Prompt engineering

The Udemy experience

- Playback speed
- Course resources – Code, diffs, Q&A
- Reviews

Course structure:

- Section 1 – Tools setup
- Section 2 - Amazon Bedrock introduction and first steps
- Section 3 – Text models
- Section 4 – Image models
- Section 5 – Embeddings and Vector databases
- Sections 7 – 9 projects (RAG, text API, image API)
- Section 10 – Bedrock Knowledge bases
- Section 11 – Custom models
- Final sections: Recap

Tools we will need:

- AWS account – personal of from work (preferred with administrator access)
- Coding parts:
  - Text editor (VSCode)
  - Python
  - NodeJS

Amazon Bedrock intro

- First contact with Bedrock models
- Familiarize with the console view
- Set up for SDK access (PY and TS)

Amazon Bedrock

Interface to AI models hosted on AWS. The access can be provisioned (expensive) or serverless (on demand)

Bedrock model providers:

- Amazon
- AI21 Labs
- Anthropic
- Cohere
- Meta
- Mistral AI
- Stability AI

Custom models – fine tune an existing model – expensive task

What is an AI model?

AI models are programs that detect specific patterns using a collection of data sets. Once trained, an AI model can be used to make future predictions or act on data that was not previously observed

- Text generation based on a prompt (text to text)
- Image generation based on a prompt (text to image)
- Text tasks – summary, translation, spelling check, style change
- Anomaly detection
- Recommender systems
- Speech
- Video

Prompt engineering helps define use cases:

- Classification
- Question-answer, without context
- Question-answer, with context
- Summarization
- Code generation
- Reasoning or logical thinking

Anatomy of a prompt:

This is a phone conversion between Alex and Emily:

Context info

Alex: Hey, Emily! How's it going?

Emily: Alex! I'm doing well, thanks. How about you?

Alex: Can't complain, just enjoying the weekend vibes. Speaking of which, do you have any plans for this weekend?

Emily: Not yet, I was actually hoping we could plan something fun. Any ideas?

Alex: I was thinking the same thing! How about a picnic at Riverside Park? We could bring some snacks, play frisbee, and just relax.

Emily: That sounds fantastic! I love the idea. Should we invite Jordan and   Casey as well?

Alex: Absolutely, the more, the merrier. I'll text them and see if they're free.

Reference text

From the call transcript above, crate a summary of the conversation in maximum 30 words.

Simple instructions

The form of the output is described

*Add **history** to chatbot*:

- Start user input with “User: ”
- Start assistant messages with “Bot: ”
- Add all messages to a context List (Array)
- Use the context as prompt

- Use titan model

Retrieval Augmented Generation

Retrieval-Augmented Generation (RAG) is the process of optimizing the output of a large language model, so it references an authoritative knowledge base **outside of its training** data sources before generating a response

RAG extends the already powerful capabilities of LLMs to specific domains or an organization's internal knowledge base, all without the need to retrain the model

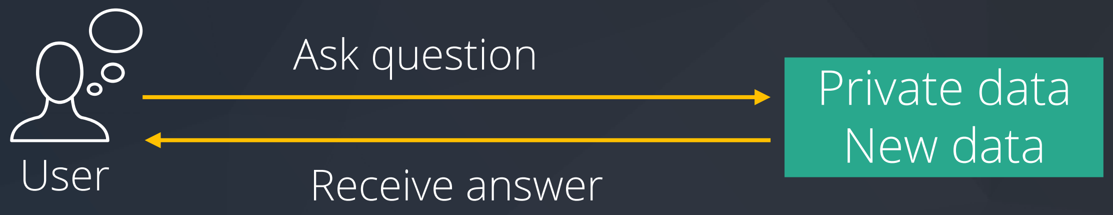
Prompt engineering refers to the practice of optimizing textual input to LLMs to obtain desired responses.

Prompts are a specific set of inputs provided by you, the user, that guide LLMs on Amazon Bedrock to generate an appropriate response or output for a given task or instruction.

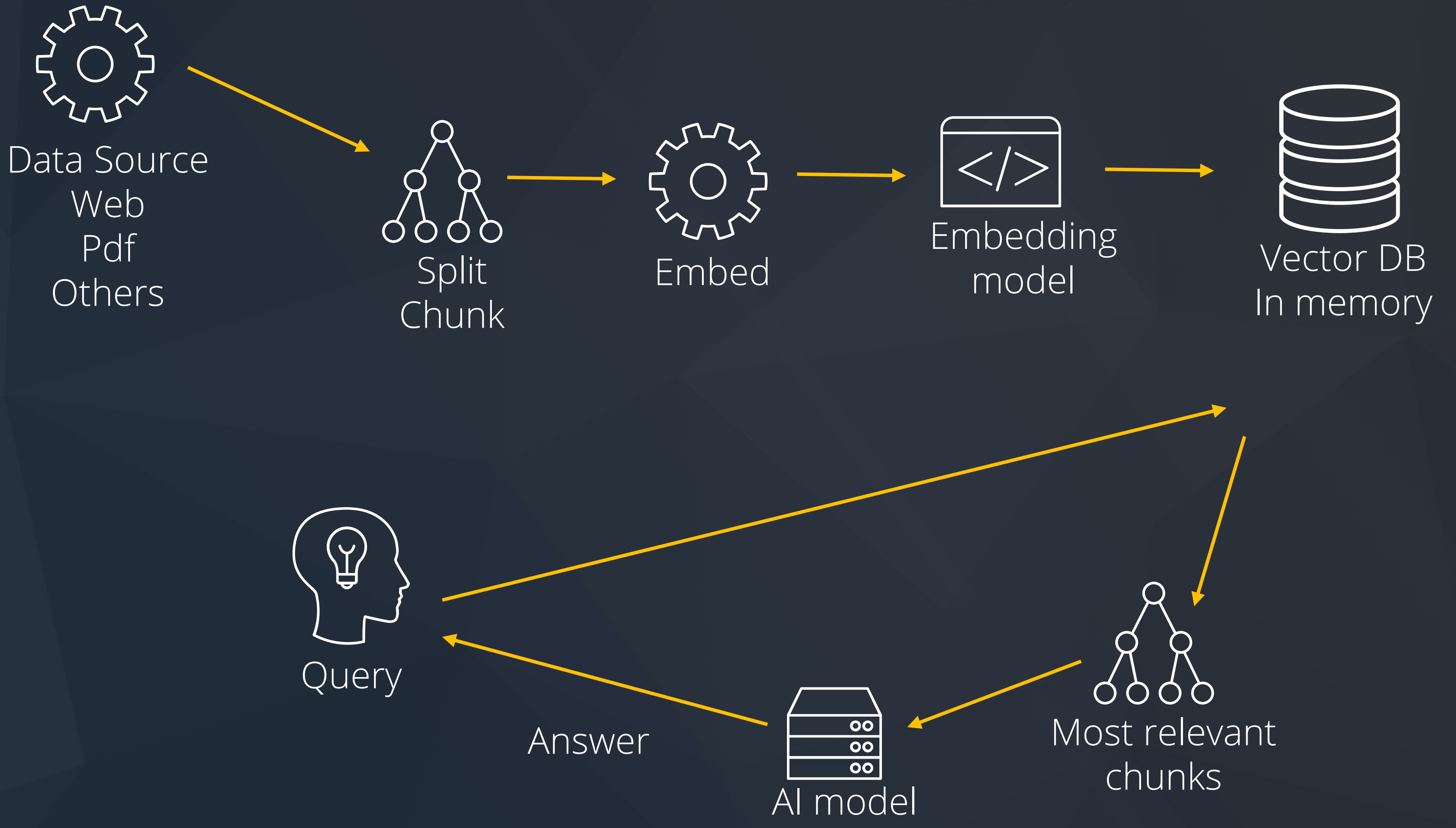
Example:

What is the highest mountain on earth?

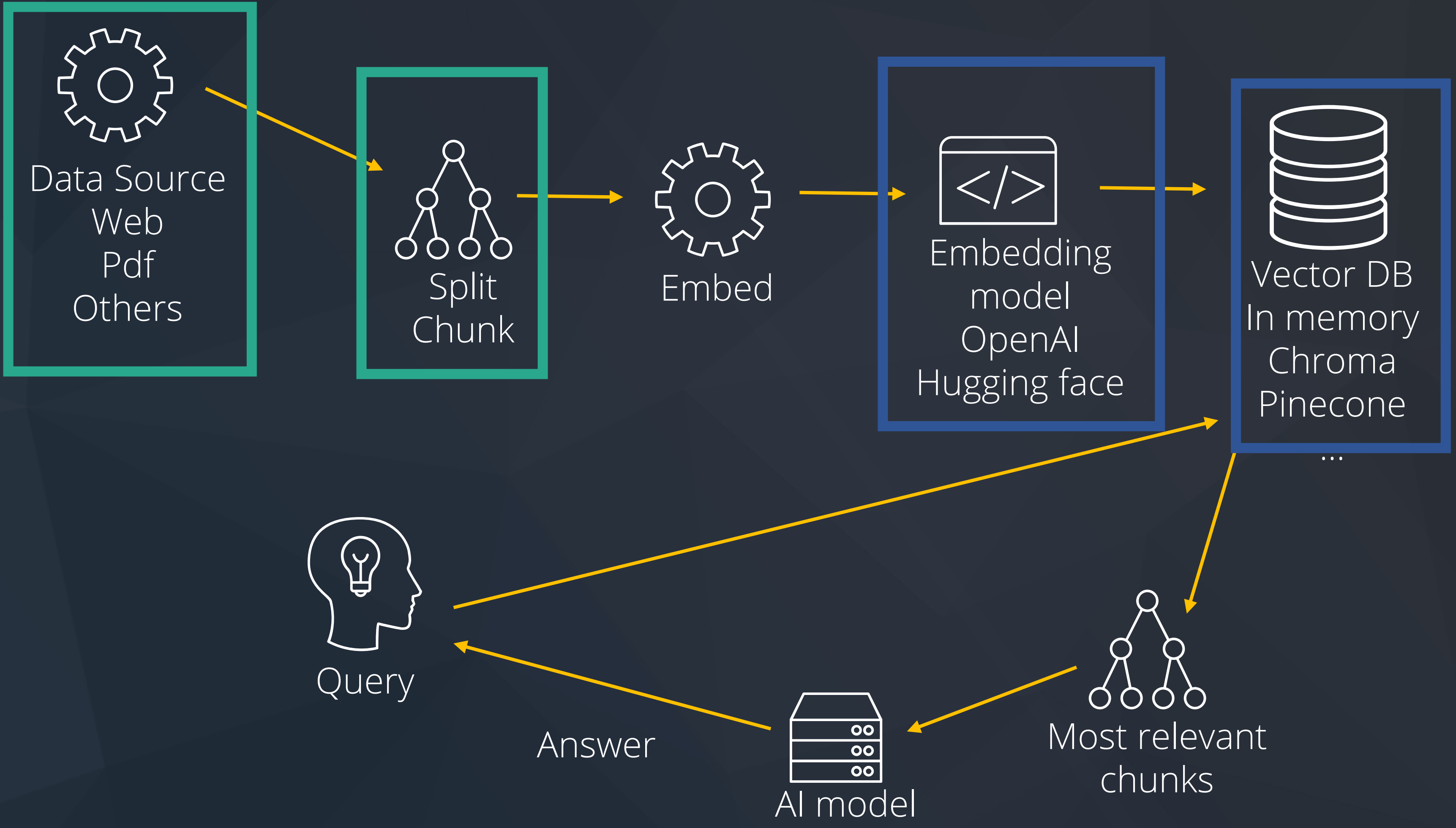
Retrieval Augmented Generation



Retrieval Augmented Generation



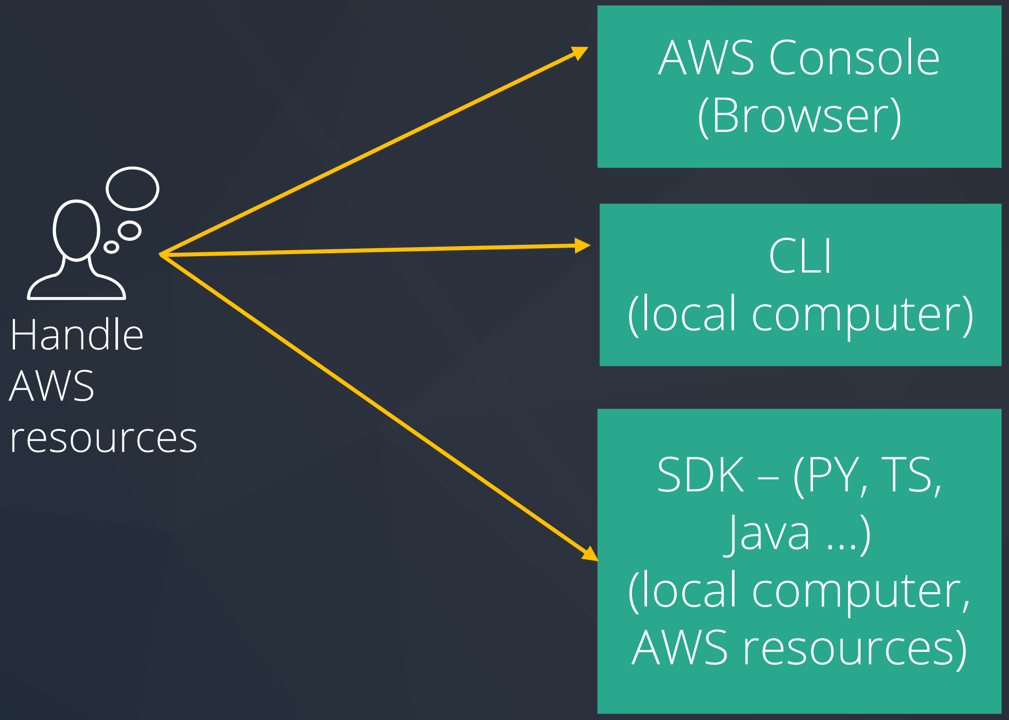
Retrieval Augmented Generation



Bedrock text models

- Text models intro
- Model parameters
- Text generation using the SDK – PY and TS

IAM user for CLI – SDK access



IAM user for CLI

- Create IAM user for console access
- Put credentials in aws credentials file
  - Optional (recommended): with the AWS CLI

Text models parameters

- Dependent on the model!
  - Check providers and model docs for examples
- Ex top P – nucleus sampling
  - topP – Titan text models
  - top_P – Anthropic Claude/Llama models

Text models parameters

- Max tokens (Titan) / Response length (Llama)
- Temperature
- Stop sequence
- Seed

Text models parameters

- Temperature – controls the randomness or creativity of the model:
  - 0 – more deterministic results
  - 1 – more creative results
    - High temperature – bigger risk of nonsensical/irrelevant content – halucinations

Text models parameters

- Top p – nucleus sampling – controls the randomness of the output – the choice of tokes
  - Example: topP set to 0.9 => model will only consider the most likely words that make up 90% of the probability mass.

Text models parameters

- Stop sequences (Titan) – the response is ended if the sequence is encountered.
  - Useful with structured outputs – greetings, structured text generation (code)

Bedrock image models

- Image models intro
- Model parameters
- Image generation using the SDK – PY and TS

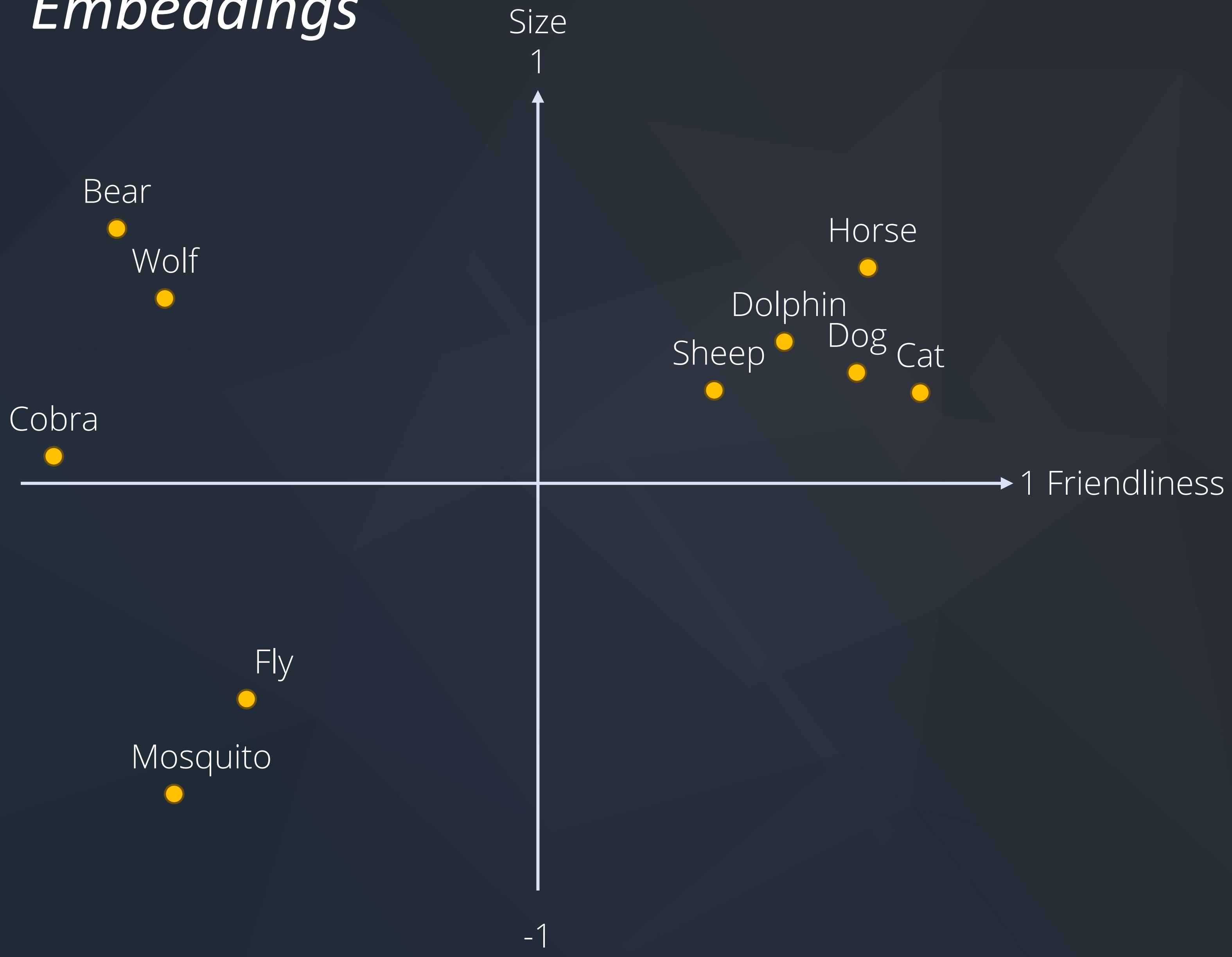
- Only guidelines – room for experimentation

Embeddings – [0.2, 0.123, 0, -0.4, 1]

- The key to AI
- Numerical representation of data: text, images, sound
- Text embedding – numerical representation of text
- Take the form of numbers array (vector)
- Embedding (latent) space – a space in which similar items are positioned closer to one another than less similar items

- [0.2, 0.123, 0, -0.4, 1]

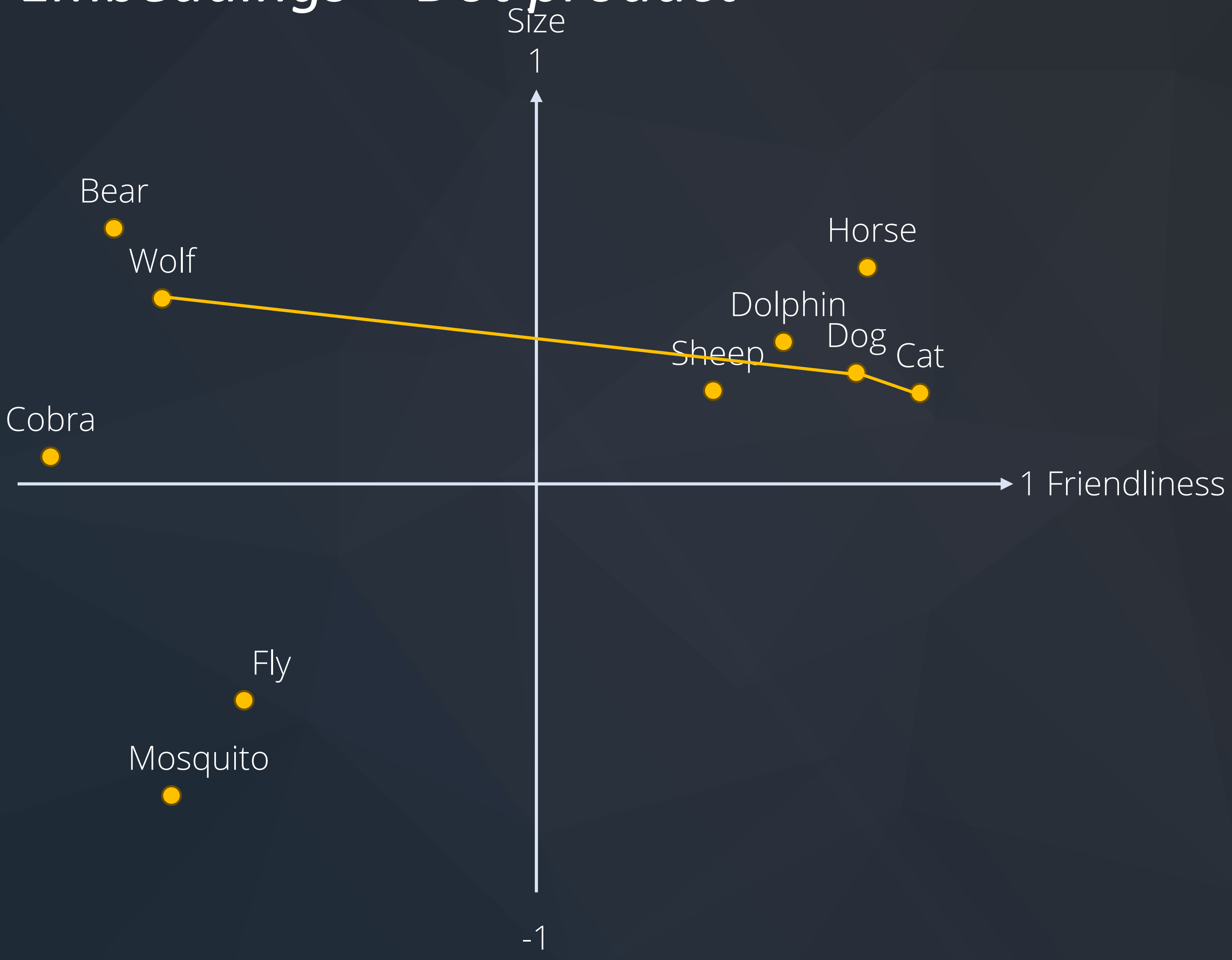
Embeddings



Cat [0.8, 0.1]
- Dog [0.75, 0.15]

…
- Fly [-0.6, -0.5]

Embeddings – Dot product



Cat [0.8, 0.1]

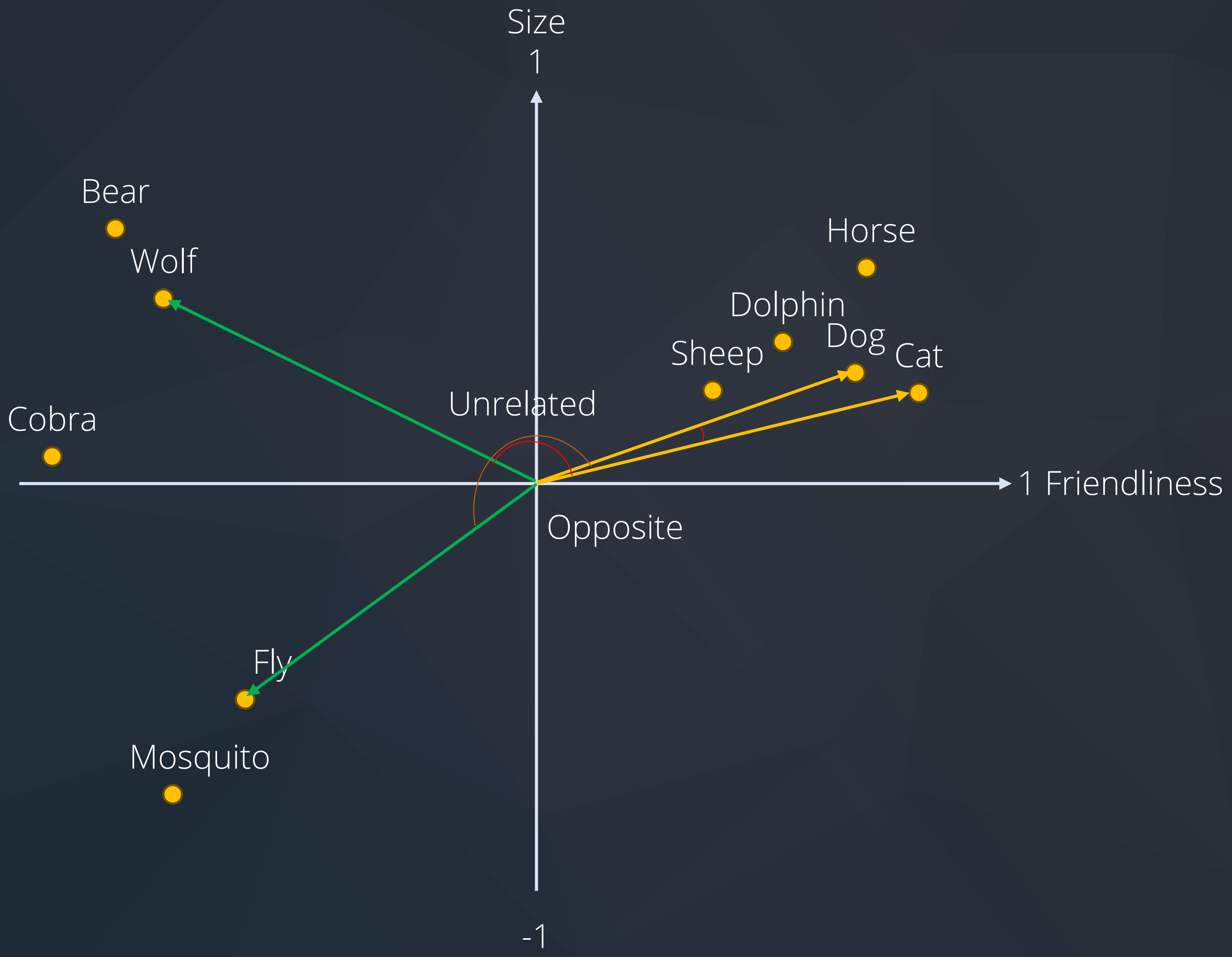
X

Dog [0.75, 0.15]

=

0.8*0.75 + 0.1*0.15

Embeddings – Dot product



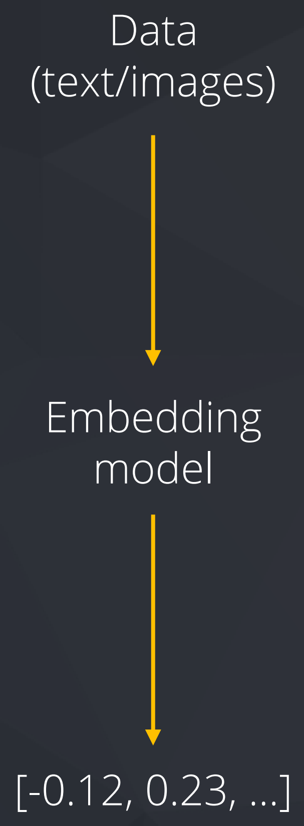
Cat Dog -> related

Cat Wolf – unrelated

Cat Fly - opposite

Embedding models

- Def: set of algorithms trained to generate embeddings
- Bedrock
  - amazon.titan-embed-text-v1
  - amazon.titan-embed-image-v1
- Why we need embeddings: similarity search
  - Recommendation systems
  - Fraud detection
  - Basic chat functionality
  - Classification systems



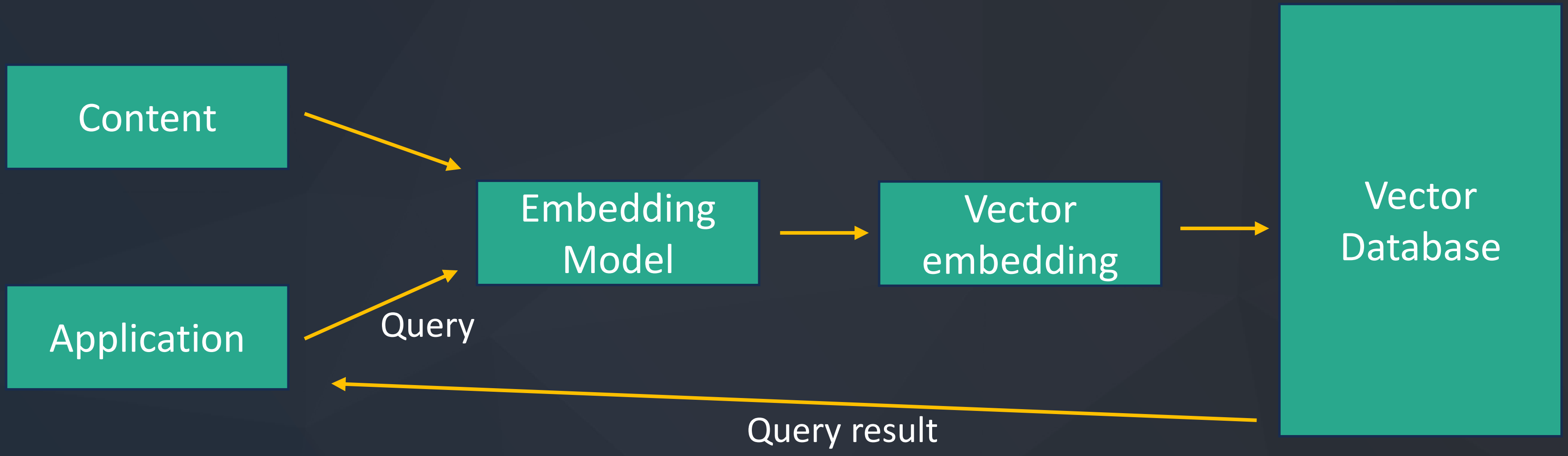
AI application challenge

- Efficient data processing
- Vector databases solve this problem
- Popular vector databases:
  - Pinecone
  - Chroma
  - Redis
  - PostgreSQL– PG vector extension
  - Amazon OpenSearch Service
  - Amazon Aurora PostgreSQL
  - Amazon Document DB (MongoDB compatibility)

Vector databases

- Specialized databases for storing vectors
- Special feature vs traditional databases: the way we query:
  - SQL: query based on values match
  - Vector: query based on similarity
- Vector db:
  - Holds data + embeddings
  - Uses an embedding as query parameter
- Vector databases use a combination of different optimized algorithms – Approximate Nearest Neighbor search

Vector databases

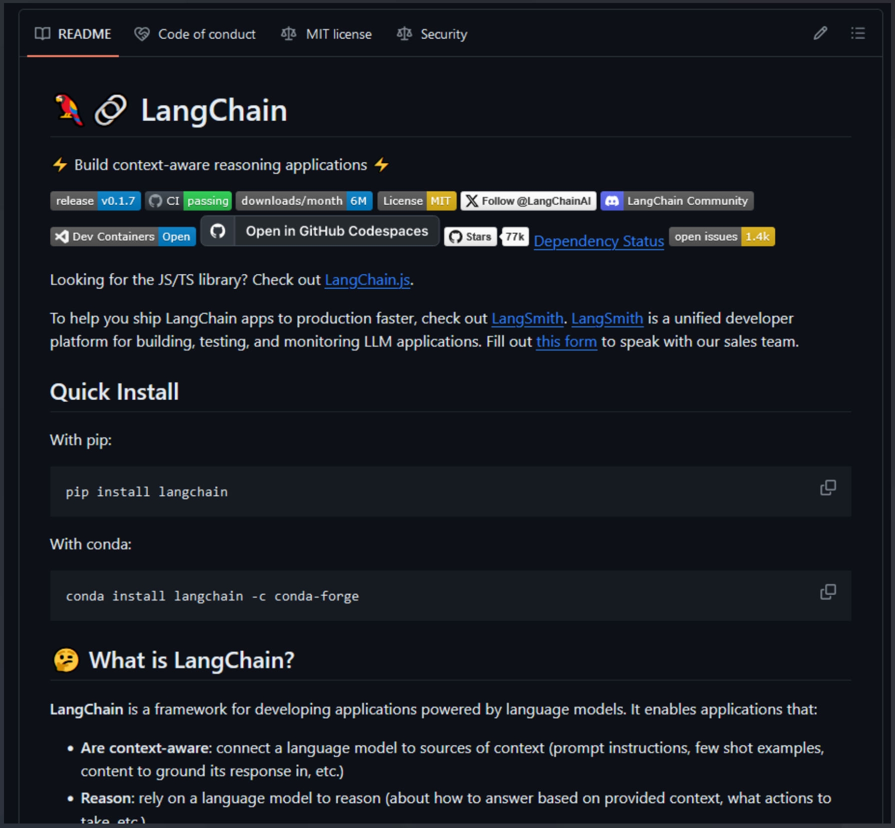


LangChain and RAG apps

- Local RAG app – chat with your data app
- Integrate Bedrock with LangChain

LangChain

- LangChain is a framework for developing applications powered by language models
- Integrate it with external data sources and vector DBs (Chroma DB)
- Implement abstractions:
  - Chains – chain commands
  - Agents – use an LLM to make decisions



🦜🔗 LangChain

⚡ Build context-aware reasoning applications ⚡

Looking for the JS/TS library? Check out LangChain.js.

To help you ship LangChain apps to production faster, check out LangSmith. LangSmith is a unified developer platform for building, testing, and monitoring LLM applications. Fill out this form to speak with our sales team.

Quick Install

With pip:

```
pip install langchain
```

With conda:

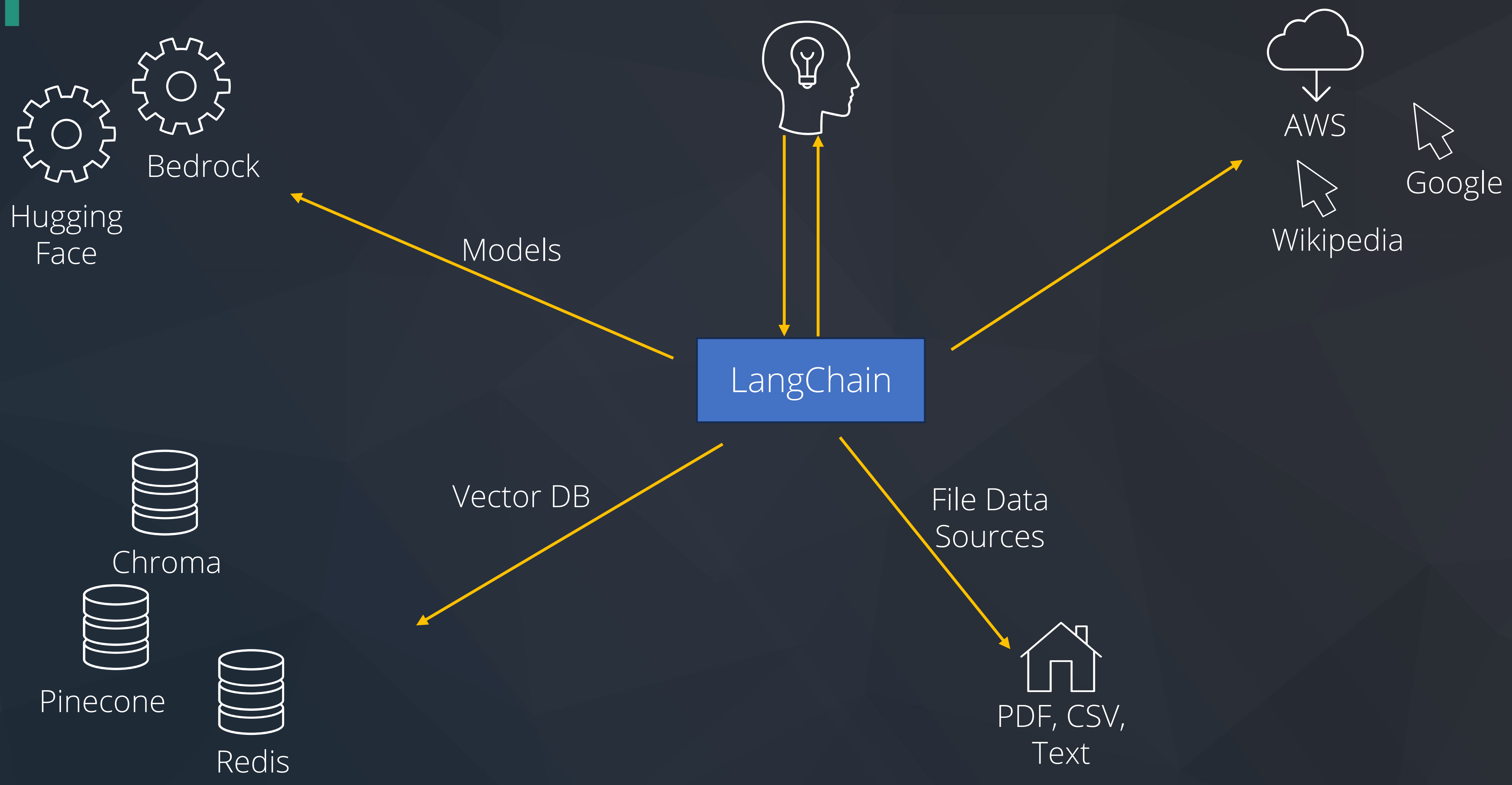
```
conda install langchain -c conda-forge
```

🤔 What is LangChain?

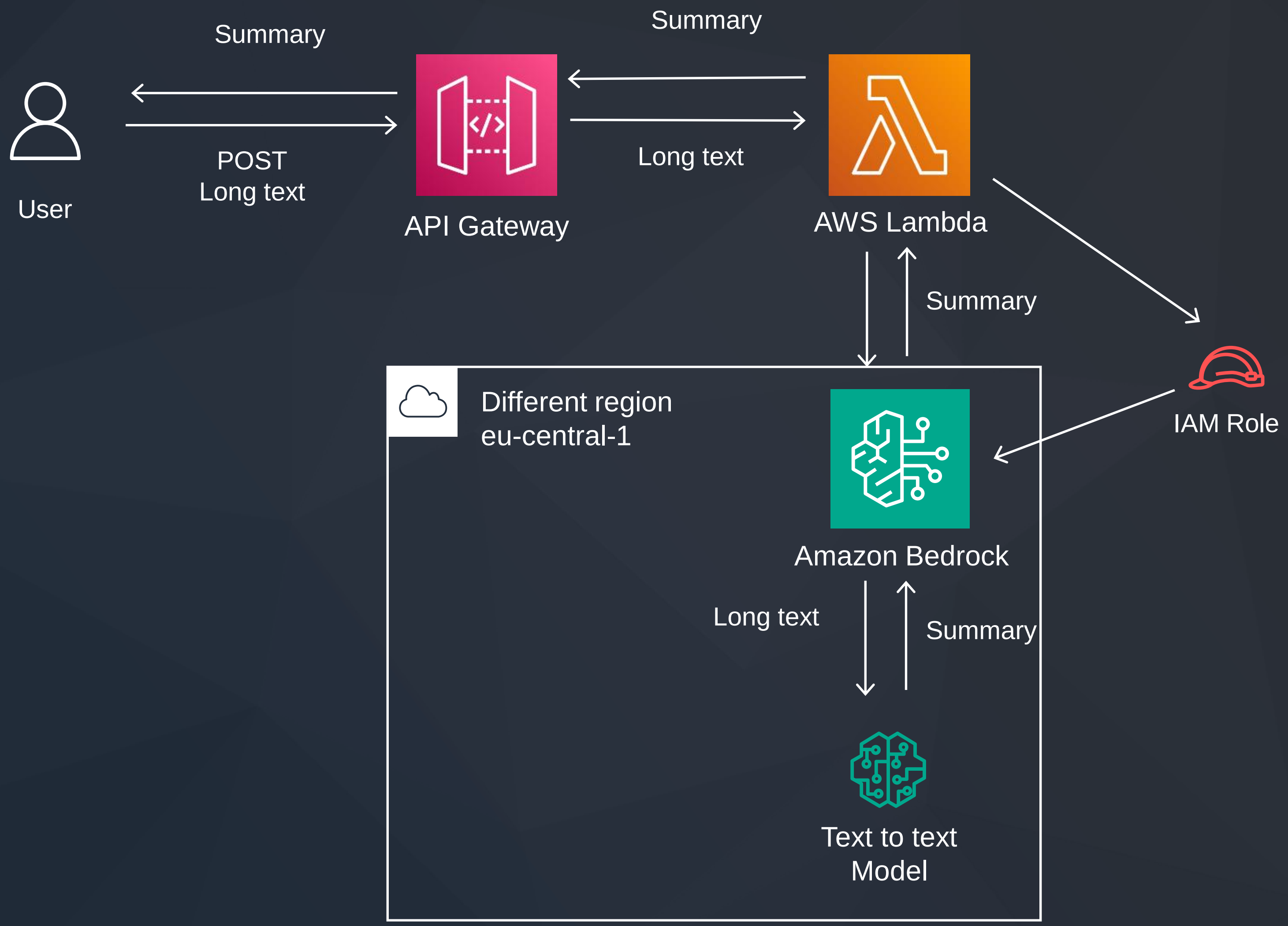
LangChain is a framework for developing applications powered by language models. It enables applications that:

- **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.)

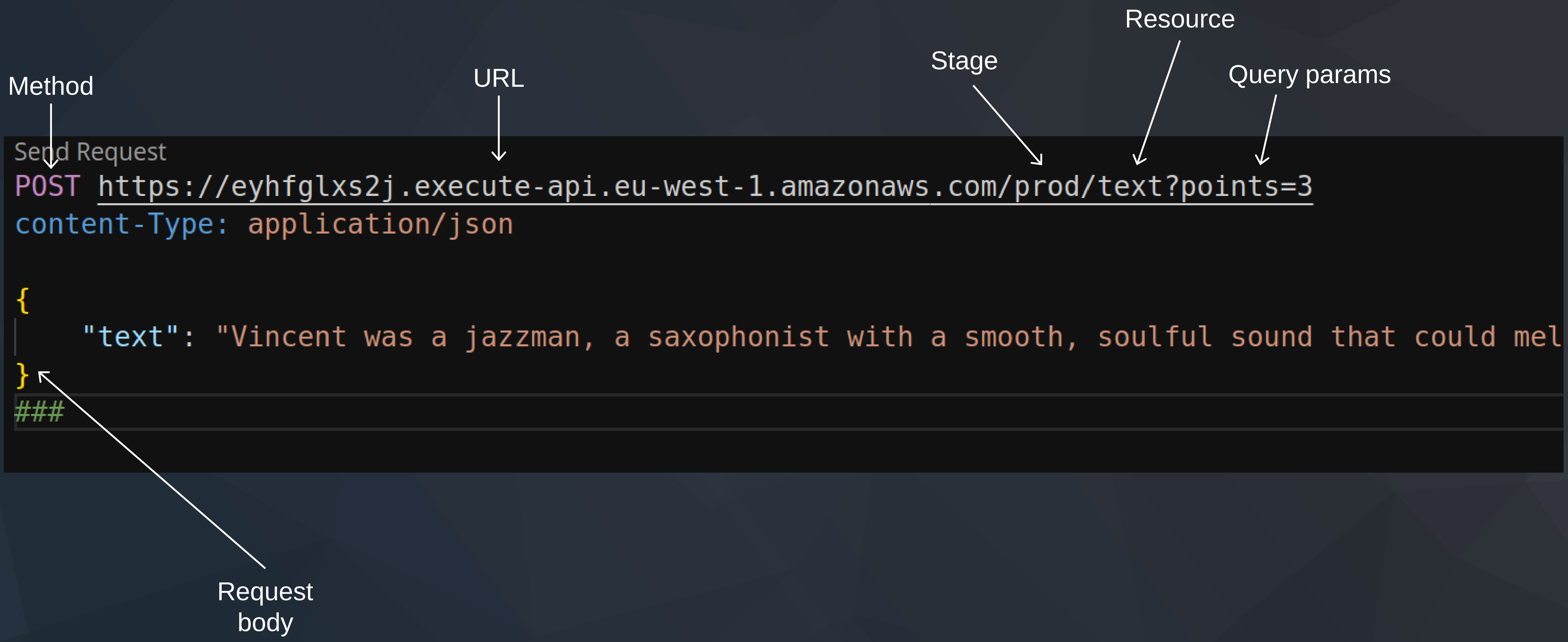
LangChain


Bedrock
Hugging Face
Models
AWS
Google
Wikipedia
LangChain
Vector DB
File Data Sources
Chroma
Pinecone
Redis
PDF, CSV, Text

Bedrock summary API



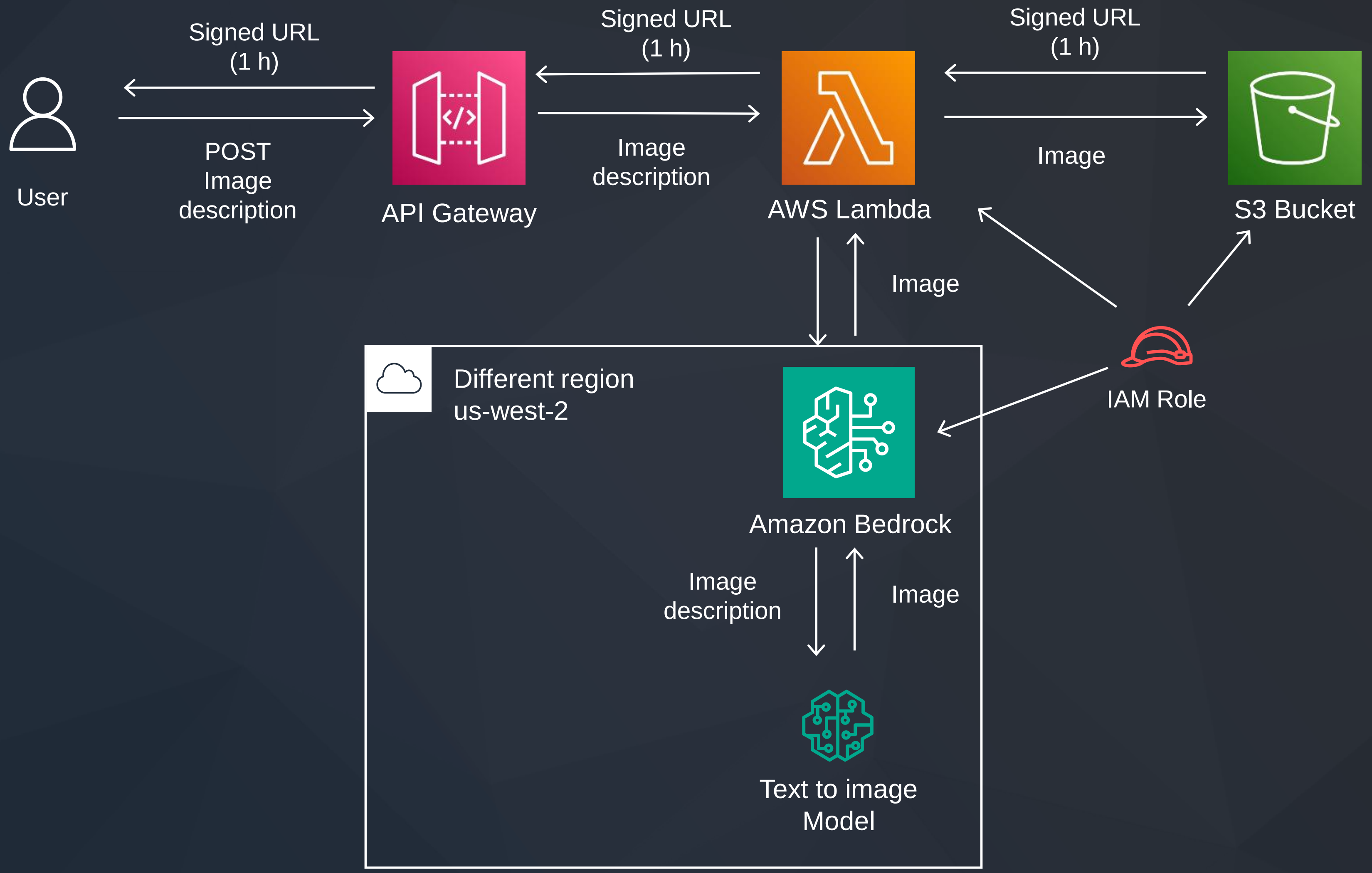
Api Gateway request



```
POST https://eyhfglxs2j.execute-api.eu-west-1.amazonaws.com/prod/text?points=3
content-Type: application/json

{
    "text": "Vincent was a jazzman, a saxophonist with a smooth, soulful sound that could mel
}
###
```

Bedrock image API



RAG applications:

- Extract data from legal documents
- Organize internal documentation
- Text generation

Bedrock Knowledge bases

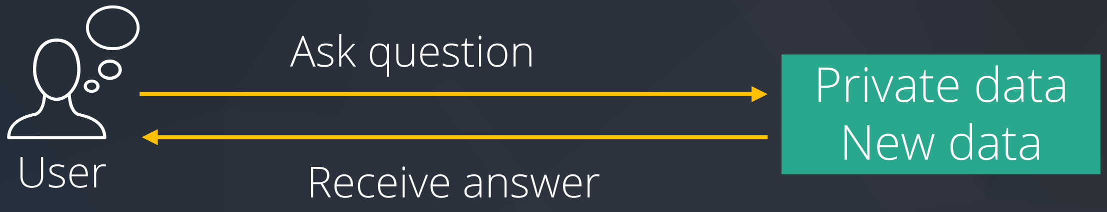
An interface to RAG applications

Retrieval Augmented Generation

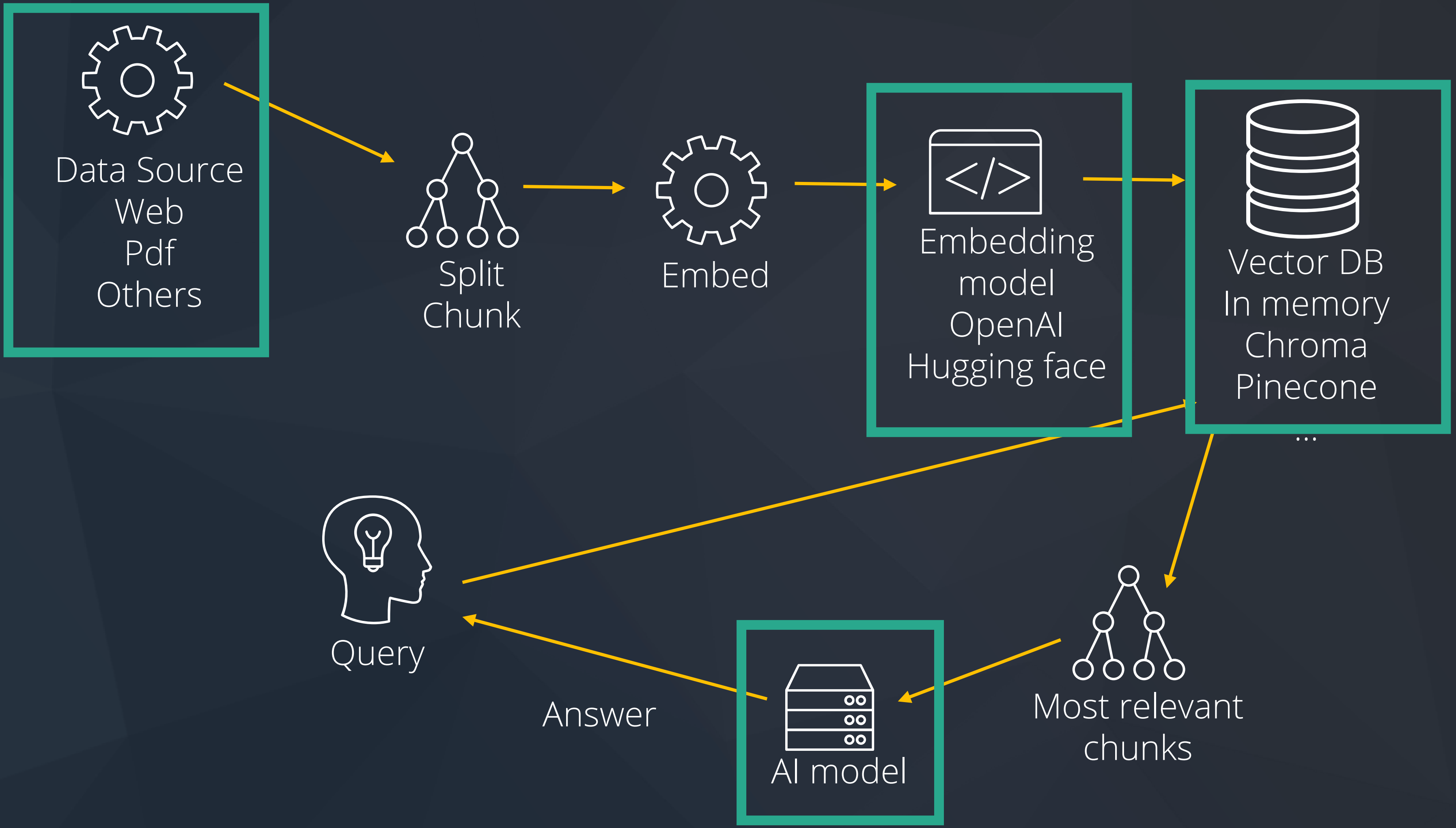
Retrieval-Augmented Generation (RAG) is the process of optimizing the output of a large language model, so it references an authoritative knowledge base **outside of its training** data sources before generating a response

RAG extends the already powerful capabilities of LLMs to specific domains or an organization's internal knowledge base, all without the need to retrain the model

Retrieval Augmented Generation

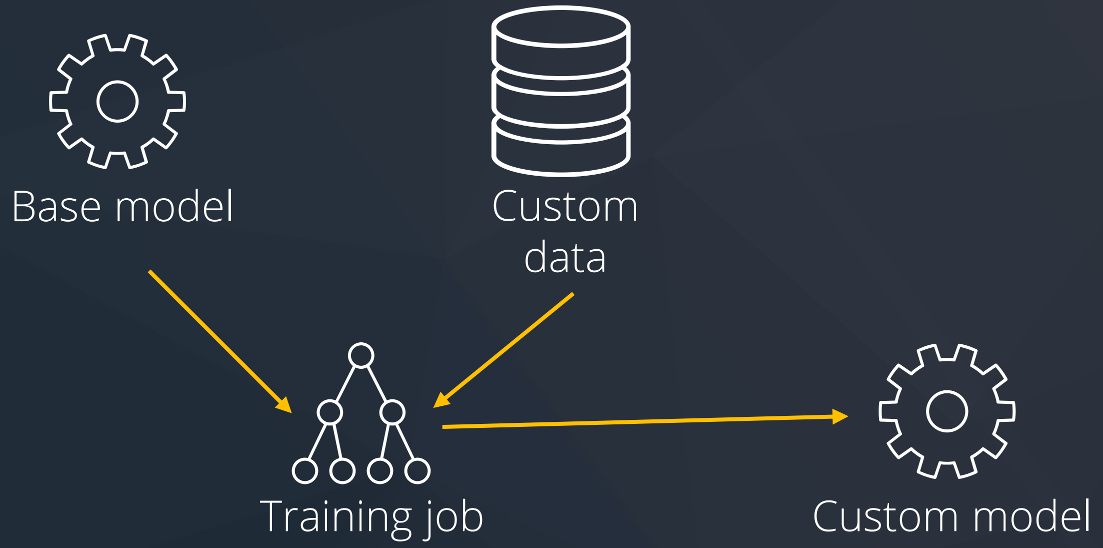


Retrieval Augmented Generation



Fine tuning of a base model -> custom model

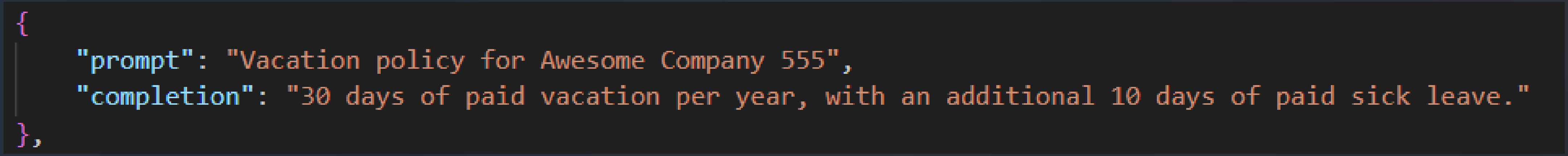
Fine-tuning trains a pretrained model on a new dataset without training from scratch



Example – Titan Text G1 - Lite

Question: What is the vacation policy for Awesome 555 Cool Company?

Model response: For up-to-date information on Awesome 555 Cool Company's vacation policy, please refer to their official HR website or ask your manager.



```
{
    "prompt": "Vacation policy for Awesome Company 555",
    "completion": "30 days of paid vacation per year, with an additional 10 days of paid sick leave."
},
```

Base model vs custom model

Base model – pay on request

Custom model – training job + provisioned Throughput

-> feasible for complex use cases

-> no worries about data privacy

-> can be used as any model – by model id

AWS Recap

- IAM
- Lambda
- Api Gateway

Infrastructure as Code

- CloudFormation
- CDK – CloudFormation super set
  - Write your infrastructure using your preferred programming language
- Terraform - incoming

Halfway discussion

- Thank you!
- What is next: projects with AWS/IAC – consider recap sections
- Use provided code and code diffs
- Please review ;)