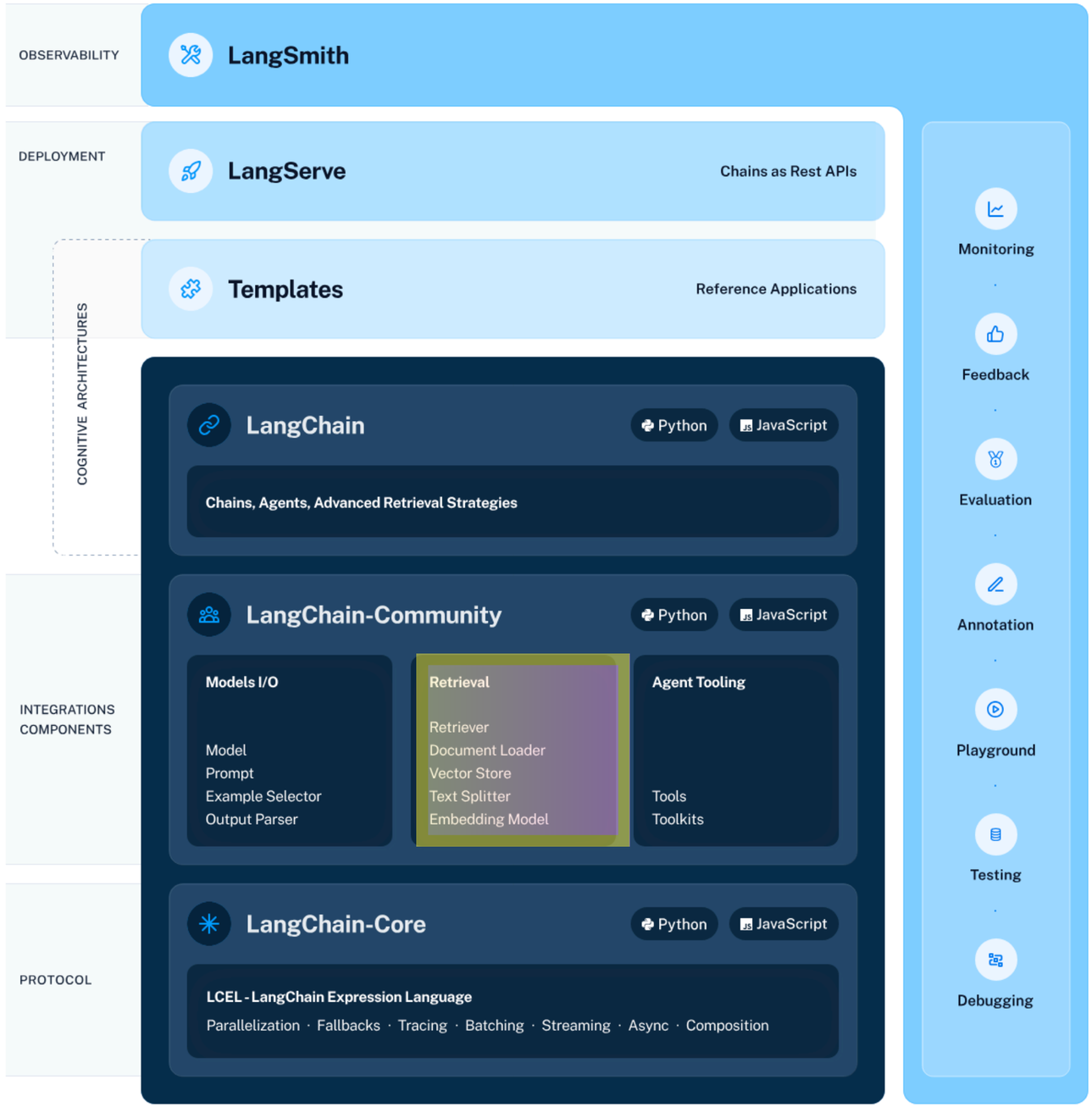
MODELS

- *"**Chat**-Style"*
  - *Fine-tuned for **chat***
  - *Often **instruction**-tuned*

OpenAI

LangChain

System Message	System Message
User Message	**Human** Message
Assistant Message	**AI** Message


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
COGNITIVE ARCHITECTURES
LangChain
Python
JavaScript
Chains, Agents, Advanced Retrieval Strategies
LangChain-Community
Python
JavaScript
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
LangChain-Core
Python
JavaScript
PROTOCOL
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Feedback
Evaluation
Annotation
Playground
Testing
Debugging

RETREIVAL AUGMENTED GENERATION

Retrieval

- **Finds references** in your documents

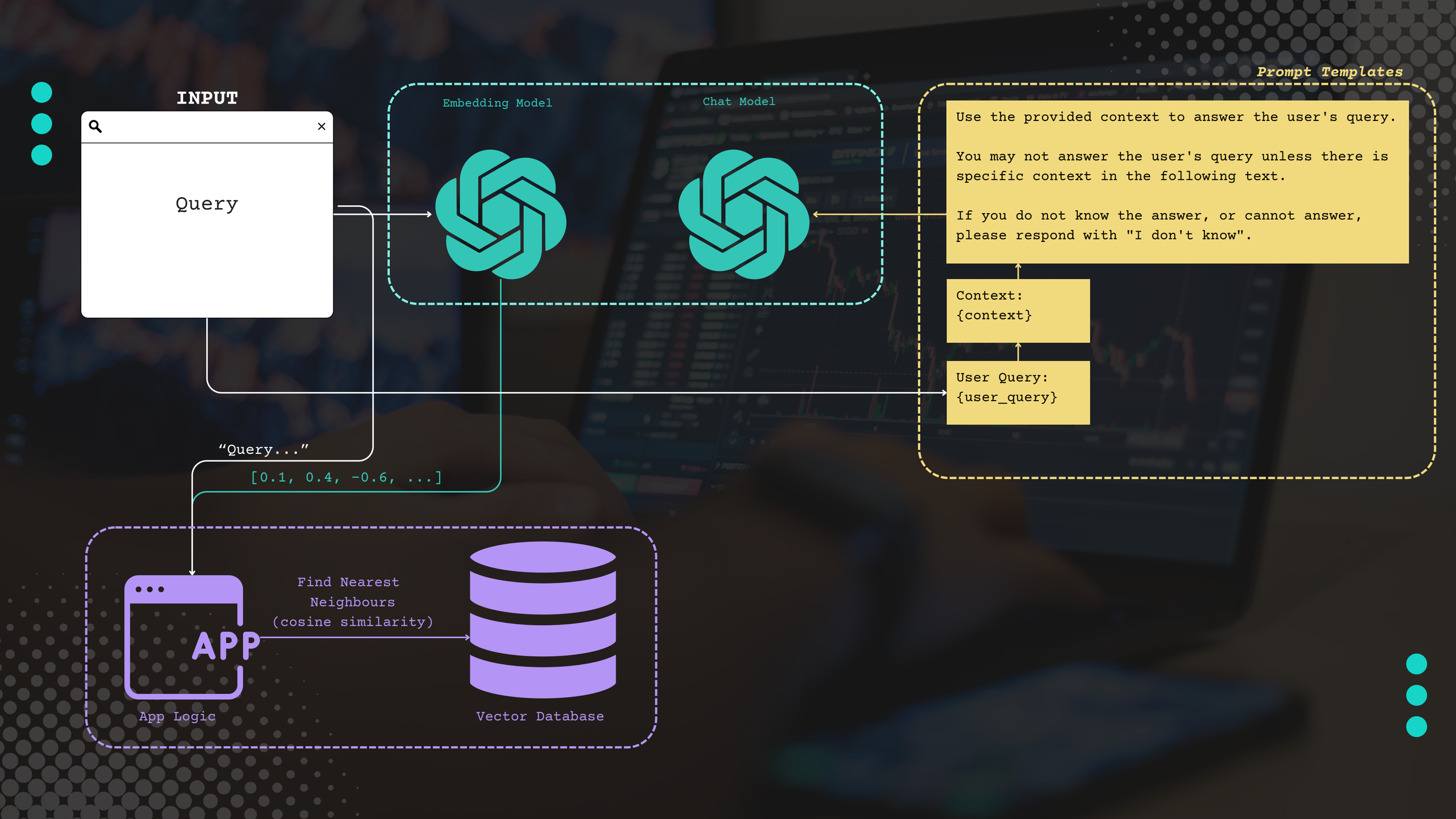
Augmented

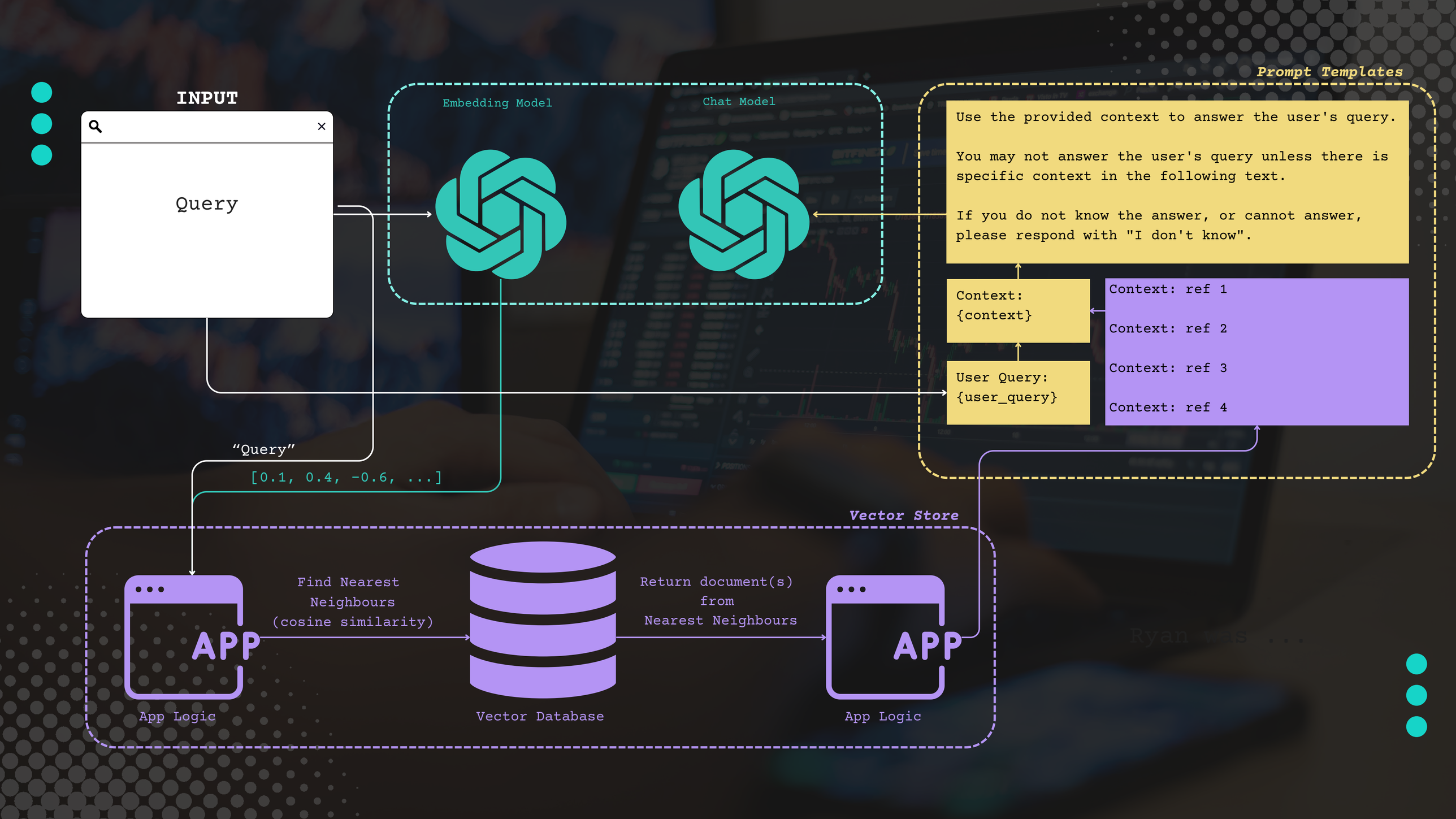
- **Adds references** to your prompts

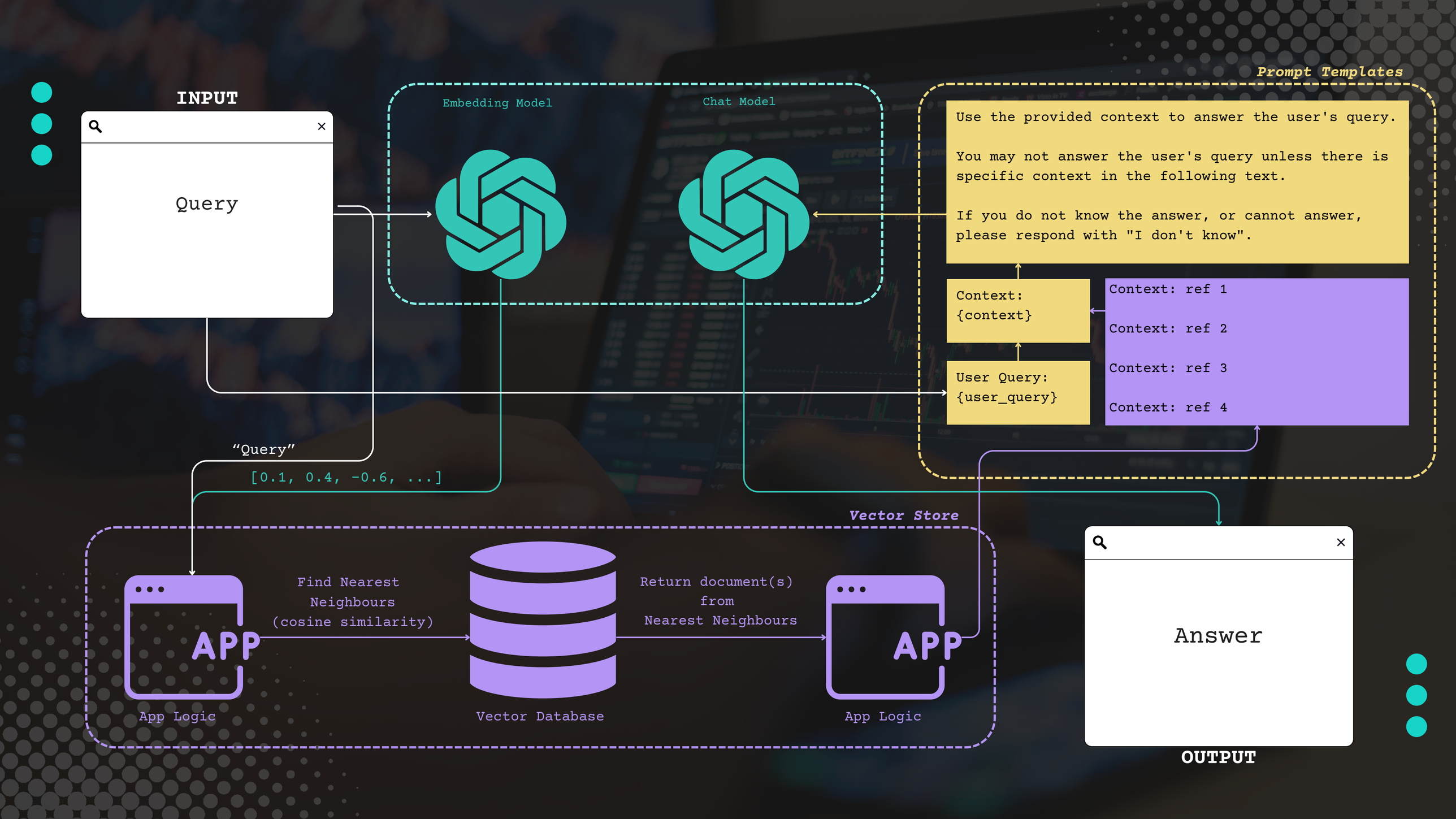
Generation

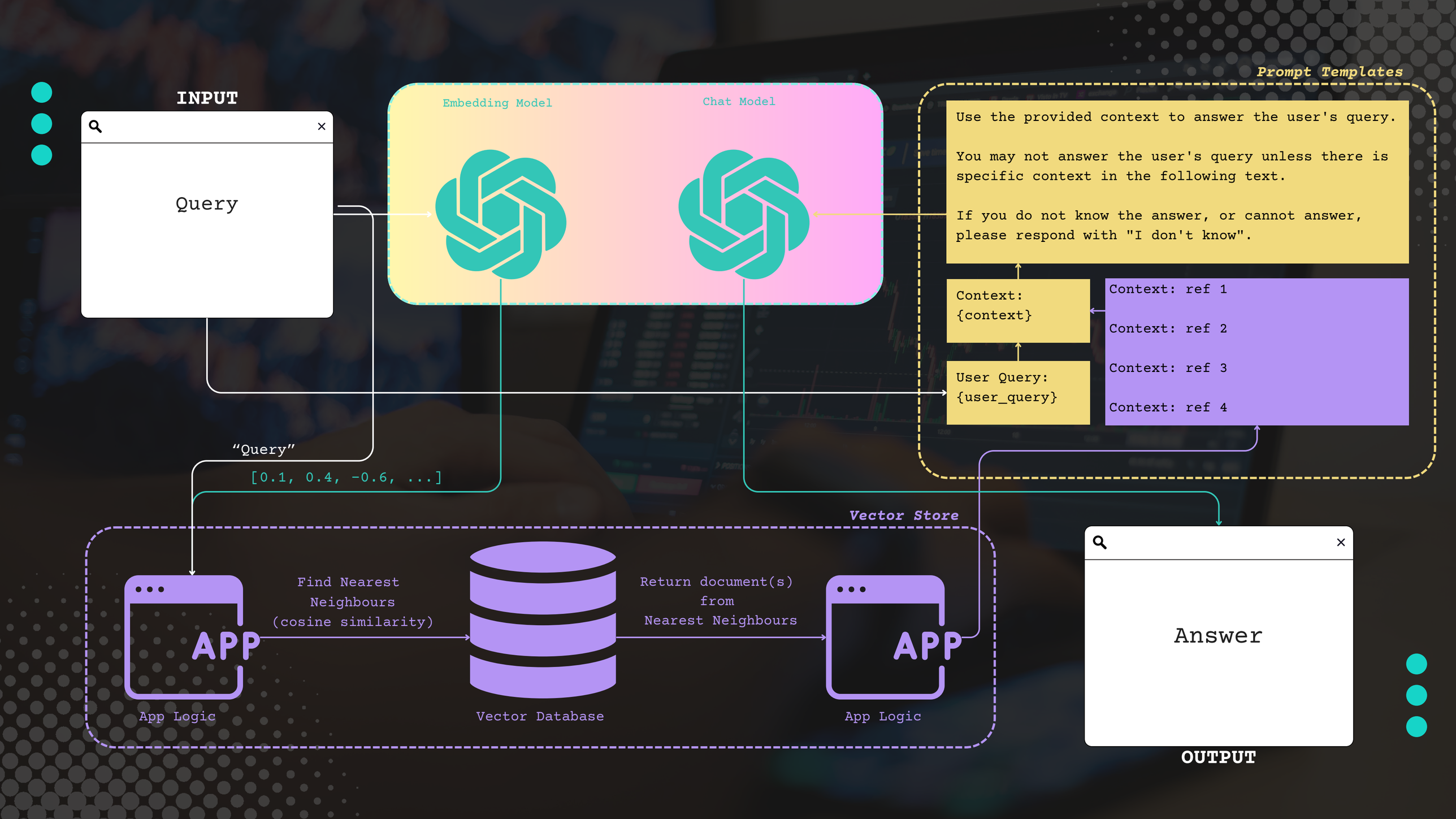
- **Improves answers** to questions!

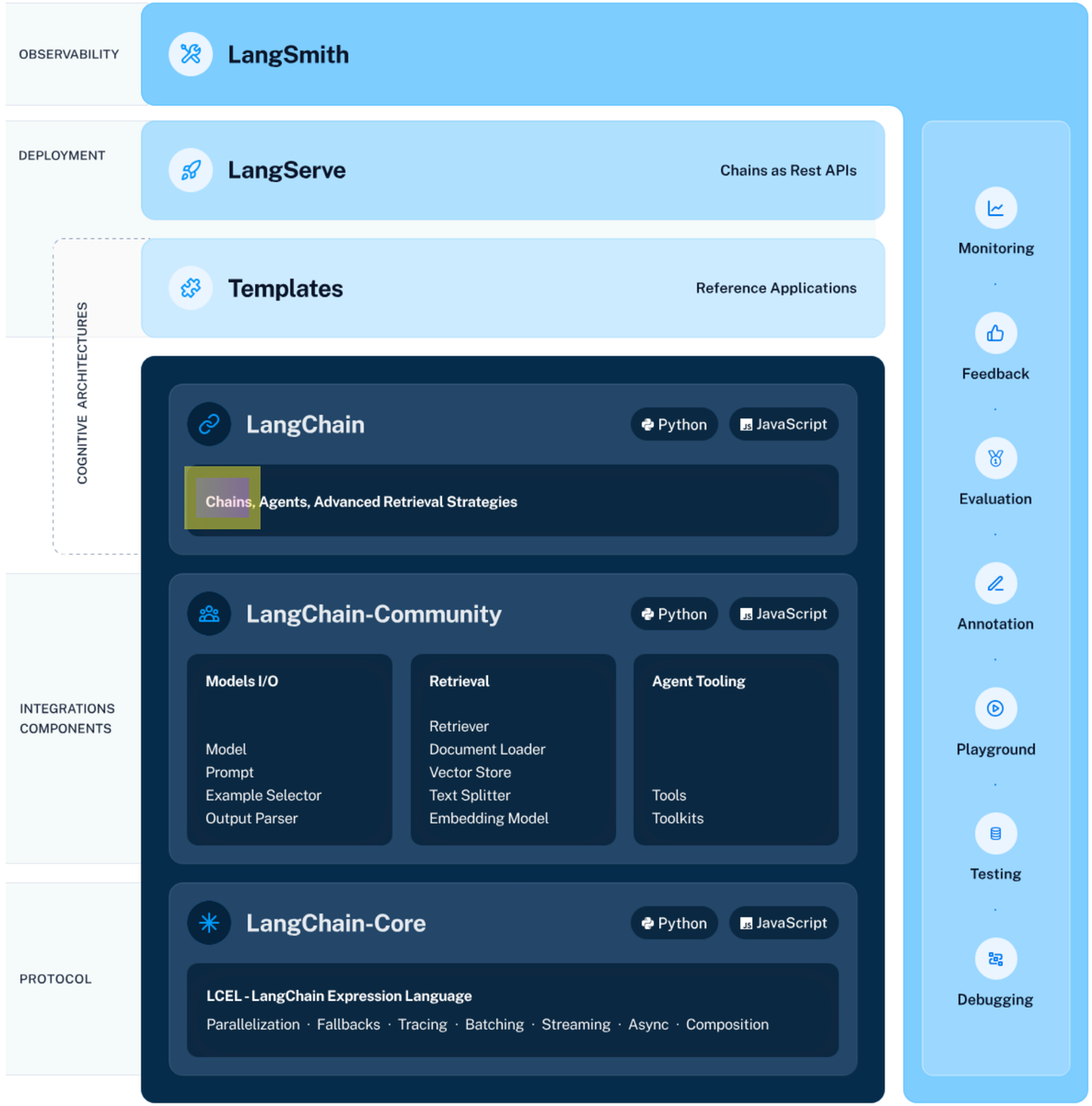

INPUT
Query
Embedding Model
"Query..."
[0.1, 0.4, -0.6, ...]
APP
App Logic
Find Nearest
Neighbours
(cosine similarity)
Vector Database


INPUT
Query
Embedding Model
Chat Model
Prompt Templates
Use the provided context to answer the user's query.
You may not answer the user's query unless there is specific context in the following text.
If you do not know the answer, or cannot answer, please respond with "I don't know".
Context:
{context}
User Query:
{user_query}
“Query...”
[0.1, 0.4, -0.6, ...]
APP
Find Nearest
Neighbours
(cosine similarity)
App Logic
Vector Database


INPUT
Query
Embedding Model
Chat Model
Prompt Templates
Use the provided context to answer the user's query.
You may not answer the user's query unless there is specific context in the following text.
If you do not know the answer, or cannot answer, please respond with "I don't know".
Context: {context}
User Query: {user_query}
Context: ref 1
Context: ref 2
Context: ref 3
Context: ref 4
"Query"
[0.1, 0.4, -0.6, ...]
Vector Store
APP
App Logic
Find Nearest Neighbours (cosine similarity)
Vector Database
Return document(s) from Nearest Neighbours
APP
App Logic


INPUT
Query
Embedding Model
Chat Model
Prompt Templates
Use the provided context to answer the user's query.
You may not answer the user's query unless there is specific context in the following text.
If you do not know the answer, or cannot answer, please respond with "I don't know".
Context: {context}
User Query: {user_query}
Context: ref 1
Context: ref 2
Context: ref 3
Context: ref 4
“Query”
[0.1, 0.4, -0.6, ...]
Vector Store
APP
App Logic
Find Nearest Neighbours (cosine similarity)
Vector Database
Return document(s) from Nearest Neighbours
APP
App Logic
Answer
OUTPUT


INPUT
Query
Embedding Model
Chat Model
Prompt Templates
Use the provided context to answer the user's query.
You may not answer the user's query unless there is specific context in the following text.
If you do not know the answer, or cannot answer, please respond with "I don't know".
Context: {context}
User Query: {user_query}
Context: ref 1
Context: ref 2
Context: ref 3
Context: ref 4
"Query"
[0.1, 0.4, -0.6, ...]
Vector Store
APP
App Logic
Find Nearest Neighbours (cosine similarity)
Vector Database
Return document(s) from Nearest Neighbours
APP
App Logic
Answer
OUTPUT


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
COGNITIVE ARCHITECTURES
LangChain
Python
JavaScript
Chains, Agents, Advanced Retrieval Strategies
LangChain-Community
Python
JavaScript
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
PROTOCOL
LangChain-Core
Python
JavaScript
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Feedback
Evaluation
Annotation
Playground
Testing
Debugging

DEF: CHAIN

A sequence of calls to other components

$$\text{LLM Chain} = \text{Chat Model} + \text{Prompt Template}$$

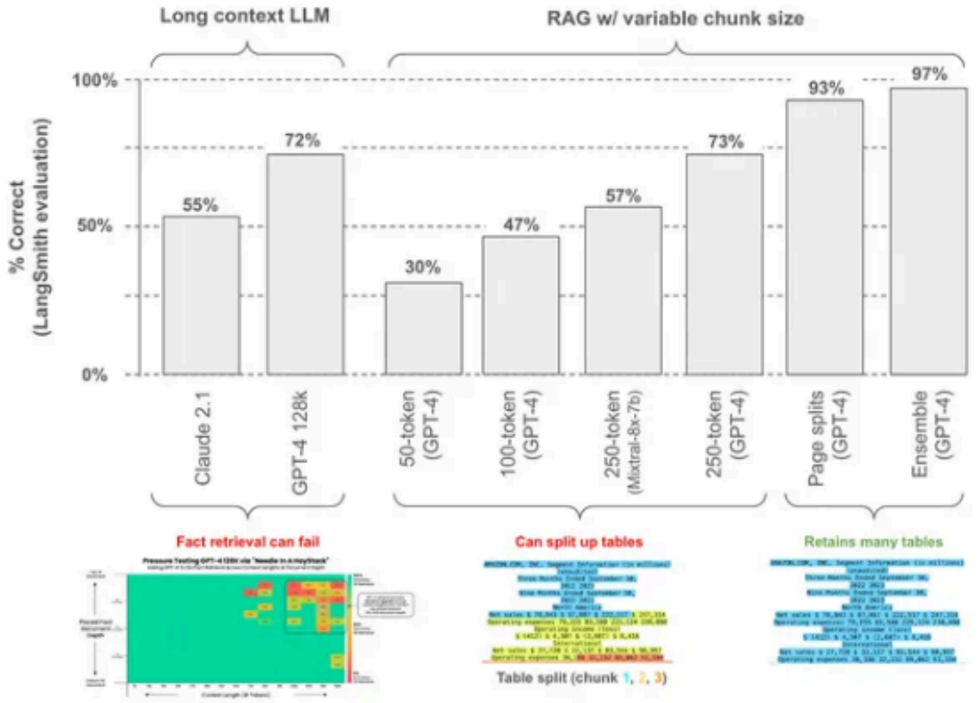
```
chain = LLMChain(llm=chat_model, prompt=chat_prompt)
```

```
llm_chain = prompt | llm
```

TODAY'S BUILD

LangChain Blog RAG Flow

1. **Search** LangChain blog docs for **top k** resources
2. **Ask** specific **questions** related to content
3. **Return answers** to questions with sources



Benchmarking RAG on tables

Key links LangChain public benchmark evaluation notebooks: * Long context LLMs here * Chunk size tuning here * Multi vector with ensemble here Motivation Retrieval augmented generation (RAG) is one of the mo...

LangChain Blog / Dec 13, 2023

OUR MODELS

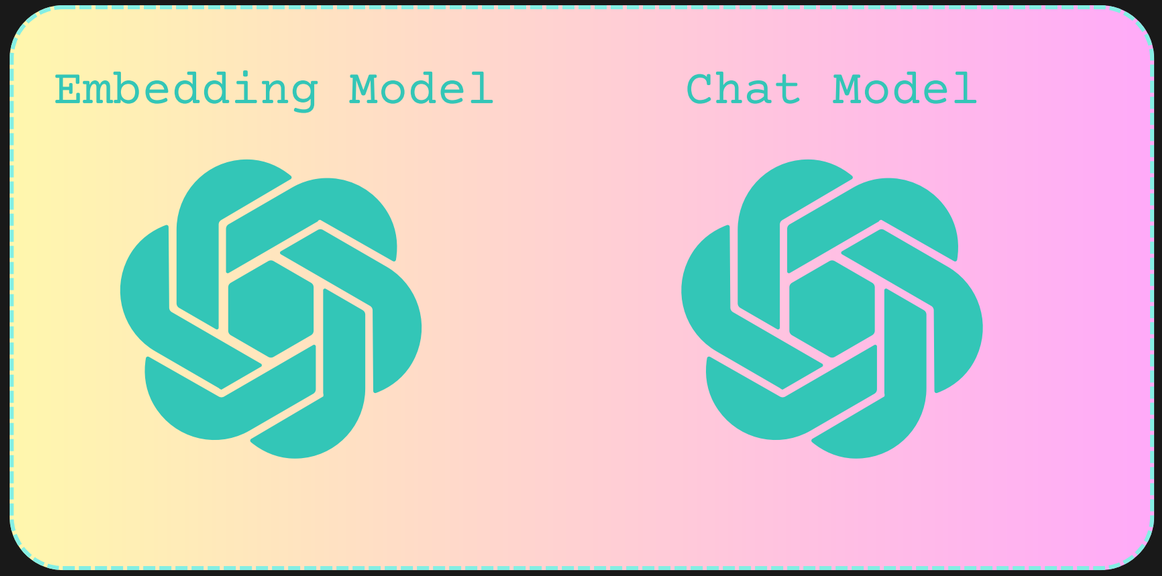
EMBEDDING & CHAT (LLM) MODEL

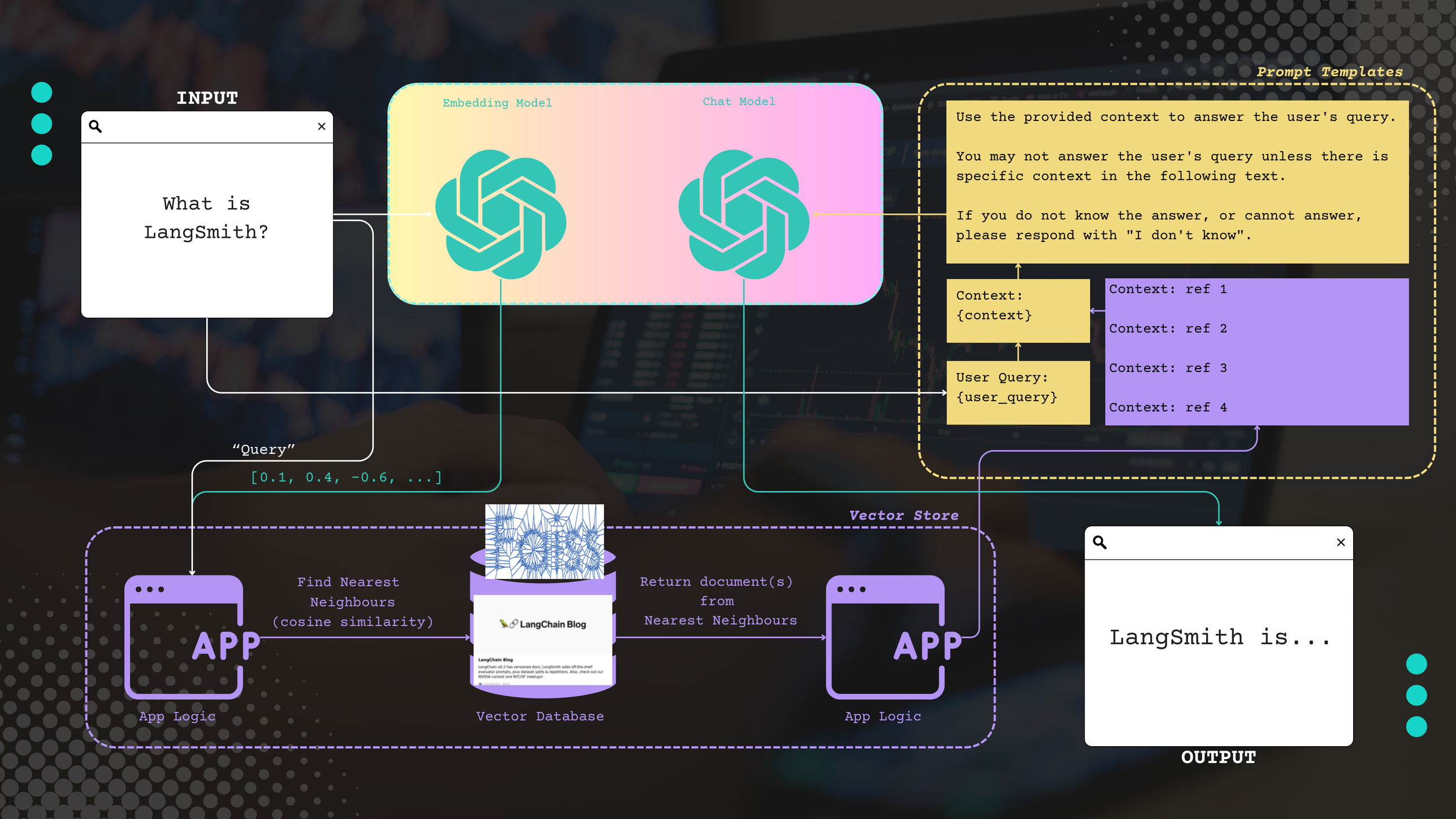
Chat Model (e.g., LLM)

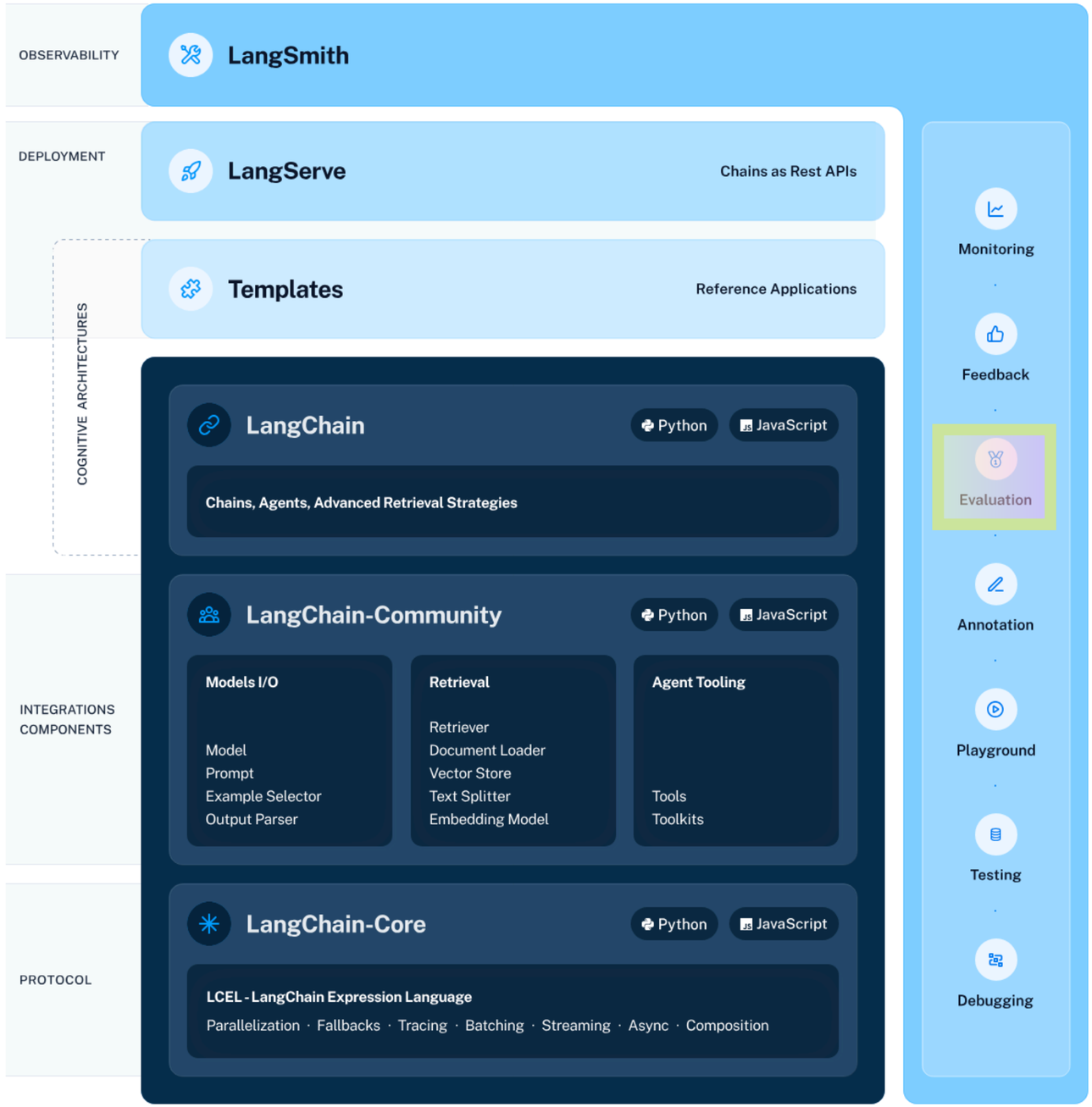
- e.g,. OpenAI GPT-3.5 Turbo

Embedding Model

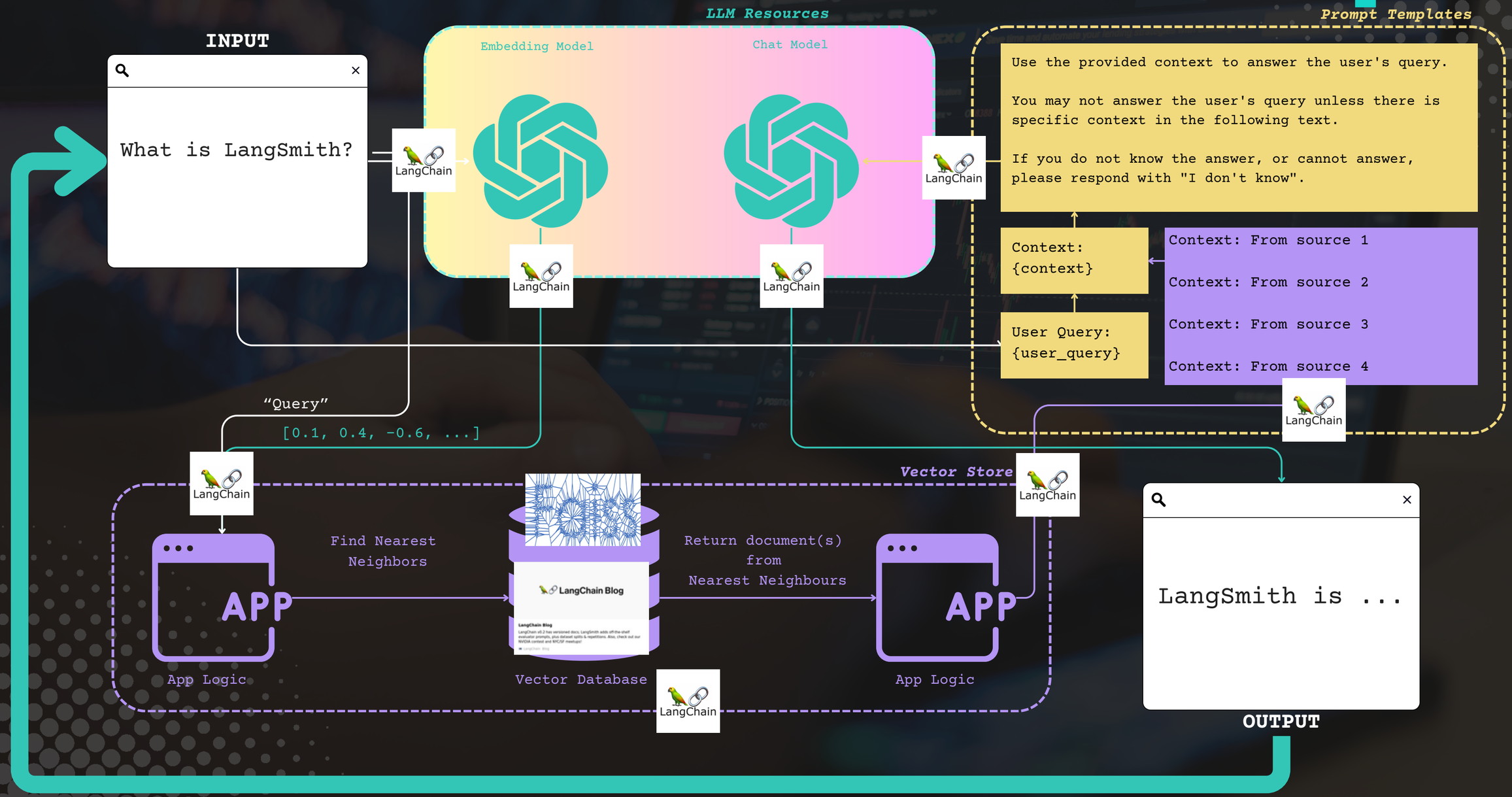
- e.g., Open AI’s Ada




INPUT
What is LangSmith?
Embedding Model
Chat Model
Prompt Templates
Use the provided context to answer the user's query.
You may not answer the user's query unless there is specific context in the following text.
If you do not know the answer, or cannot answer, please respond with "I don't know".
Context: {context}
User Query: {user_query}
Context: ref 1
Context: ref 2
Context: ref 3
Context: ref 4
"Query"
[0.1, 0.4, -0.6, ...]
Vector Store
APP
App Logic
Find Nearest Neighbours (cosine similarity)
LangChain Blog
Vector Database
Return document(s) from Nearest Neighbours
APP
App Logic
LangSmith is...
OUTPUT


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
COGNITIVE ARCHITECTURES
Templates
Reference Applications
LangChain
Python
JavaScript
Chains, Agents, Advanced Retrieval Strategies
INTEGRATIONS
COMPONENTS
LangChain-Community
Python
JavaScript
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
PROTOCOL
LangChain-Core
Python
JavaScript
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Feedback
Evaluation
Annotation
Playground
Testing
Debugging

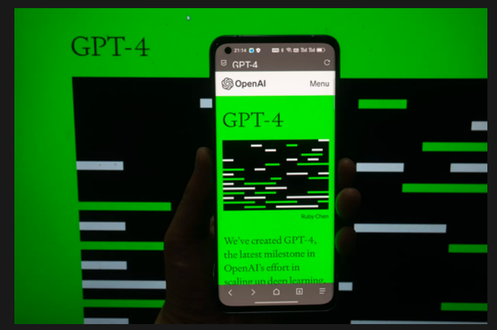
Evaluation Feedback Loop



RAG EVALUATION

Generally we have 4 pieces of data...

Question | Result | Context | Answer



CHAIN-OF-THOUGH QA

- **Instructs** the LLMChain to use **chain of thought** "reasoning"
- Responses **correlate better** with **human** labels
- **Higher** token and runtime **cost**.

```
_PROMPT_TEMPLATE = """You are an expert professor
You are grading the following question:
{query}
Here is the real answer:
{answer}
You are grading the following predicted answer:
{result}
Respond with CORRECT or INCORRECT:
Grade:
"""

PROMPT = PromptTemplate(
    input_variables=["query", "answer", "result"],
)
```

BASELINING RAG

Presented by

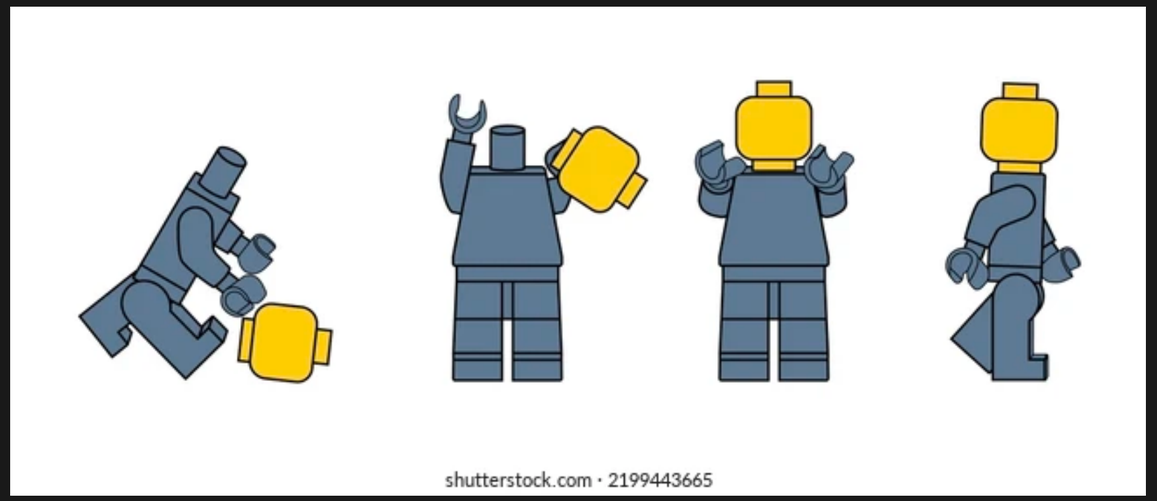
Chris Alexiuk, LLM Wizard 🪄

DATA SCIENCE

THE AGE OF THE AI ENGINEER

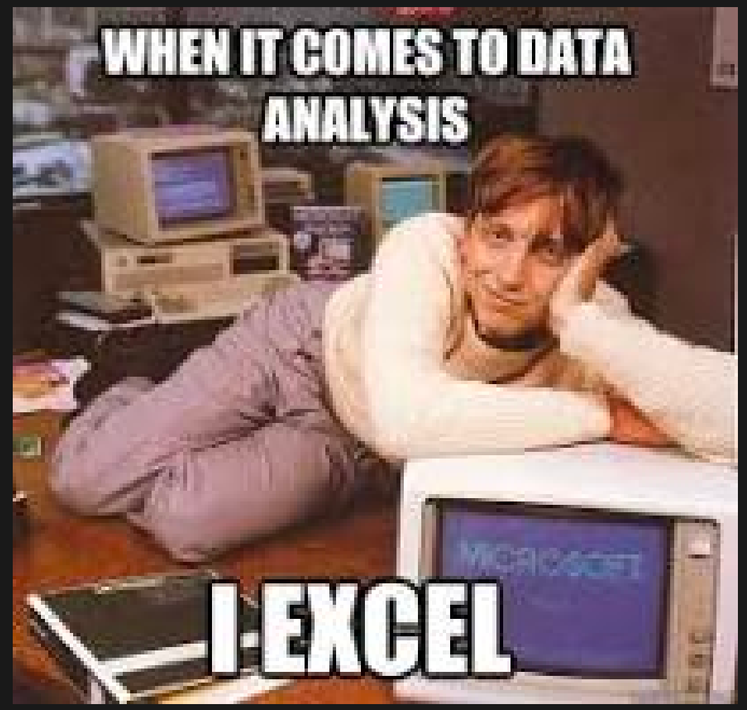
“A wide range of AI tasks that used to take 5 years and a research team to accomplish in 2013, now just require API docs and a spare afternoon in 2023.”

It is now possible to build what used to take months in a single day!



DATA SCIENTISTS!

- **Evaluation**
- **Enhance Retrieval** (and thus Generation!)
- **Fine-Tuning**
  - Embeddings
  - Chat Models



ADVANCED RAG

From Simple to Advanced RAG

Table Stakes
Better Parsers
Chunk Sizes
Hybrid Search
Metadata Filters

Advanced Retrieval
Reranking
Recursive Retrieval
Embedded Tables
Small-to-big Retrieval

Fine-tuning
Embedding fine-tuning
LLM fine-tuning

Agentic Behavior
Routing
Query Planning
Multi-document Agents

Less Expressive
Easier to Implement
Lower Latency/Cost

More Expressive
Harder to Implement
Higher Latency/Cost

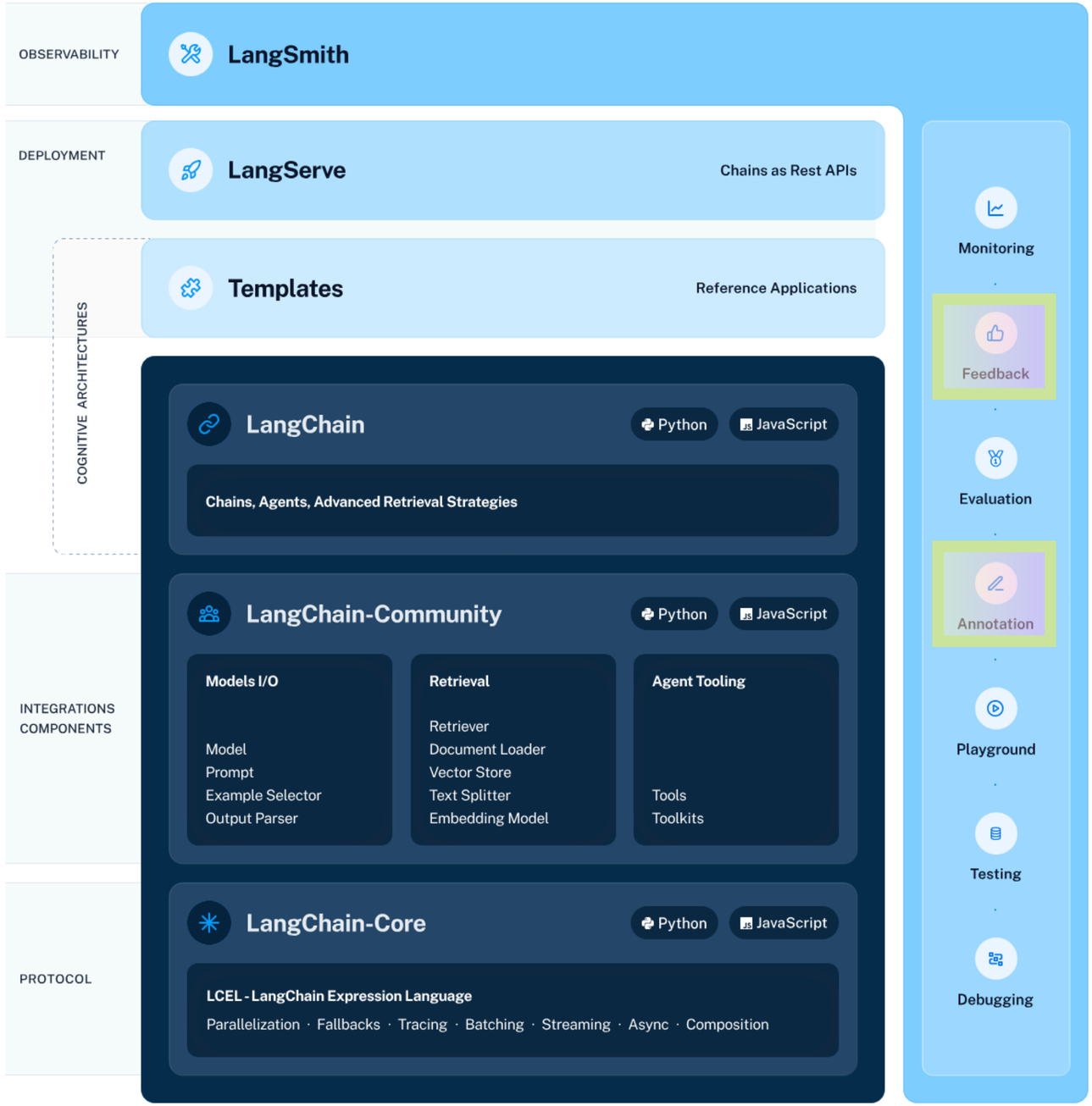
ENSEMBLE RETRIEVER

- A.k.a. “Hybrid Retrieval”

- Sparse retriever (BM25)
- Dense retriever (embedding similarity)

- Reciprocal Rank Fusion

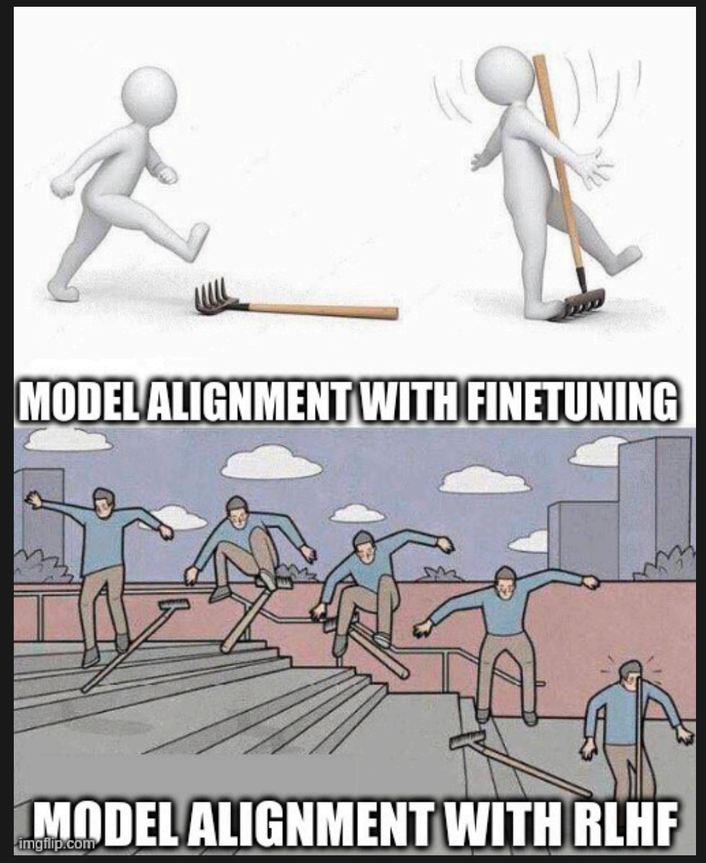

```
from langchain.retrievers import BM25Retriever, EnsembleRetriever
```


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
COGNITIVE ARCHITECTURES
LangChain
Python
JavaScript
Chains, Agents, Advanced Retrieval Strategies
LangChain-Community
Python
JavaScript
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
PROTOCOL
LangChain-Core
Python
JavaScript
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Feedback
Evaluation
Annotation
Playground
Testing
Debugging

ANNOTATION & FEEDBACK

- **Feedback**
  - As simple as 👍 / 👎
  - As complex as engaging users for **detailed assessments**

- **Annotation**
  - Devs can provide **qualitative feedback** manually prior to next deployment!



PROTOTYPING/IMPROVING LLM APPS

1. Prompt Engineering
2. Question Answering Systems
3. Fine-Tuning Models

FEW-SHOT

- **Zero**-shot
  - **Instruction** only; closest to human

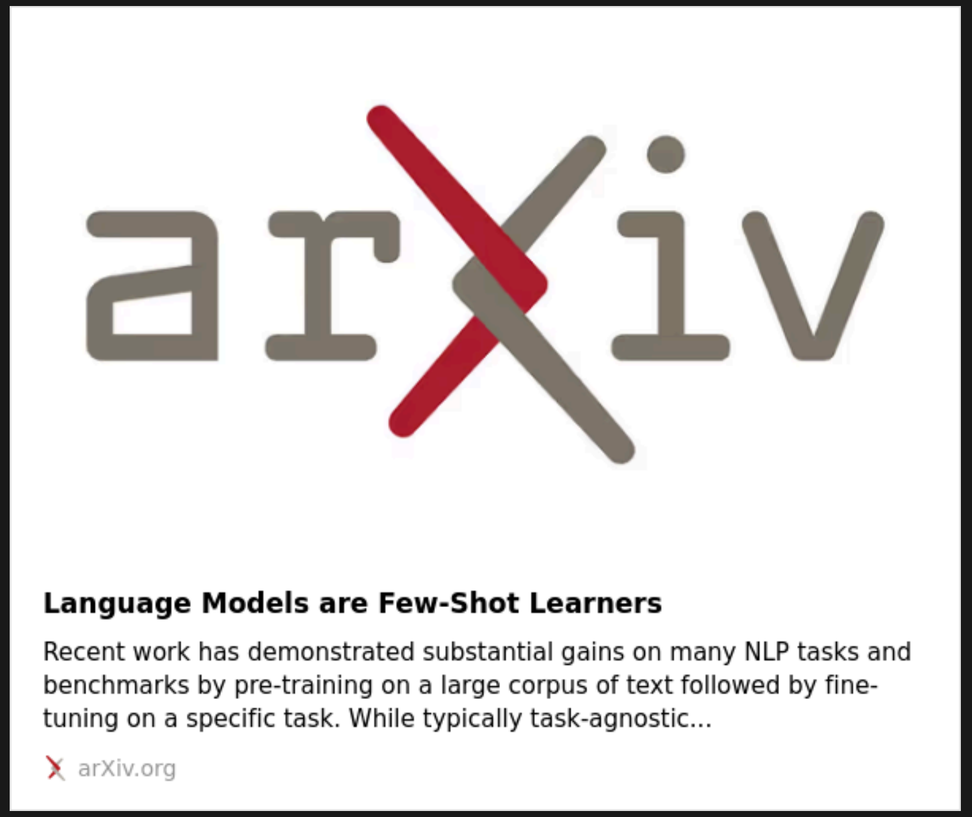
```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

- **One**-shot
  - Only **1** example

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

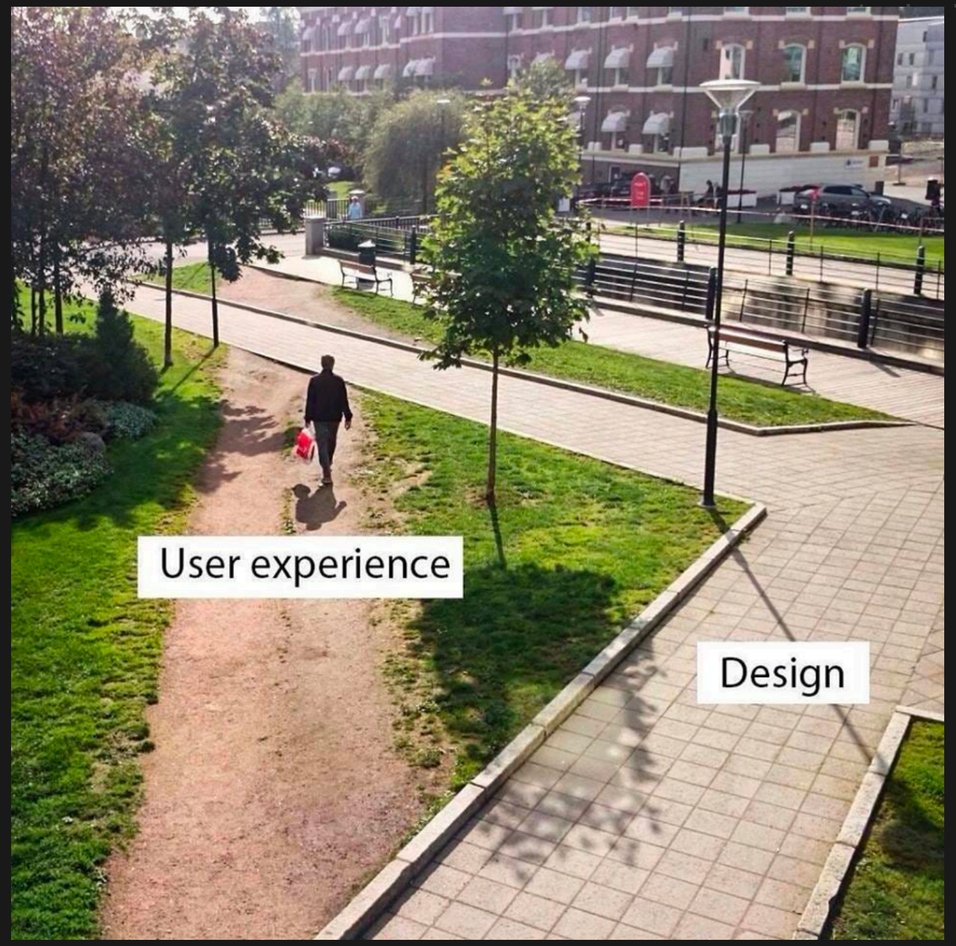
- **Few**-shot
  - Given a “few” demonstrations **(10-100)**

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                           ← prompt
```



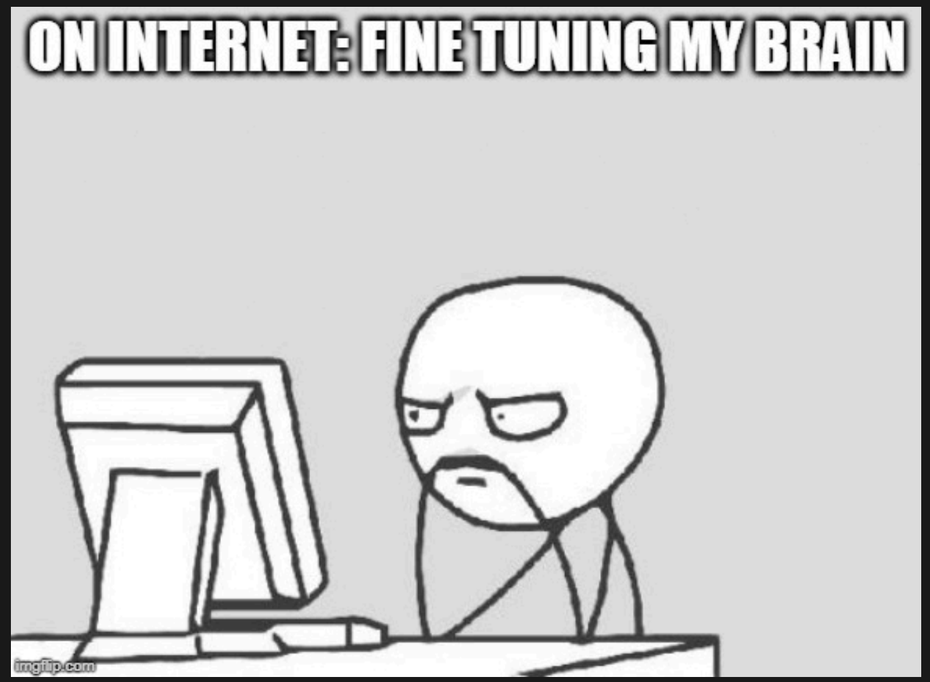
USER EXPERIENCE

- Control for **uncertainty** (PE)
- Build **guardrails** for steerability and safety (Harm/Help)

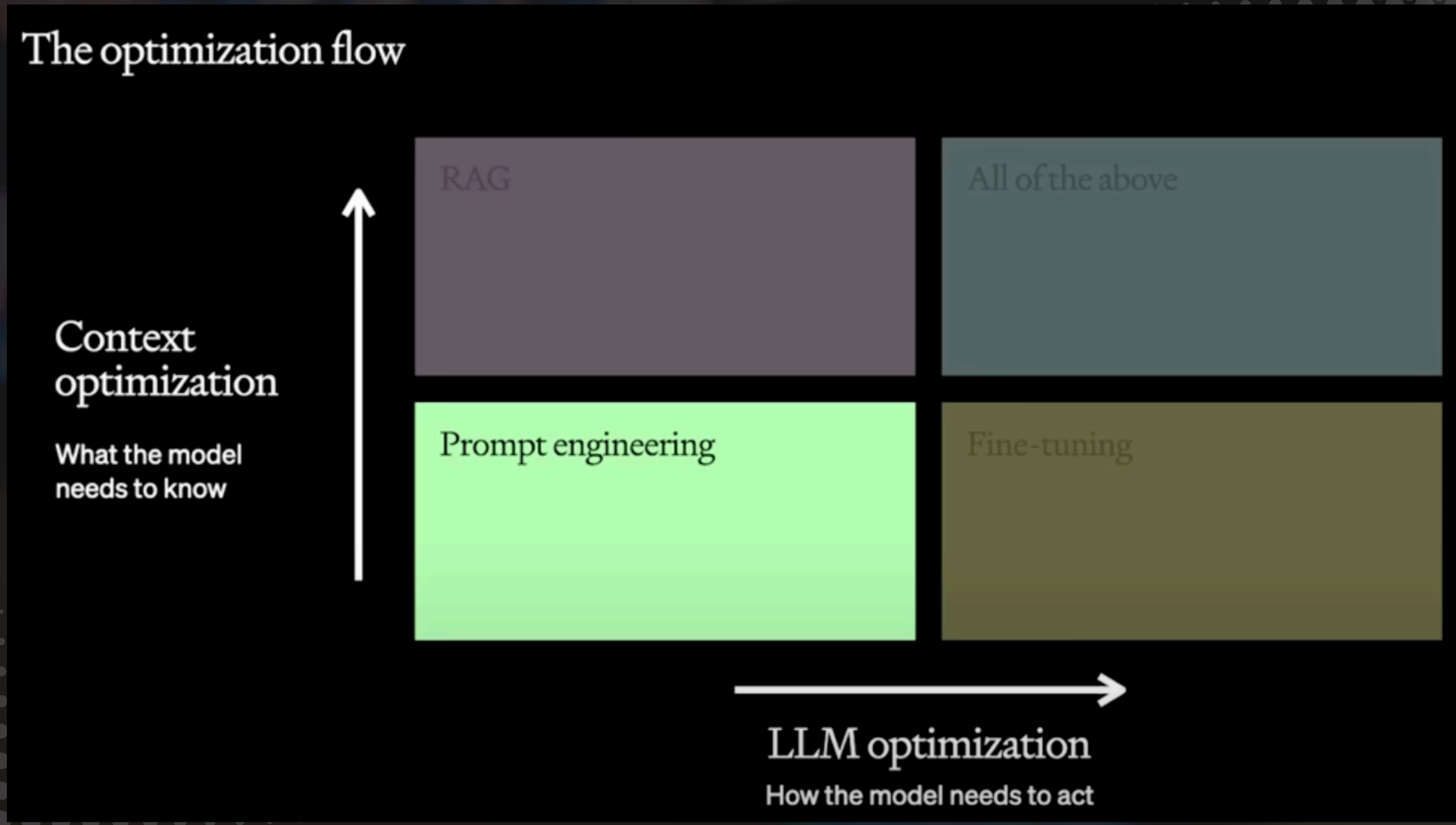


MODEL CONSISTENCY

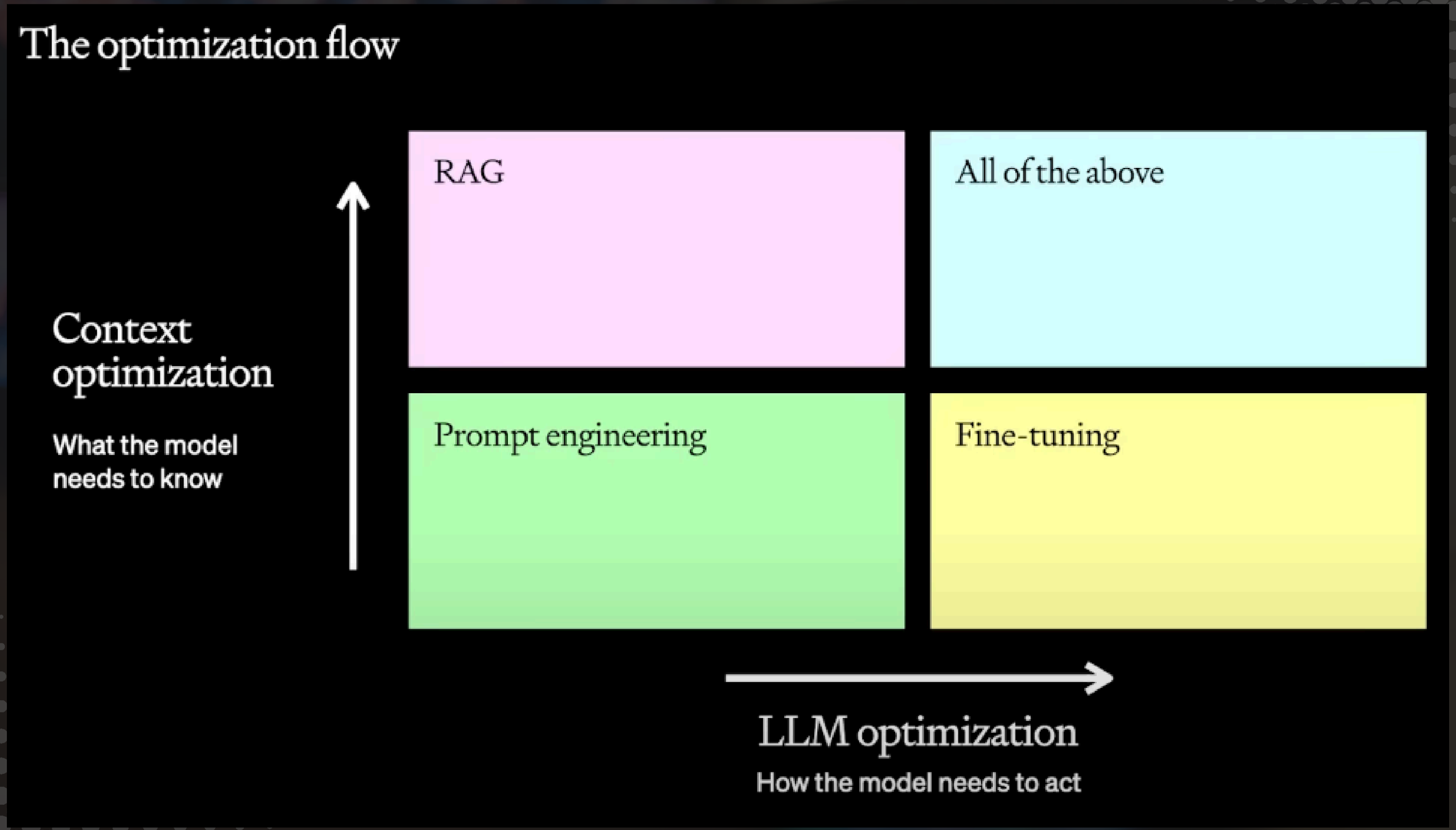
- **Constrain** model behavior (FT)
- **Ground** the model (RAG)



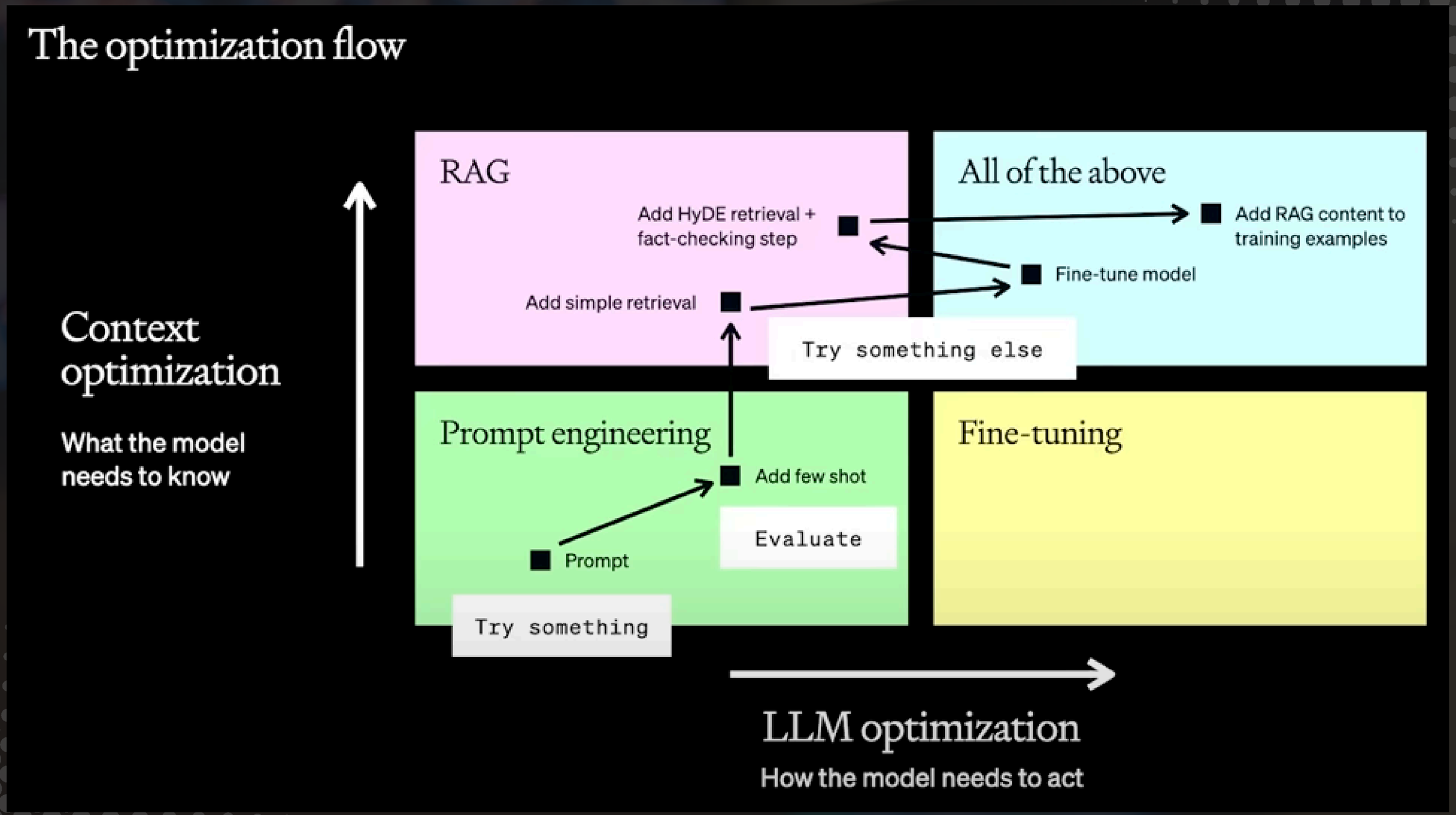
The optimization flow



The optimization flow



The optimization flow



LATENCY AND COST

- Use **semantic** caching (Prompts)
- Route to **cheaper** models (FT)

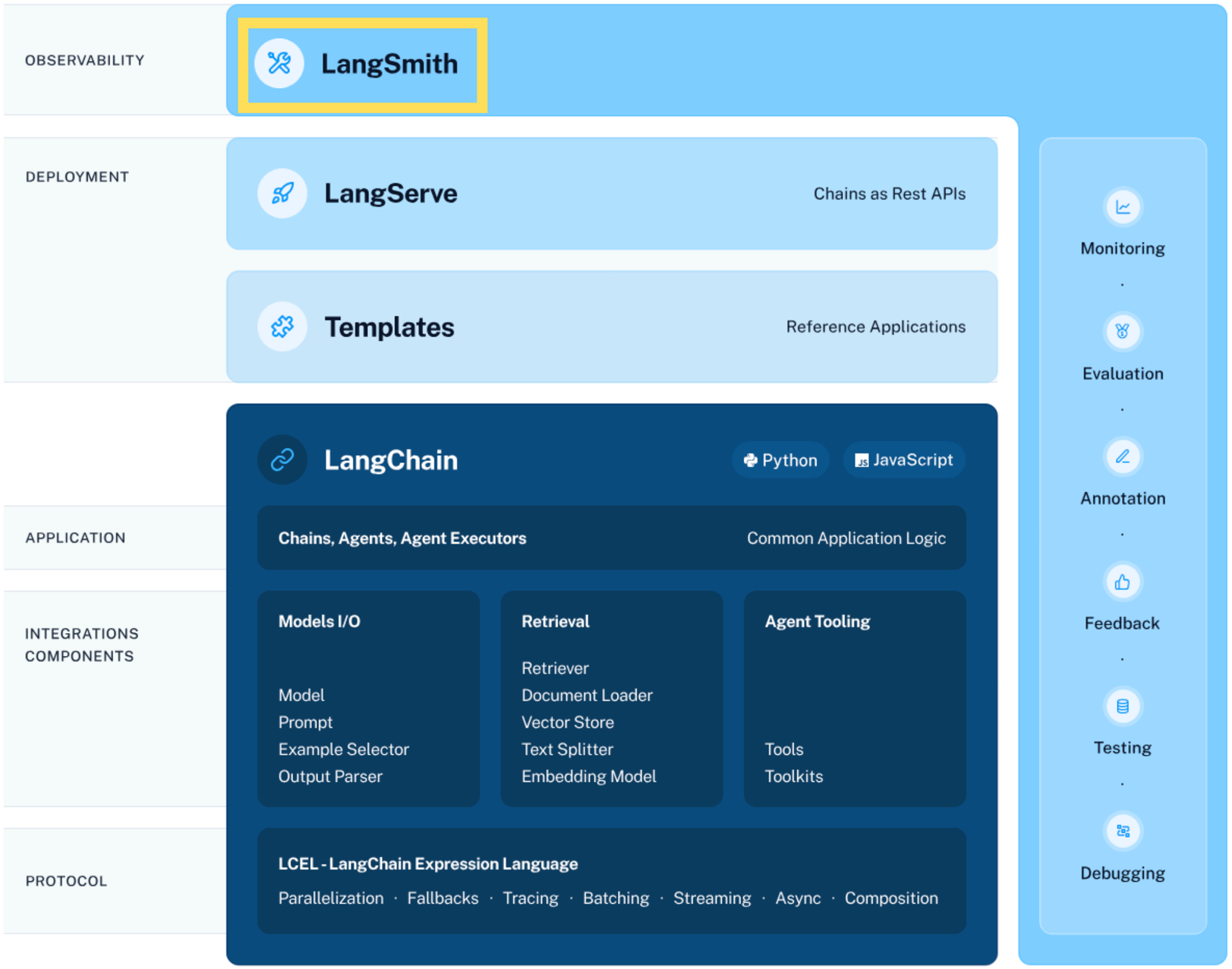
IMPROVING RAG

Presented by

Chris Alexiuk, LLM Wizard 🪄

CONCLUSIONS

- **LangServe** is solving end-to-end LLM application problems!
  - **Streamlines production improvements**, from Prompt Engineering to RAG to Fine-Tuning
- **Baseline evaluation** is key
  - LLMs as evaluators is an emerging space!
- From there, **improving** your system is the data science of the future!


OBSERVABILITY
LangSmith
DEPLOYMENT
LangServe
Chains as Rest APIs
Templates
Reference Applications
APPLICATION
LangChain
Python
JavaScript
Chains, Agents, Agent Executors
Common Application Logic
INTEGRATIONS
COMPONENTS
Models I/O
Model
Prompt
Example Selector
Output Parser
Retrieval
Retriever
Document Loader
Vector Store
Text Splitter
Embedding Model
Agent Tooling
Tools
Toolkits
PROTOCOL
LCEL - LangChain Expression Language
Parallelization · Fallbacks · Tracing · Batching · Streaming · Async · Composition
Monitoring
Evaluation
Annotation
Feedback
Testing
Debugging

QUESTIONS?

Thank you!