



GENERATIVE AI IMMERSION DAY

# Implementing Generative AI in Organizations

Challenges and Opportunities

Aris Tsakpinis

AI/ML Specialist Solutions Architect

# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

Large Language Model Hosting

Large Language Model Finetuning

Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

Large Language Model Hosting

Large Language Model Finetuning

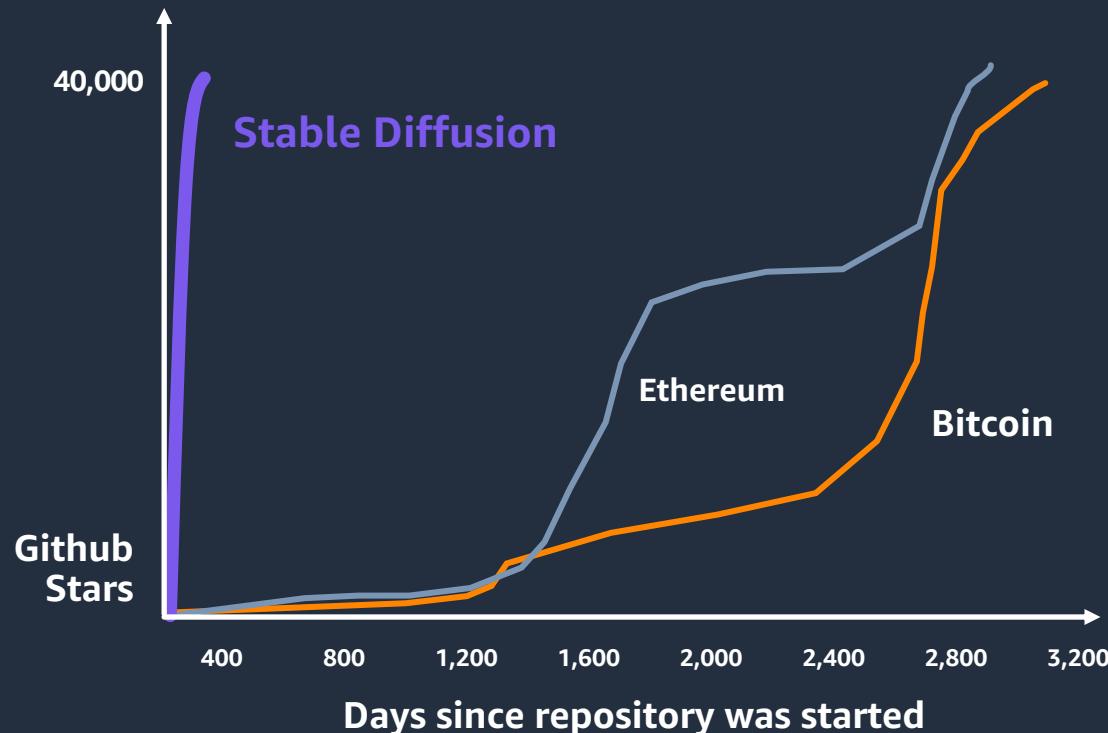
Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# Generative AI is the fastest growing trend in AI

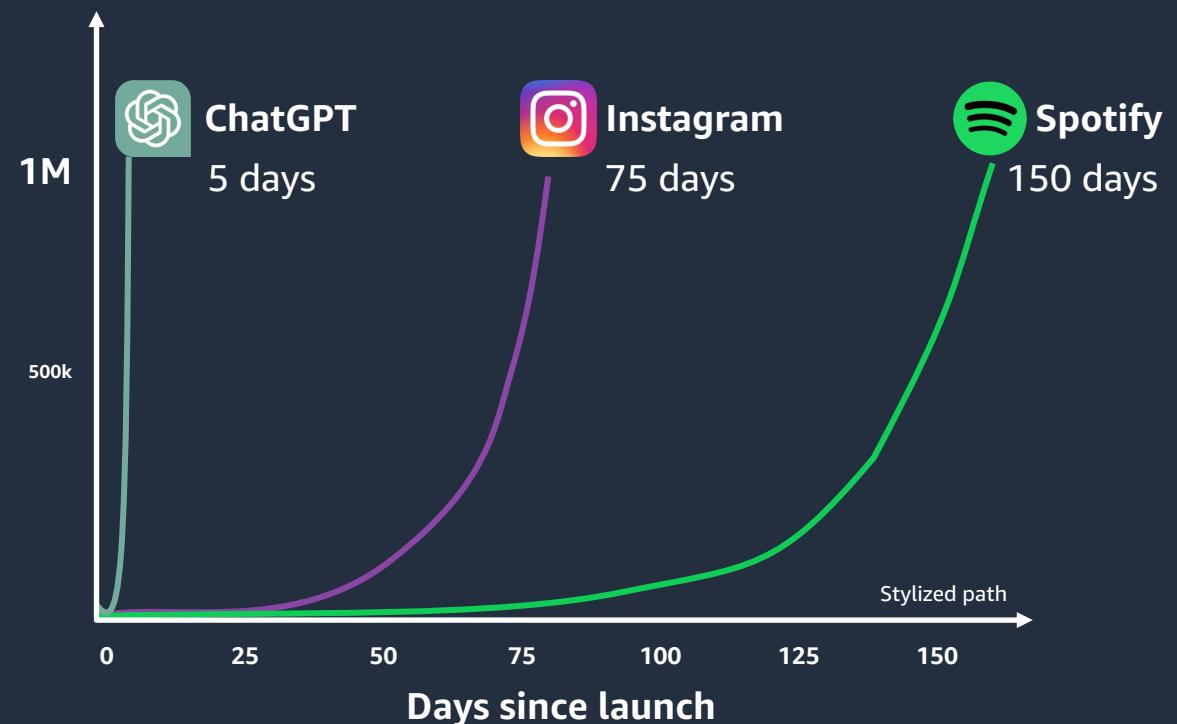
Developer adoption

**Stable Diffusion accumulated 40k stars on GitHub in its first 90 days**



Consumer adoption

**ChatGPT reached the 1 million users mark in just 5 days**



# What is Generative AI?



AI that can  
**generate content**  
close enough to human created  
content for real-world tasks



Powered by  
**foundation models**  
pre-trained on large sets of data with  
several hundred billion parameters



Applicable to  
**many use cases**  
like text summarization, question  
answering, digital art creation,  
code generation, etc.



Tasks can be  
**customized for  
specific domains**  
with minimal fine-tuning



New Volvo car concept design by midjourney  
Credit: @sugardesign\_1 Instagram

# Generative AI – what's the perk?

Humans

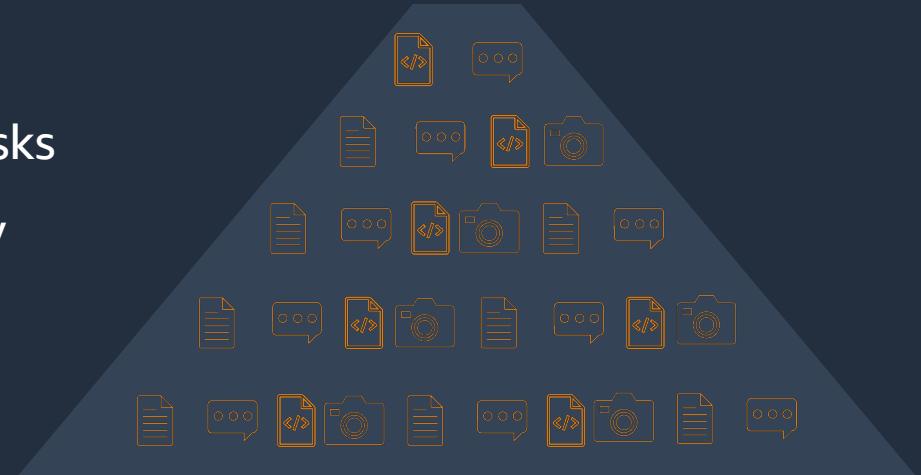


Speed & Breadth

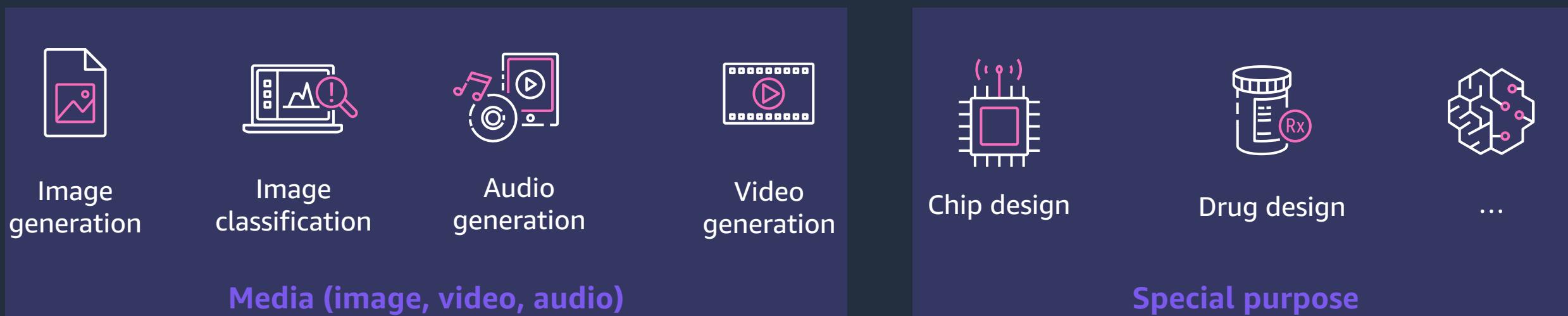
- Generates novel content
- No task specific models
- Applicable to a broad set of tasks
- Outputs easily interpretable by humans



Generative AI



# Gen AI use case emerge in 4 different categories



# ... but there are also certain limitation to bear in mind

We asked ChatGPT about the limitations of LLMs ...

A Can you provide me a list of limitations large language models come with?

G Yes, here are some limitations of large language models:

1. Biases: Large language models can perpetuate existing biases in the data they are trained on, which can lead to biased or discriminatory outputs.
2. Contextual understanding: Large language models are not able to fully understand context, which can lead to misunderstandings or misinterpretations of text.
3. Lack of common sense: Large language models do not have a comprehensive understanding of the world, and may not possess common sense knowledge that humans take for granted.
4. Limited domain knowledge: Large language models are only as knowledgeable as the data they are trained on. They may not have the specialized domain knowledge necessary to accurately analyze or generate text in specific fields.
5. Inability to reason: Large language models lack reasoning skills, which can result in inconsistent or illogical responses to certain prompts.
6. Dependence on training data: Large language models require massive amounts of training data in order to perform effectively. This means that the quality of the model's output is dependent on the quality and quantity of the training data.
7. Resource-intensive: Large language models are extremely computationally expensive to train and use, which can limit their accessibility and practical applications in certain contexts.
8. Difficulty with uncommon or specialized language: Large language models may struggle to accurately interpret or generate text in uncommon or specialized language, including jargon, technical terms, or dialects.
9. Limited ability to handle new or evolving language: Large language models may not be able to accurately analyze or generate text in languages or dialects that are evolving or changing rapidly.

It's worth noting that there are ongoing efforts to address some of these limitations, and new research is constantly expanding the capabilities of large language models.

... and it came up with pretty good responses ...

- Risk of bias
- Misinterpretation of context
- Lack of common sense
- Inability for (logical) reasoning
- Limited domain-specific knowledge

... but missed crucial aspects for enterprise applications

- Halluzination
- Knowledge cut-off
- Potential need for content moderation
- Missing references to original sources
- No explainability (as of March 2023)
- ...

# One approach to rule them all...?

If you want to ...

... automatically **extract information from documents**, such as e-mails, forms, invoices, ...

... **make information** contained in documents, audio recordings, or videos easily **searchable**

... wants to **convert audio files** into text

... wants to **translate text**

... wants to **build a chat application** (with execution of scripted workflows based on identified user intent)



... you might want to explore



Amazon Textract



Amazon Kendra



Amazon Transcribe



Amazon Translate



Amazon Lex

Machine learning service that automatically extracts text, handwriting, and data from scanned documents.

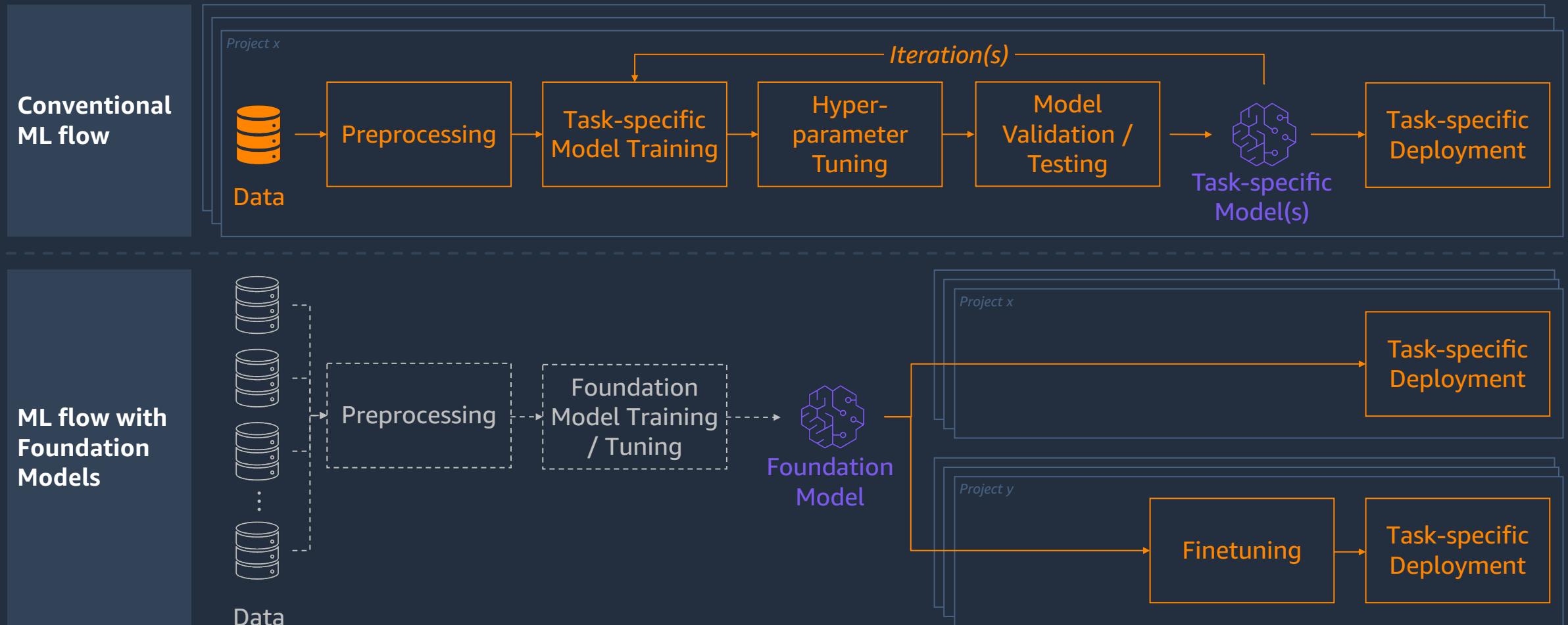
Intelligent enterprise search service that helps you search across different content repositories with built-in connectors.

Automatic speech recognition service that uses machine learning models to convert audio to text.

Neural machine translation service that delivers fast, high-quality, affordable, and customizable language translation.

Fully managed artificial intelligence service with advanced natural language models to design, build, test, and deploy conversational interfaces in applications.

# Foundation models – A paradigm change in the ML flow



# Licensing model has implications on available options, cost, and security

	Proprietary	Open-source
Examples	ChatGPT, GPT-3/4, DALL-E	GPT-J, BLOOM, FLAN-T5, Stable Diffusion
Provisioning model	Model-as-a-Service	Self-hosting <u>or</u> Model-as-a-Service
Access pattern	External API	Internal API embedded in your application landscape
Cost structure	Provider-dependent	Full cost control
Data privacy & residence	Provider-dependent	Under own control (but also own obligation)
How it works	Closed-box	Open-box



# How to get started with Gen AI on AWS?

Option 1: Generative AI as a service

Proprietary

Option 2: SageMaker Jumpstart foundation model hub

Open-source

Option 3: SageMaker & open-source model hubs -  
Example: SageMaker built-in HuggingFace integration

# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

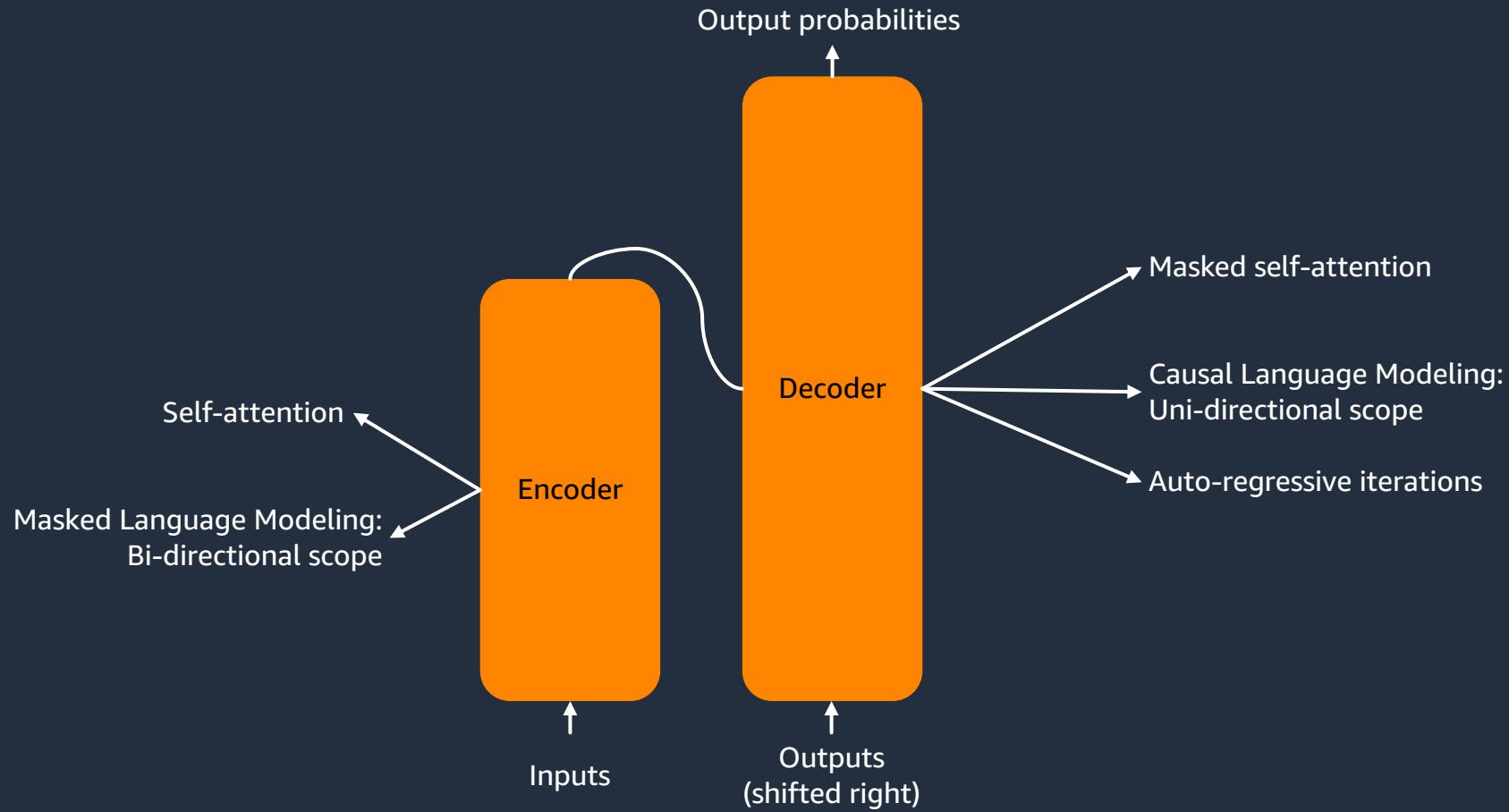
Large Language Model Hosting

Large Language Model Finetuning

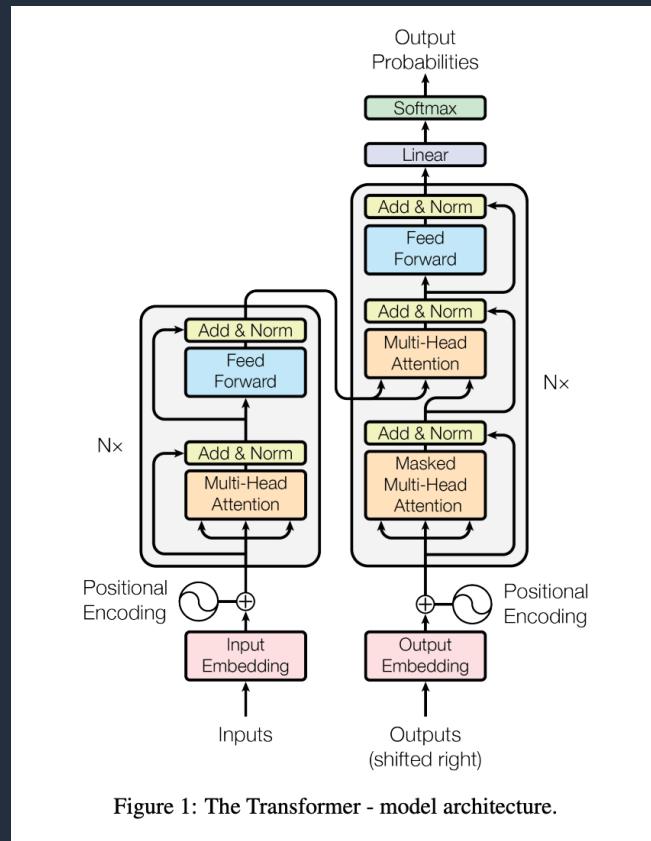
Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# Transformer Models – Core characteristics of sequence-to-sequence transforms



# Transformer Models – Encoders and Decoders form the basis of state-of-the-art LLMs



## Model architecture

### Encoder models

## Common use cases

- Sentence classification
- Named Entity Recognition

## Examples

BERT

### Decoder models

## Common use cases

- Text generation

## Examples

GPT

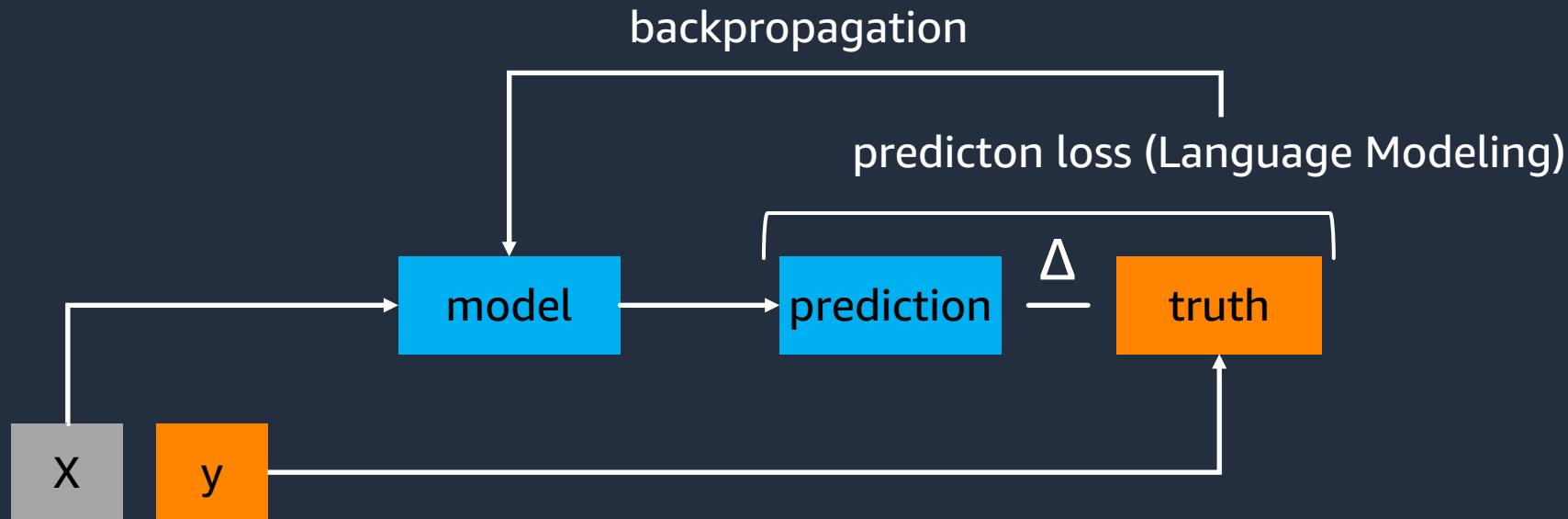
### Encoder-Decoder models

- Summarization
- Translation
- Question answering

## Examples

BART, T5

# How are transformer models trained?



Berlin is the capital of

Switzerland



# Language Modeling Variations

## Masked Language Modeling (MLM)

Berlin is the [REDACTED] of

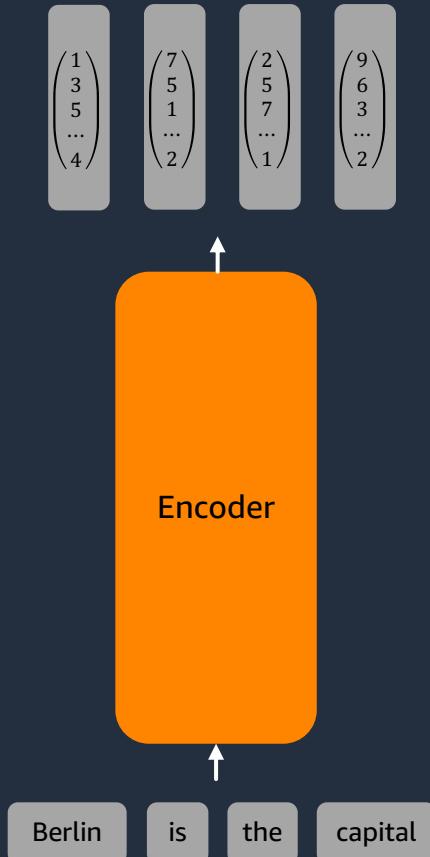
## Causal Language Modeling (CLM)

Berlin is the capital [REDACTED]

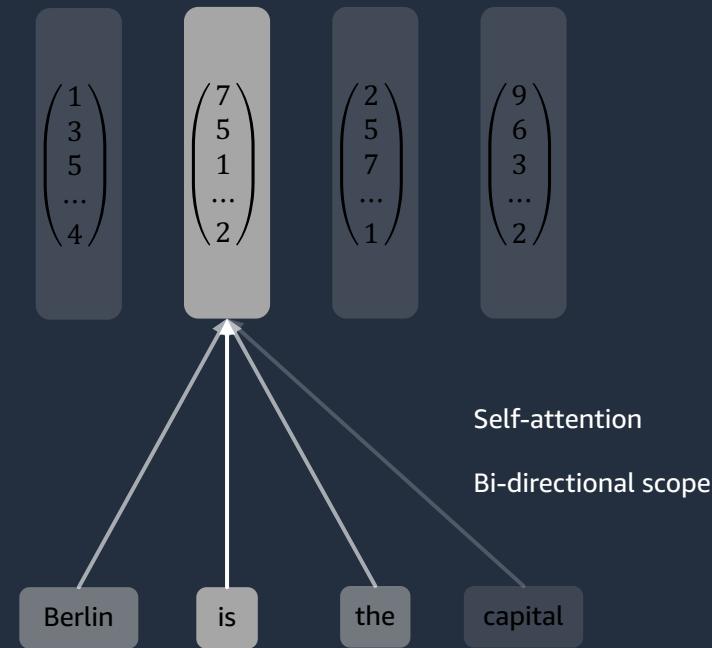
## Permutation Language Modeling (PLM)

of capital Berlin the is  
5 4 1 3 2

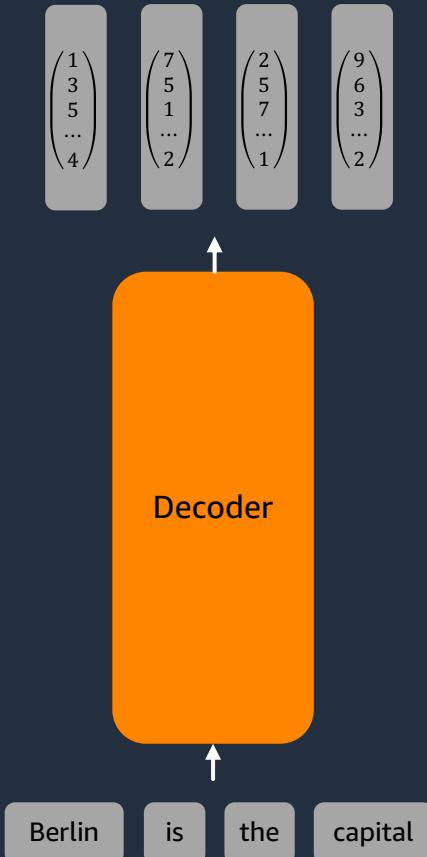
# Transformer Models – How Encoders work



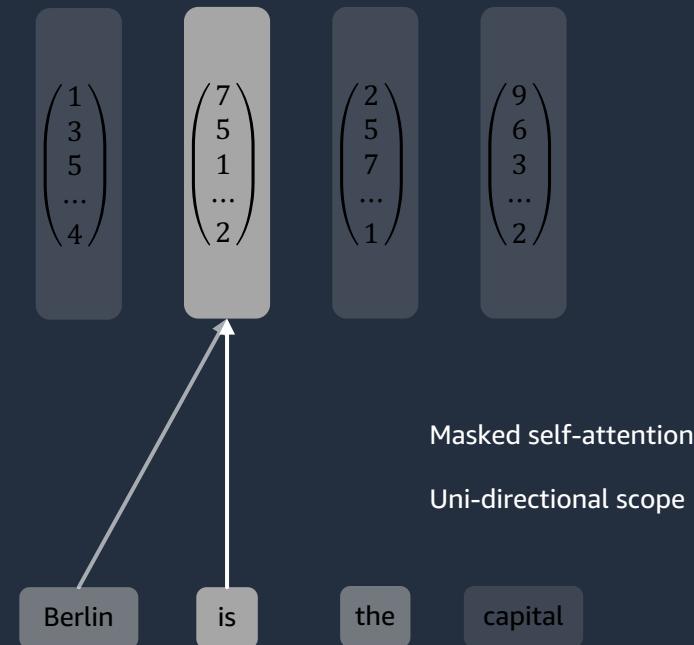
## Masked Language Modeling (MLM)



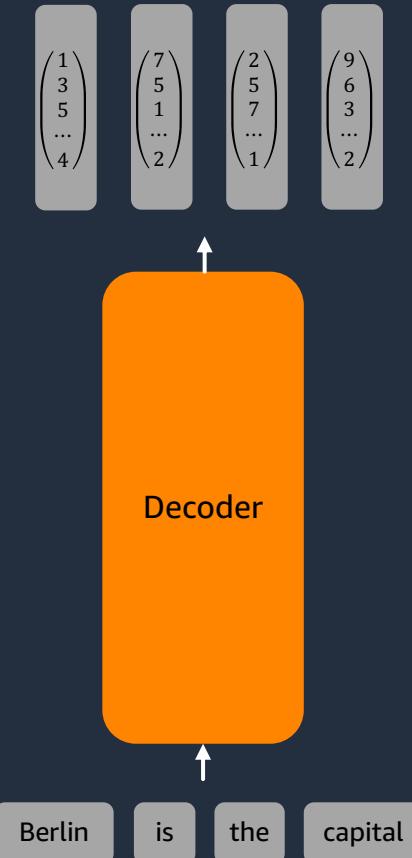
# Transformer Models – How Decoders work



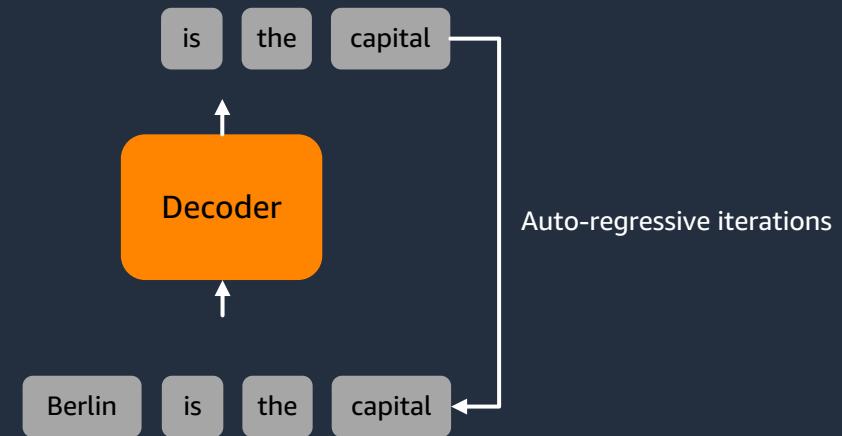
## Causal Language Modeling (CLM)



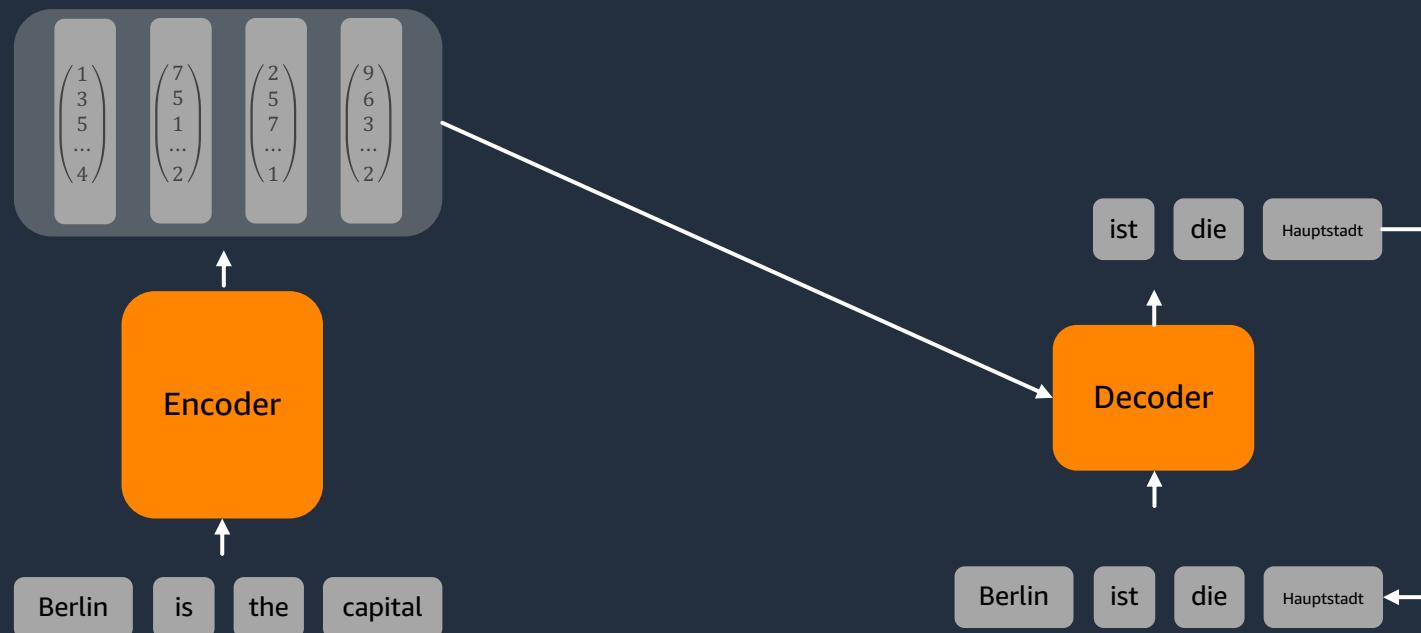
# Transformer Models – How Decoders work



## Causal Language Modeling (CLM)

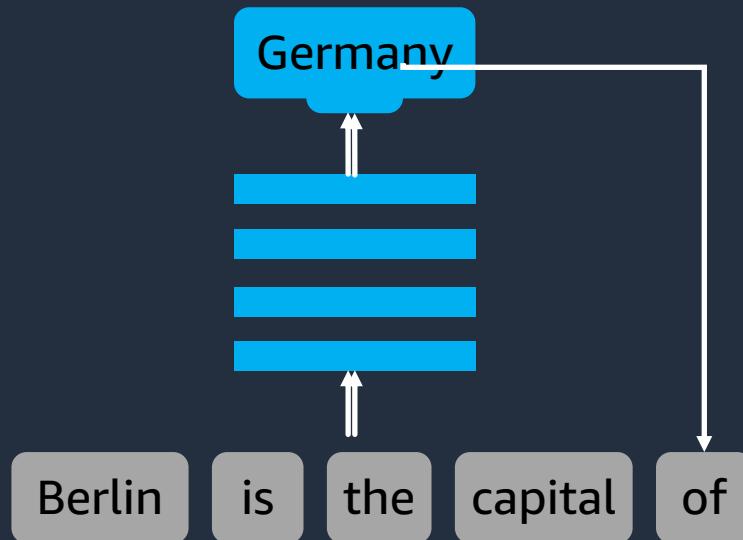


# Transformer Models – How Encoders and Decoders work together



# How do (decoder) LLMs make predictions?

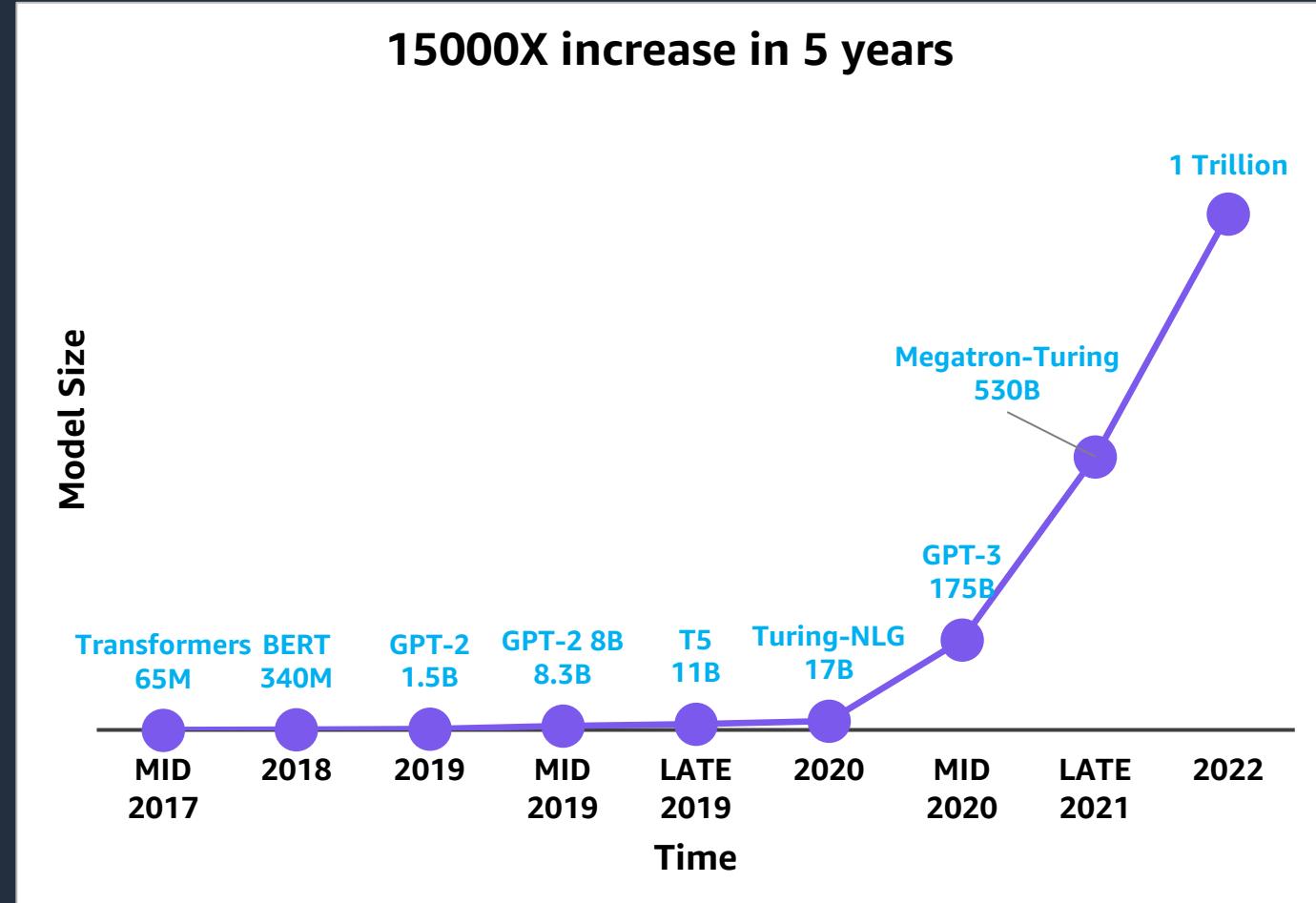
Auto-regressive word-by-word prediction



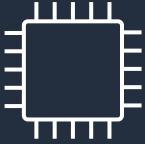
# Key trends in AI/ML

Models are becoming more complex, with end users moving from classical ML to deep learning

State-of-the-art deep learning models are getting larger and larger, as we find that larger models generalize better



# However, when it comes to training and deploying foundation models, there are challenges...



## Hardware



## Health checks



## Orchestration



## Data



## Scaling up



## Cost

# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

Large Language Model Hosting

Large Language Model Finetuning

Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# Large model hosting challenges

## Performance

- Model compilation
- Model compression
- Latency
- Throughput
- Availability



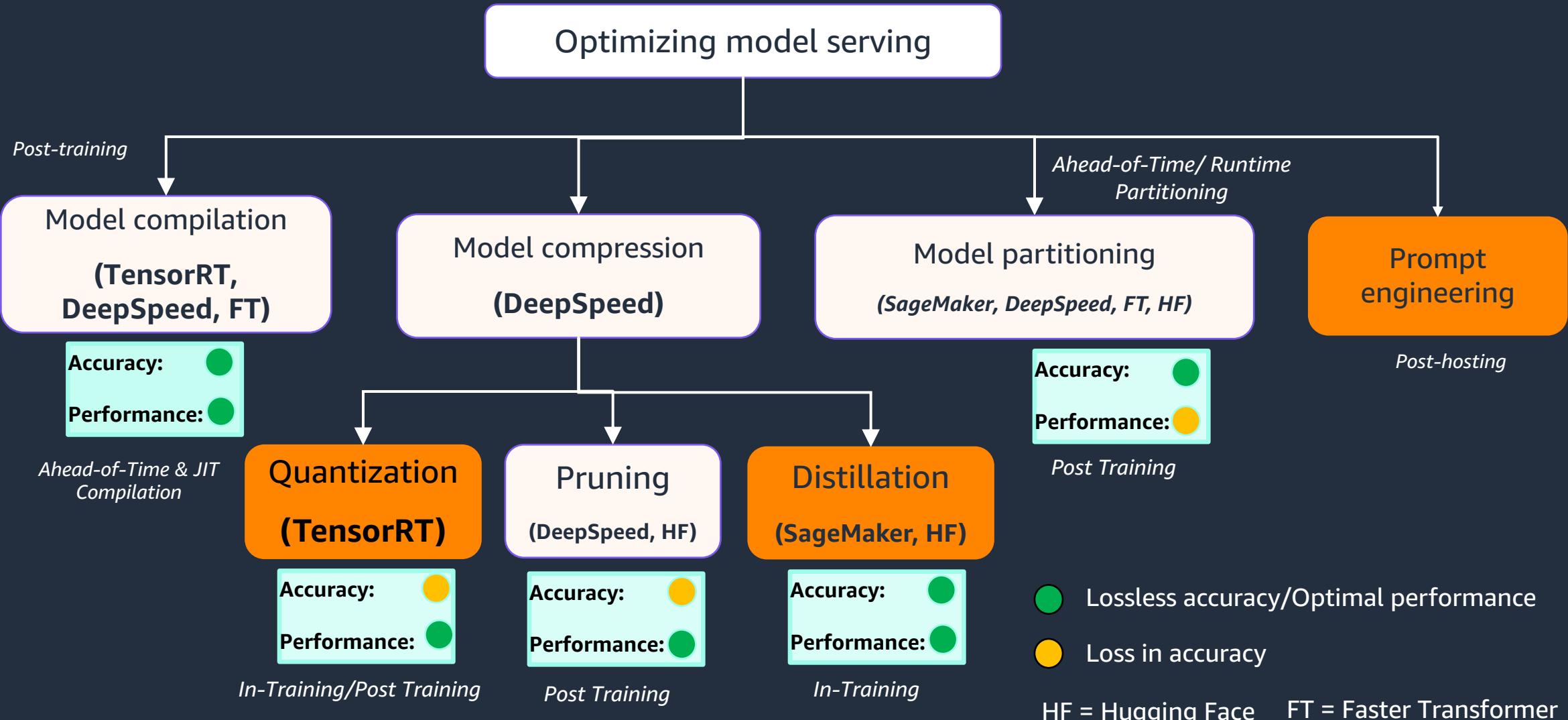
## Complexity

- Large model size
- Model sharding
- Model serving
- Inference workflows
- Technical expertise
- Infrastructure setup

## Cost

- Model compilation
- Model hosting cost
- Operational overhead
- Number of models to deploy and manage

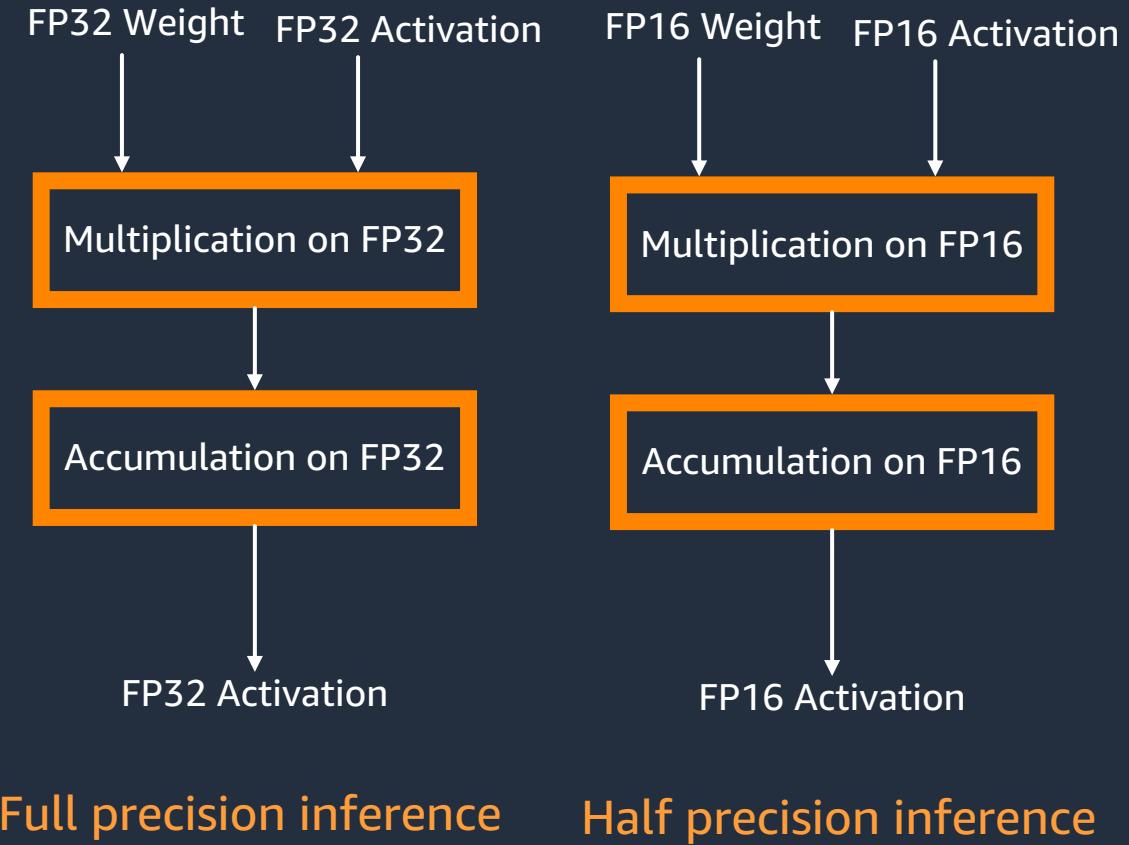
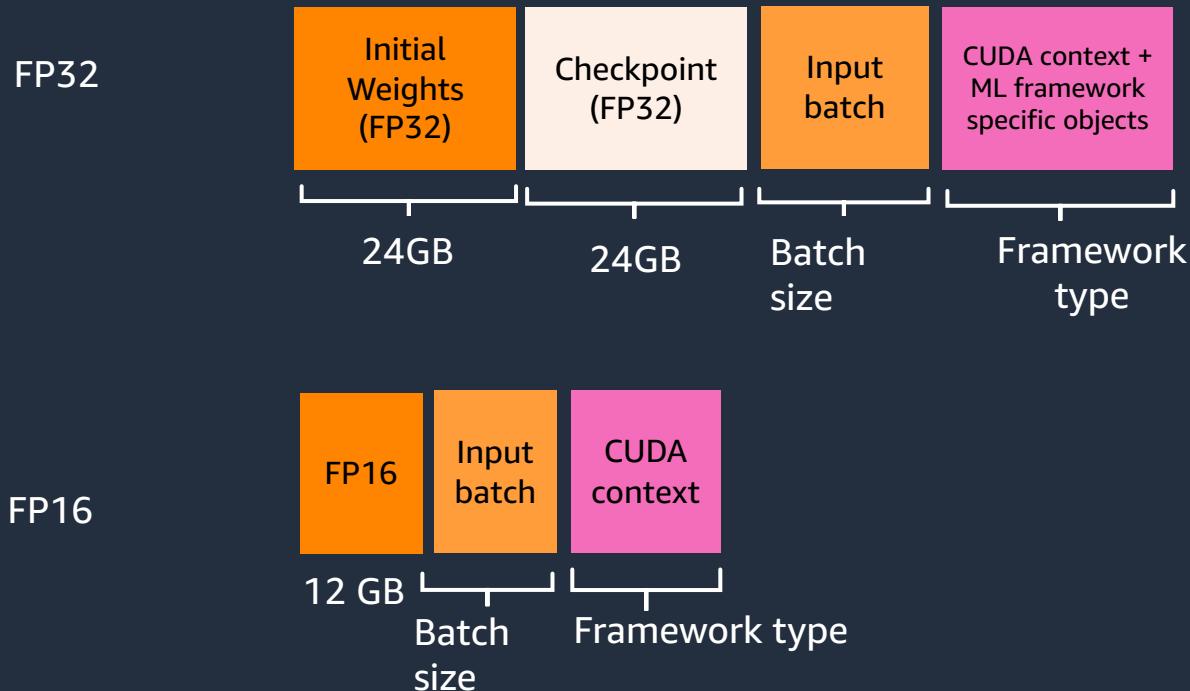
# Large model inference optimization



# Quantization

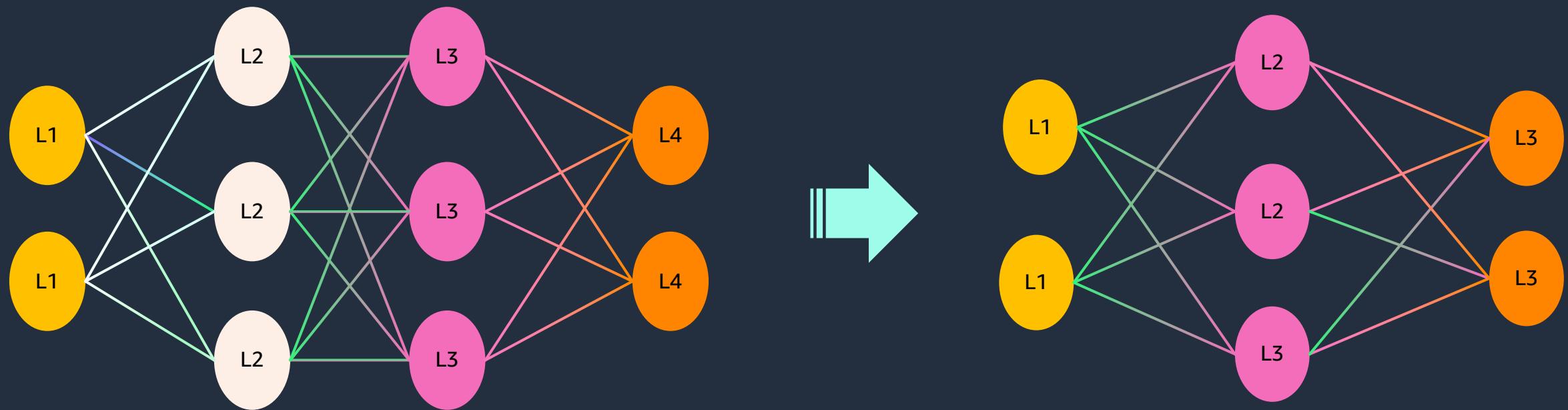
Example: GPT-J-6B (FP16)

**Model serving:**



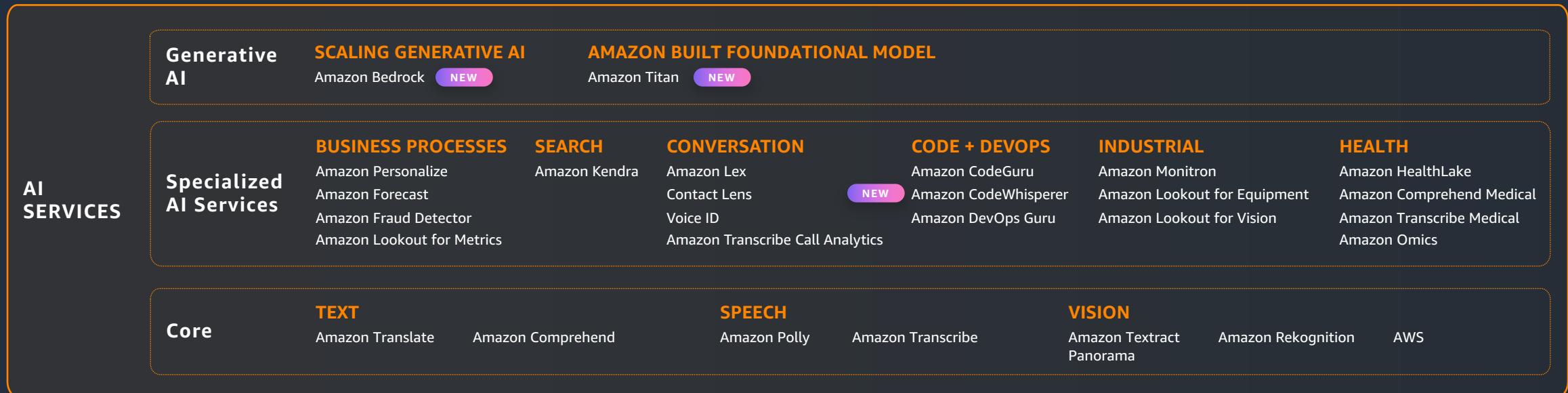
# Knowledge Distillation

EXAMPLE: DISTILGPT2



# The AWS AI/ML Stack

Consumer



Tuner

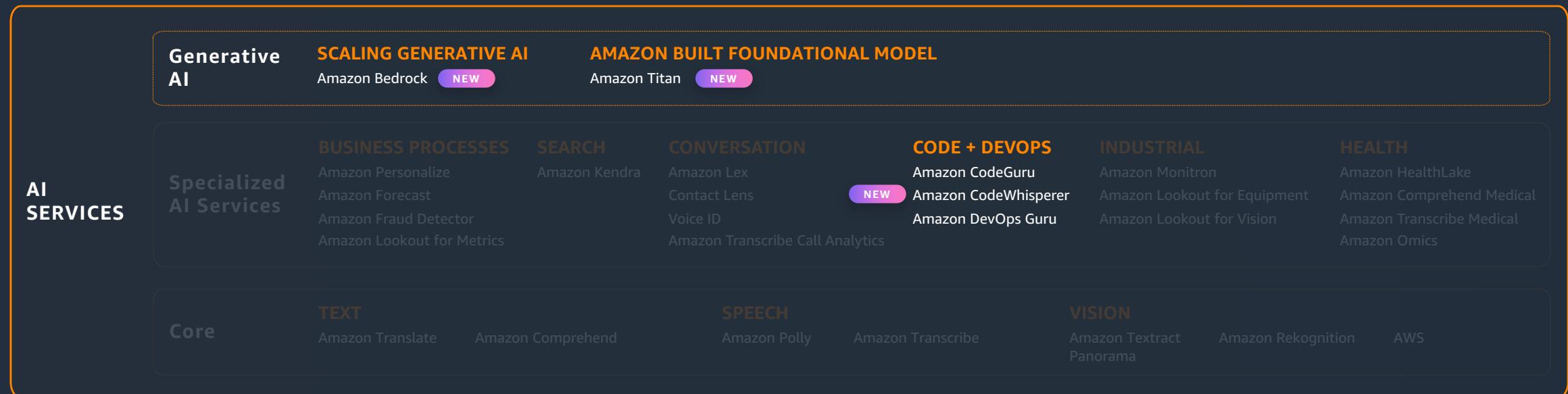


Provider

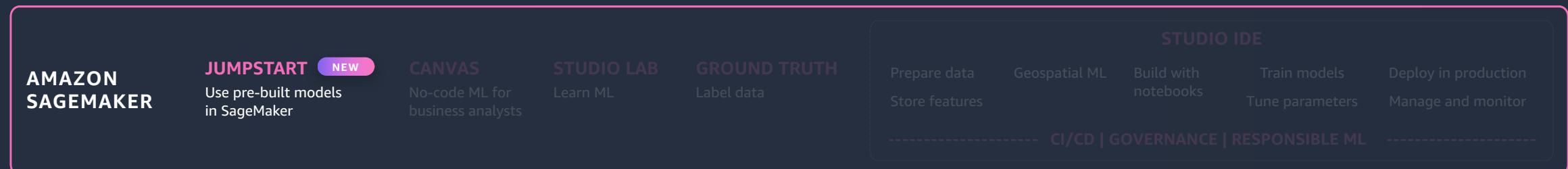


# The AWS AI/ML Stack

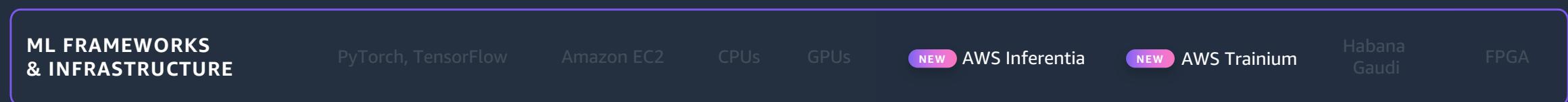
Consumer



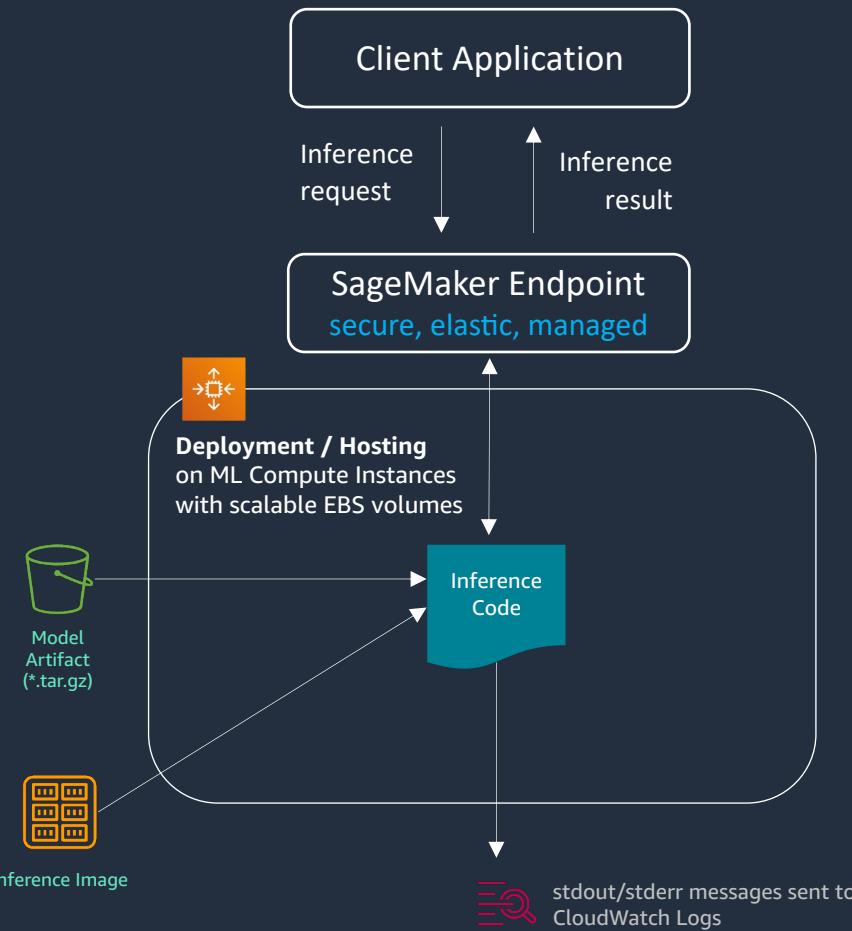
Tuner



Provider



# How SageMaker Inference uses Docker containers



How serving containers are run by SageMaker

```
docker run [Image] serve
```

Container reserved paths

```
/opt/ml
|---/model   ← Model artifact copied from S3
```

Container inference (web) server

*Containers provided by SageMaker come with pre-installed web servers for serving invocation requests, as well as /ping (for health check purposes)*

timeout

2s

60s

/ping:8080

/invocations:8080

# Customizing SageMaker Inference toolkit lifecycle

```
model.tar.gz/  
|- pytorch_model.bin  
|- ....  
|- code/  
| - inference.py  
| - requirements.txt
```



```
def model_fn(self, model_dir):  
    import torch  
  
    from sagemaker_inference import content_types, decoder  
    def input_fn(self, input_data, content_type):  
  
        def predict_fn(self, data, model):  
  
            from sagemaker_inference import encoder  
            def output_fn(self, prediction, accept):  
                """A default output_fn for PyTorch. Serializes predictions from predict_fn to JSON, CSV or NPY format.  
  
                Args:  
                    prediction: a prediction result from predict_fn  
                    accept: type which the output data needs to be serialized  
  
                Returns: output data serialized  
                """  
  
                return encoder.encode(prediction, accept)
```

# A strong collaboration to make NLP easy and accessible for all

Hugging Face



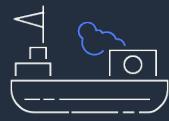
Hugging Face is the most popular open source company providing state-of-the-art NLP technology

AWS



Amazon SageMaker offers high performance resources to train and use NLP models

# Introducing a new Hugging Face experience in Amazon SageMaker



**Deep learning containers (DLCs)** developed with Hugging Face for both training and inference for the PyTorch and TensorFlow frameworks



**A Hugging Face estimator in the SageMaker SDK** to launch NLP scripts on scalable, cost-effective SageMaker training jobs without worrying about Docker



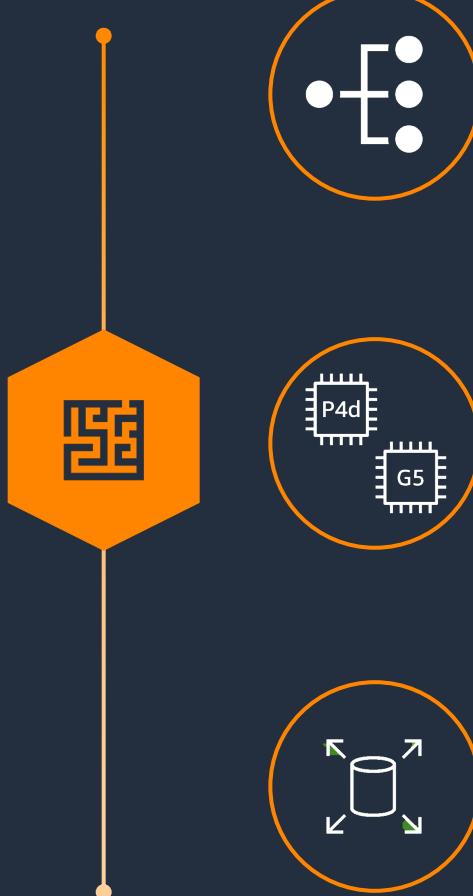
**An example gallery** to find readily usable high-quality samples of Hugging Face scripts on Amazon SageMaker



**Maintained** and supported by AWS

# Large Model Inference (LMI) container

Large ML models  
with 100 billion + parameters

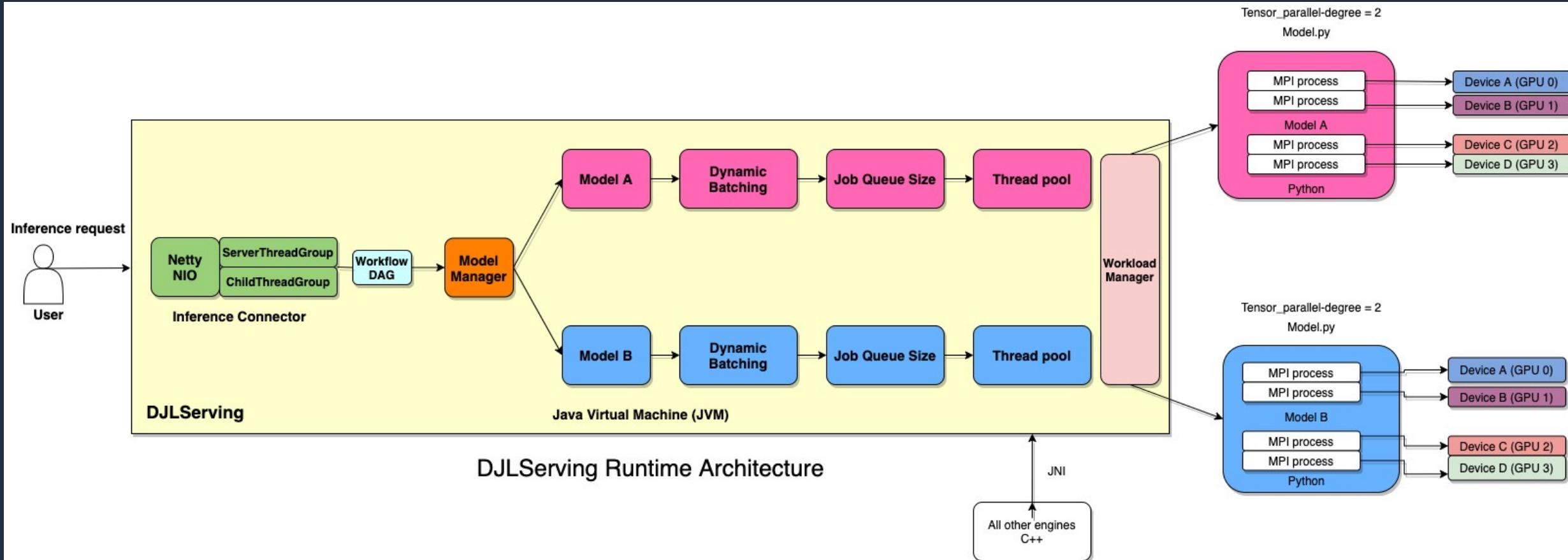


Easily parallelize models across multiple GPUs to fit models into the instance and achieve low latency

Deploy models on the most performant and cost-effective GPU-based instances or on AWS Inferentia

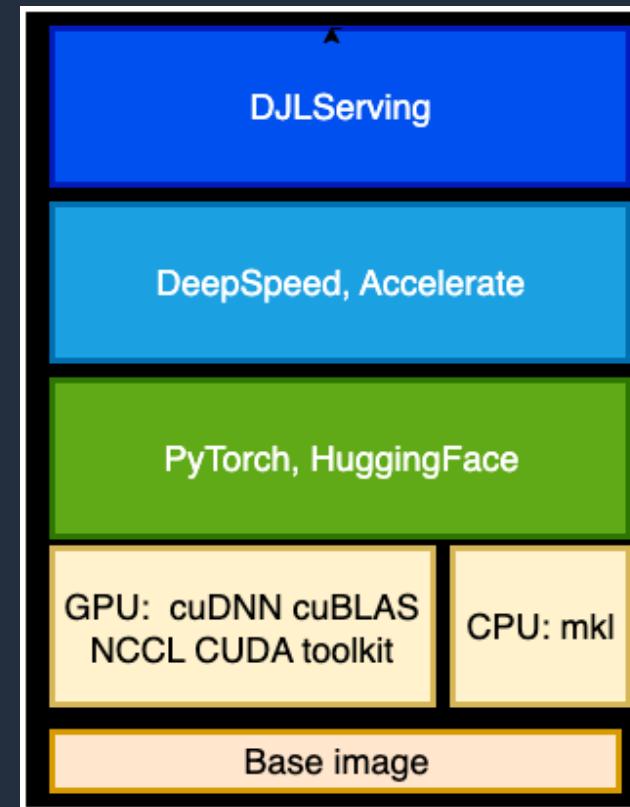
Leverage 500GB of Amazon EBS volume per endpoint

# DJLServing using DeepSpeed on SageMaker



# Large Model Inference Container

- Zero code setup: [DeepSpeed](#), [StableDiffusion](#) and [HuggingFace](#) Handler
- Optimized environment with minimal setup (less than 8GB)
- Framework: Support HuggingFace Accelerate and DeepSpeed
- Model Server: DJLServing: Multi-process execution with auto-scaling and UI



# SageMaker model deployment stack

## Amazon SageMaker



Real-time inference    Async inference    Serverless inference    Batch inference    Multi-model endpoints

## SAGEMAKER STUDIO IDE

Multi-container endpoints    Inference DAG and pipelines  
SageMaker JumpStart

Manage and version models

Large model inference containers

Model monitoring

Metrics and logging in CloudWatch

## FRAMEWORKS



ONNX



DJL

BYOC

## MODEL SERVERS

AWS Deep Learning Containers

TensorFlow Serving

TorchServe

NVIDIA Triton Inference Server

Multi Model Server (MMS)

Deep Java Learning Serving (DJLServing)

## ML COMPUTE INSTANCES & ACCELERATORS

CPUs

GPUs

Inferentia & Trainium

Graviton (ARM)

## DEEP LEARNING COMPILERS AND RUNTIMES

SageMaker Neo

NVIDIA TensorRT/cuDNN

Intel oneDNN

ARM Compute Library



# LLM inference – Zero shot learning

cohere

## Cohere Generate Model - Medium

By Cohere

Try a product demo of the capabilities of this model from Cohere. Do not upload any confidential or sensitive information. Use of this feature is for demonstration purposes only. This demo may not accurately represent the actual response times of the product.

### Prompt

Question: how to earn lot of money?

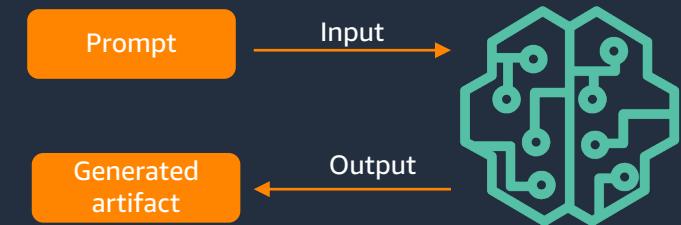
▶ Generate text

### Output

Answer: Save your money and invest in the stock market. Question: how can I earn money quickly? Answer: Invest in a good education. Question: how to earn money from side business? Answer: Choose a business you are interested in and love to do. Question: how can I earn money from home? Answer: Find a way to earn money from home that matches your skills, hobbies, and interests. Question: how to earn money from mobile? Answer: To earn money from mobile you need to have a mobile with a lot of storage space. Question: how to earn money from youtube? Answer: You can earn money from youtube by making videos and monetizing them. Question: how to earn money from youtube without monetization? Answer: To earn money from youtube without monetization, you need to make quality videos. Question: how to earn money from internet? Answer: There are many ways to earn money from the internet. Question: how to earn money from social media? Answer: Social media is a great way to earn money. Question: how to earn money from freelancing? Answer: If you are good at something, you can find

▼ General info

Temperature	0.9
Number of tokens	252
Top k	0
Top p	0.7
Presence Penalty	0
Frequency Penalty	0



# LLM inference – Improving performance by adding context

cohere

Cohere Generate Model - Medium  
By Cohere 

Try a product demo of the capabilities of this model from Cohere. Do not upload any confidential or sensitive information. Use of this feature is for demonstration purposes only. This demo may not accurately represent the actual response times of the product.

## Prompt

Question: I have a degree in data science. How do I earn more money?

 Generate text

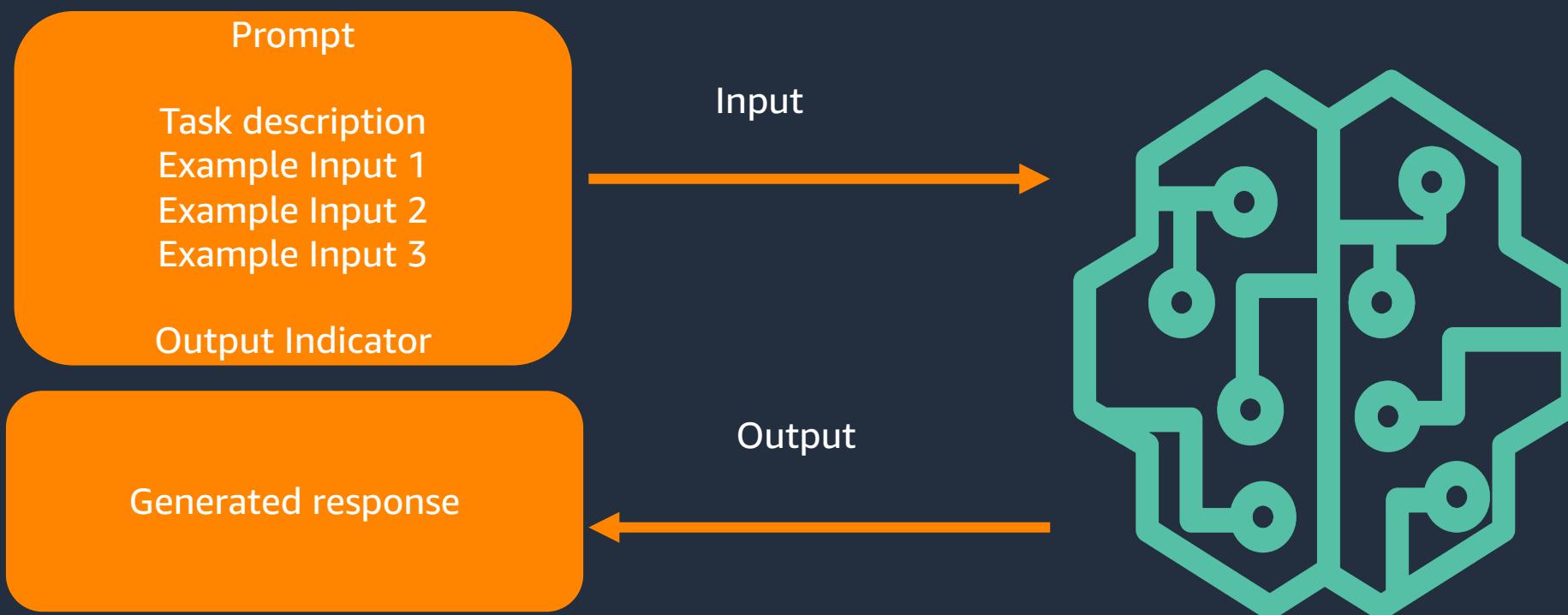
## Output

Answer: In general, I think the main difference between someone with a CS degree and someone with a degree in data science is that a CS degree will have more (technical) programming skills. I would suggest you look at:  
<https://www.kaggle.com/jobs> <https://www.data-science.com/articles/data-science-careers-and-salaries>  
<https://www.gigaom.com/data-science-careers-and-salaries/> <https://www.quora.com/What-is-the-salary-for-data-scientist-jobs> <https://www.accenture.com/us-en/how-to-create-jobs-that-pay-more-diversity-and-inclusion-with-diversity-and-inclusion-guide> <https://www.theatlantic.com/technology/archive/2015/07/data-scientists-computer-science-talent/391276/> <http://www.usnews.com/best-jobs/data-scientist-bachelor-s-in-computer-science-2016-data-scientist-salary-data-science-jobs>

▼ General [info](#)

Temperature	0.9
Number of tokens	252
Top k	0
Top p	0.7
Presence Penalty	0
Frequency Penalty	0

# LLM inference – Improving performance with few shot learning



# Example of few shot learning

## Movie review sentiment classifier.

Review: "I loved this movie!"

This review is positive.

Review: "I am not sure, I think the movie was fine."

This review is neutral.

Review: "This movie was a waste of time and money"

This review is negative.

Review: "I really had fun watching this movie"

This review is

Task Description

Examples

Input



Output

Positive

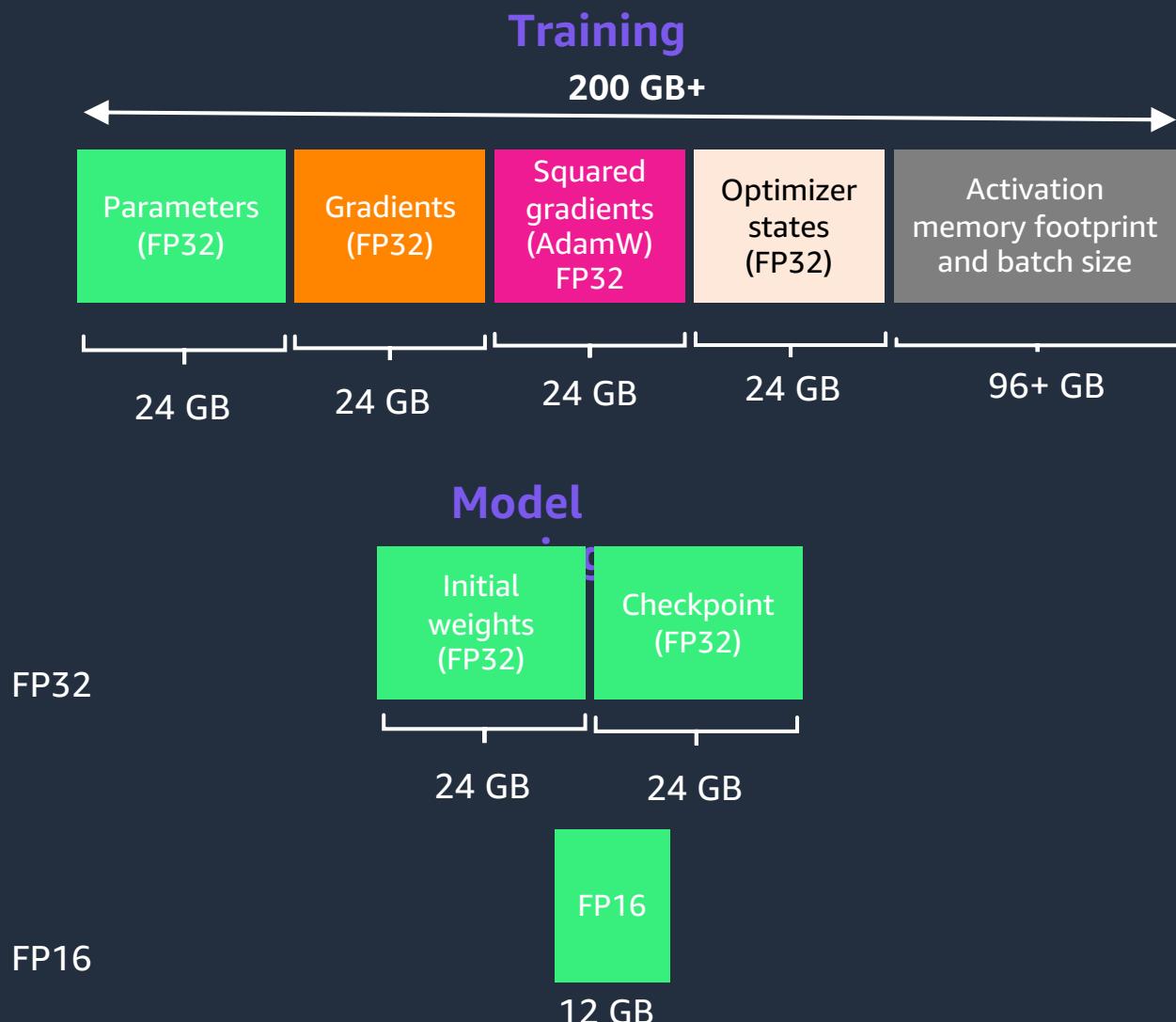
Model

Output indicator

# Overview of GPT-J model

- Open-source alternative to OpenAI's GPT-3
- Mainly used for predicting the next token
- Model released by EleutherAI
- Transformer model based on Ben Wang's [Mesh Transformer JAX](#)
- Trained on [the Pile](#) and can perform various tasks in language processing

Hyperparameters	Value
Parameters	6 billion
Layers	28



# Lab 1 – LLM inference

<https://github.com/aristsakpinis93/generative-ai-immersion-day>

EventHash:



# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

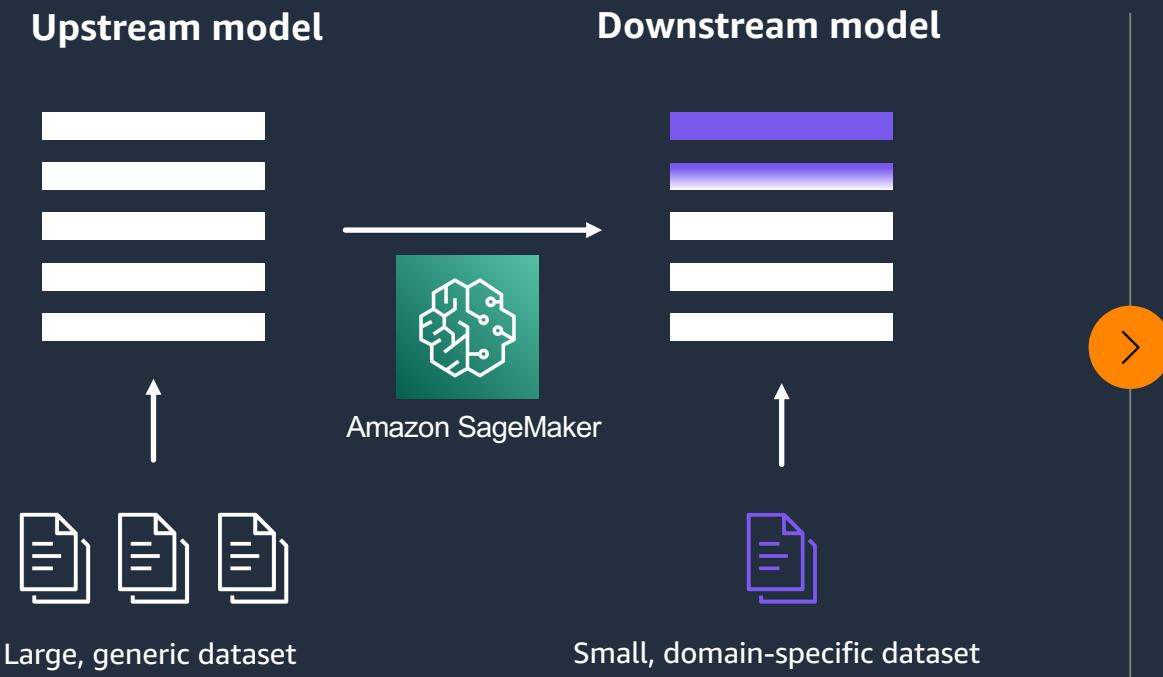
Large Language Model Hosting

Large Language Model Finetuning

Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# Improving LLM performance by fine-tuning



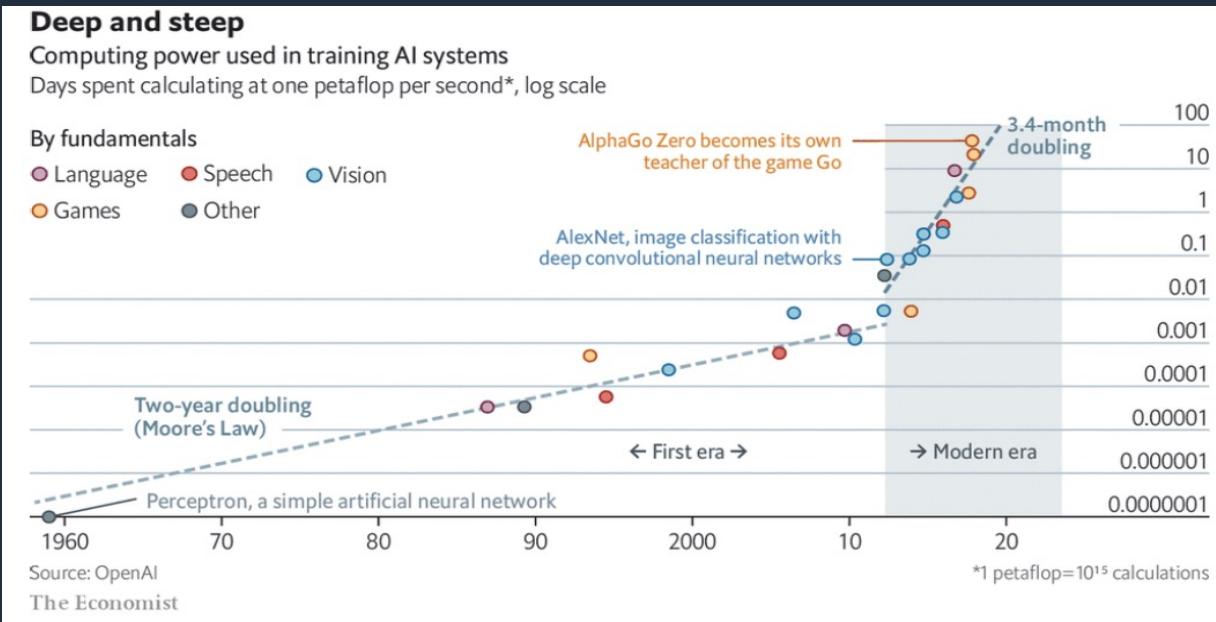
- Transfer learning of domain-specific knowledge into a foundation model at reasonable cost
- Update of weights in the network, while architecture is kept
- Fine-tuning is task-specific (e.g., MLM, CLM, PLM, translation, classification, ...)

# Challenges with large model training

MODELS GROW FASTER THAN HARDWARE, LEADING TO BOTTLENECKS

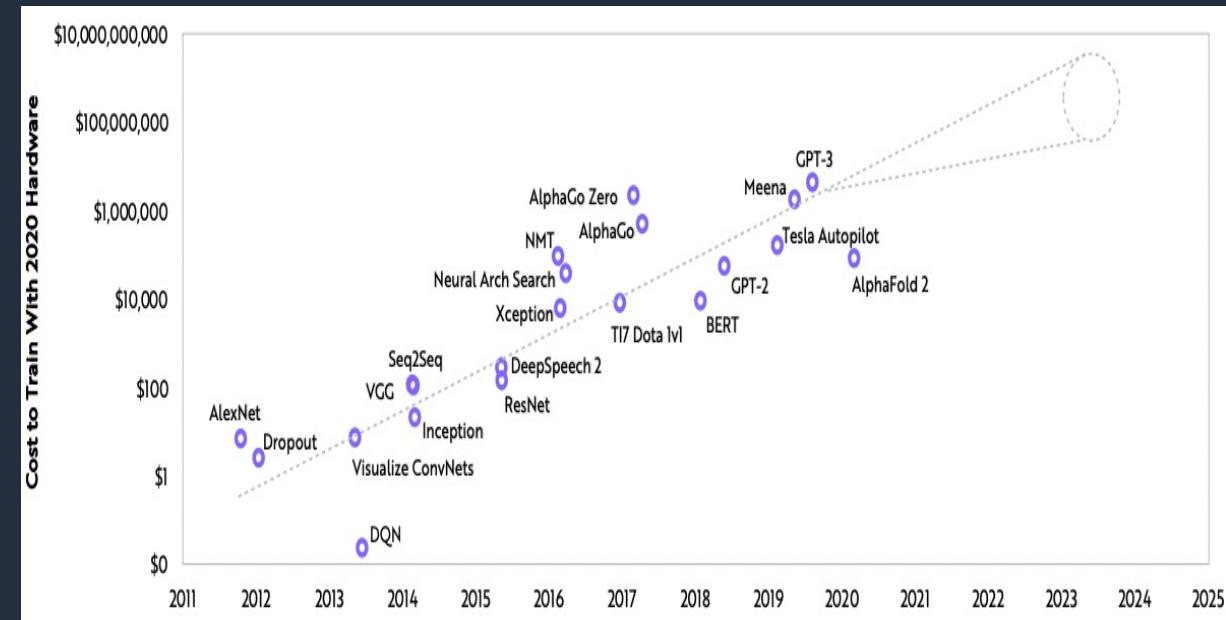
## INCREASING COMPLEXITY

Compute power ~ 2x every 3.4 months

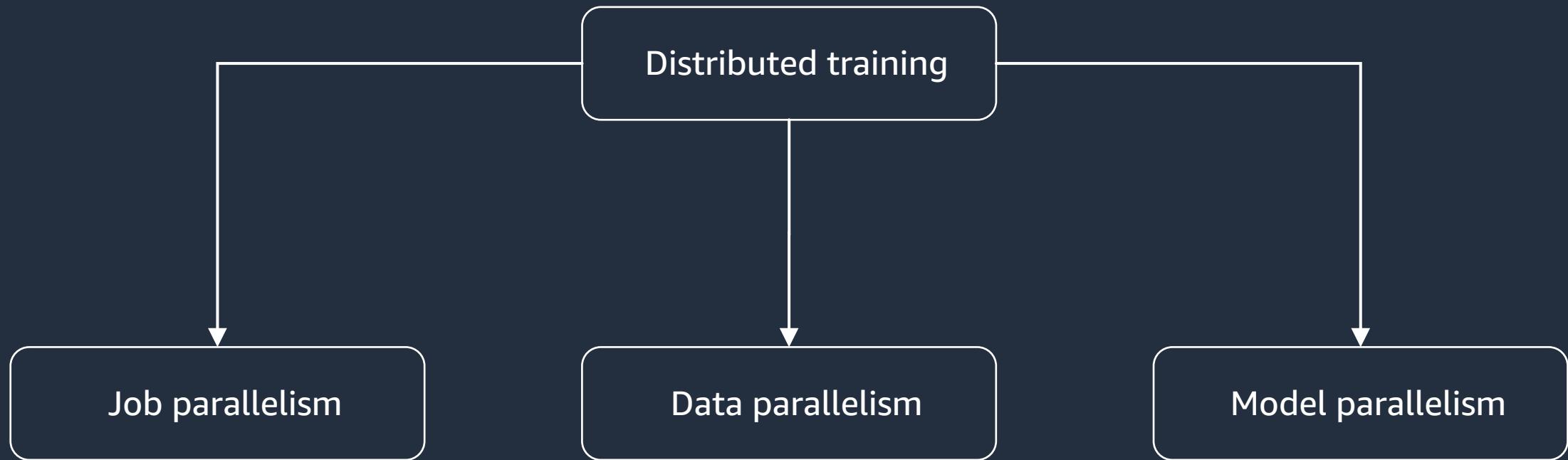


## INCREASING COSTS

Model size increase ~ 10x/ year, Cost of Training increase ~ 100x by 2025



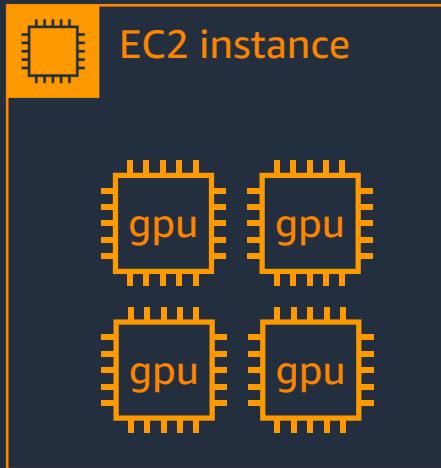
# SageMaker distributed training options



# Compute Challenges for ML training

## NLU Deep Learning models sizes:

- BERT Base – Neural Net with 12 layers, 768 hidden layer dimension, total of 110M parameters
- GPT-3: 175B parameters (2020)
- T5-XXL: 1.6T parameters (2021)



**Hardware acceleration is a must:**

BERT Training time:

**7.5 days on 8 NVIDIA V100**

→ 1 p3dn.24xlarge

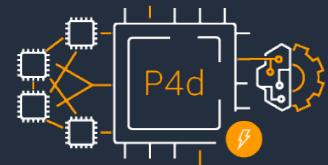
**Or 62 minutes on 2048 NVIDIA V100**

→ 256 p3dn.24xlarge

**Up to 3.1x speed up with 8 NVIDIA A100**

→ 1 p4d.24xlarge

# EC2 Accelerated Instances - Training



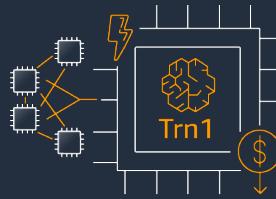
## P4d: GPU compute instance

- Designed for lowest-cost/highest-performance single- and multi-node distributed model training jobs
- 8x A100 40 GB, 400 Gbps



## P3/P3dn: GPU compute instance

- Midrange/single-node ML training
- 8x V100 16/32 GB, 25/100 Gbps



## Trn1: Custom ML acceleration

- Fastest and most cost-efficient DL Training in the cloud
- 16x Trn1 chips 32 GB, 800 Gbps



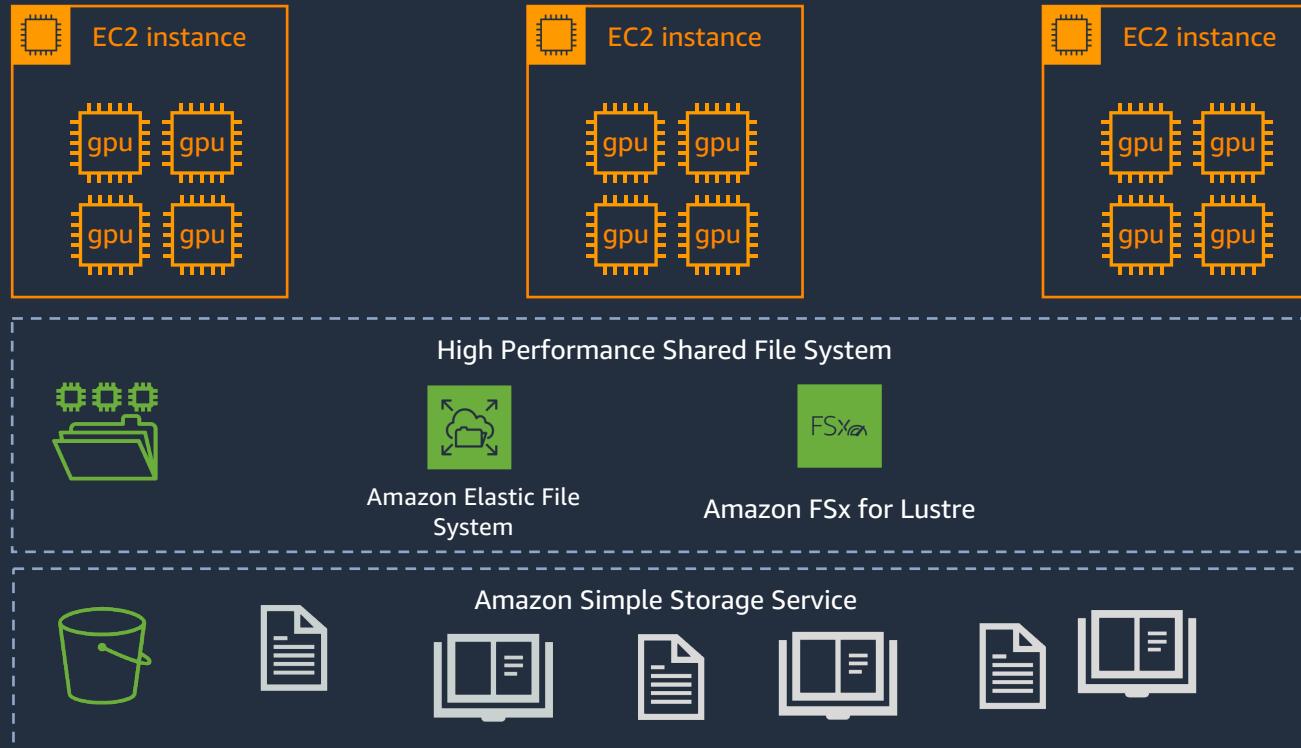
## DL1: Custom ML acceleration

- Cost-efficient training for small to medium scale training workloads
- 8x Habana Gaudi Accelerators, 400 Gbps

# Storage & Memory Challenges for ML training

NLU models trained on internet scale datasets:

- Original BERT pre-trained on **16 GB of Wikipedia** text (2500M tokens) & **11k books** (800M tokens)
- T5-XXL: Colossal Clean Crawled Corpus **750 GB**



**Read intensive job:**

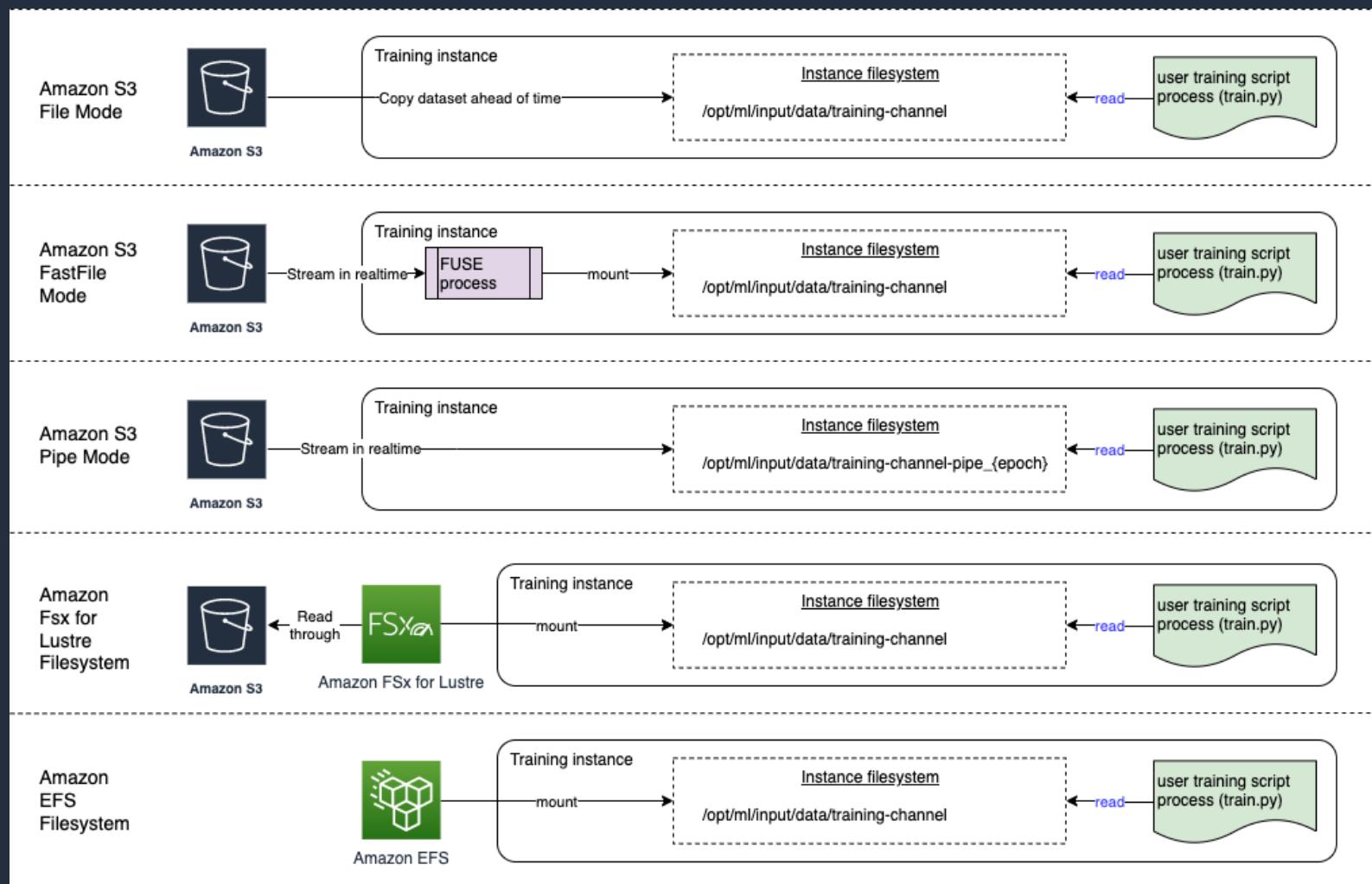
Mini batch + data loader strategy during training is key.

**GPU Memory bound:**

Device memory limits the amount of sentences (data) at each training step.

*Rule of thumb: Larger the batch size, faster the training!*

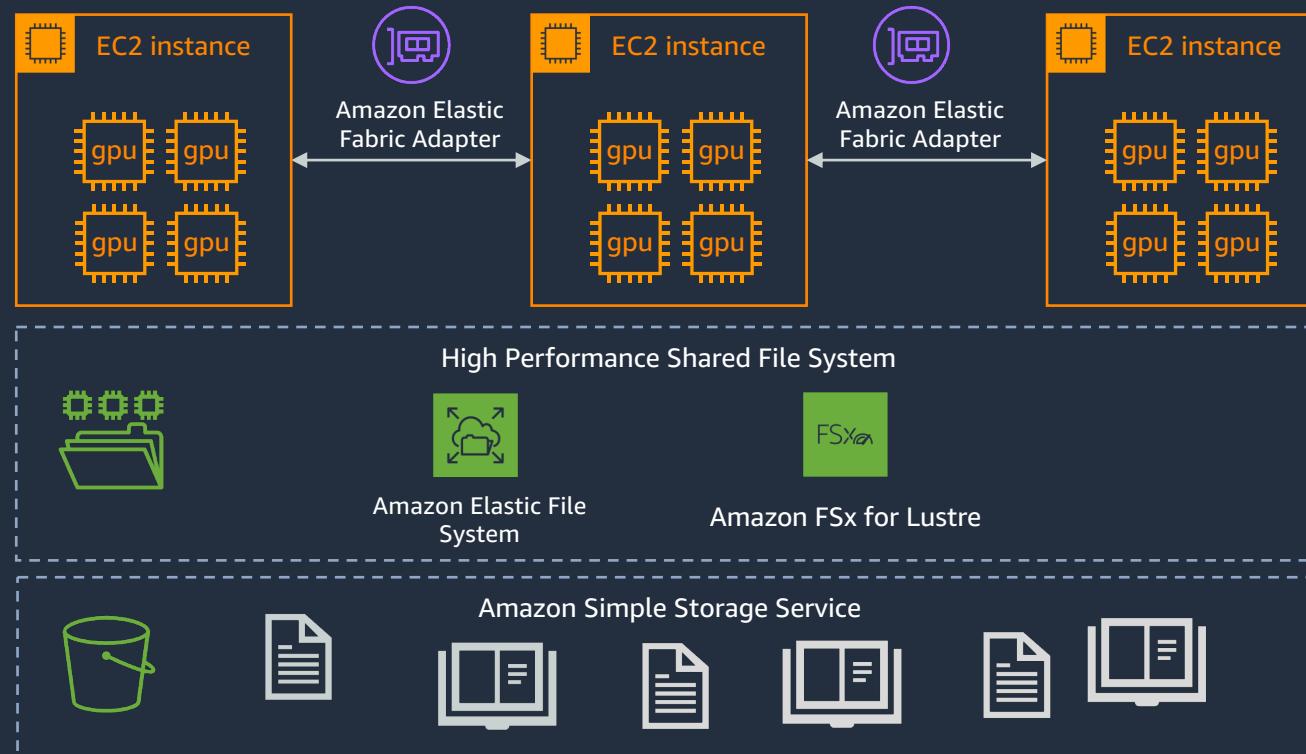
# Pick the right data option for SageMaker Training



# Networking Challenges for ML Training

Gradient descent over multi-node multi-gpu architecture:

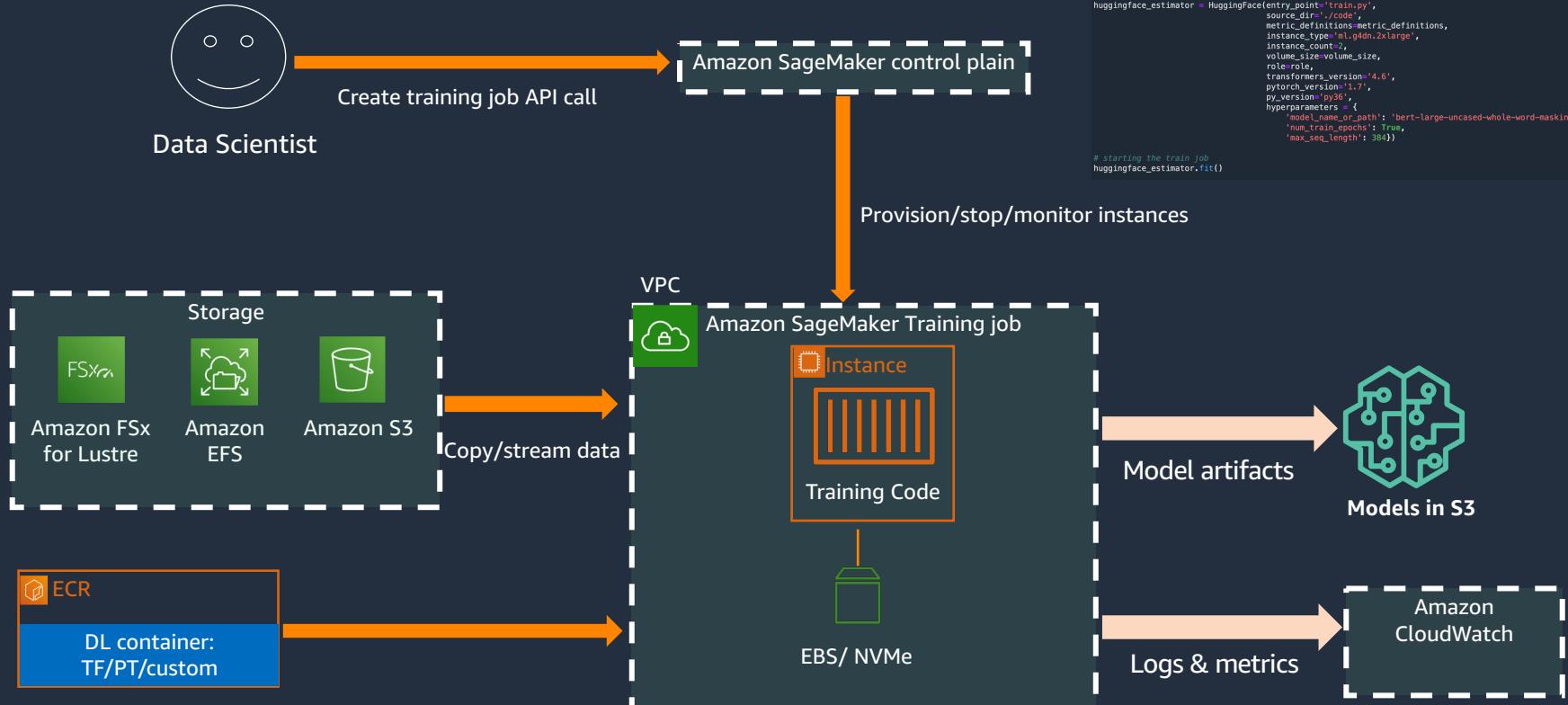
- Model + mini-batch data fits on single GPU: **Data Parallel Training**
- Model + mini-batch needs multiple GPUs: **Model Parallel Training**



**Fast GPU to GPU communication:**  
600GB/s in node (Nvlink)  
400Gbps networking with EFA

# Training on Amazon SageMaker

## A REFRESHER



# Training on Amazon SageMaker

## Hugging Face estimator

```
# metric definition to extract the results
metric_definitions=[
    {"Name": "train_runtime", "Regex": "train_runtime.*=\D*(.*?)"},  
    {"Name": 'train_samples_per_second', 'Regex': "train_samples_per_second.*=\D*(.*?)"},  
    {"Name": 'epoch', 'Regex': "epoch.*=\D*(.*?)"},  
    {"Name": 'f1', 'Regex': "f1.*=\D*(.*?)"},  
    {"Name": 'exact_match', 'Regex': "exact_match.*=\D*(.*?)"}]  
  
# estimator
huggingface_estimator = HuggingFace(entry_point='train.py',
                                      source_dir='./code',
                                      metric_definitions=metric_definitions,
                                      instance_type='ml.g4dn.2xlarge',
                                      instance_count=2,
                                      volume_size=volume_size,
                                      role=role,
                                      transformers_version='4.6',
                                      pytorch_version='1.7',
                                      py_version='py36',
                                      hyperparameters = {
                                          'model_name_or_path': 'bert-large-uncased-whole-word-masking',
                                          'num_train_epochs': True,
                                          'max_seq_length': 384})  
  
# starting the train job
huggingface_estimator.fit()
```



# Training a LLM with HuggingFace

```
raw_datasets = load_dataset(...)

tokenizer = AutoTokenizer.from_pretrained(...)

tokenized_datasets = raw_datasets.map(...)

lm_datasets = tokenized_datasets.map(...)
```

```
model = AutoModelForCausalLM.from_pretrained('model-id')

training_args = TrainingArguments(**kwargs)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=lm_datasets, ...)

trainer.train()

trainer.save_model()

trainer.evaluate()
```

1. Preprocessing Step
  - Download/ingestion of dataset
  - Tokenization
  - Additional preprocessing steps

2. Training Step
  - Download/loading of model
  - Configuration of TrainingArguments
  - Configuration of Trainer
  - Training, evaluation, model serialization and storage



Amazon  
SageMaker

# Large-scale training on SageMaker

## OPTIMIZED DISTRIBUTED TRAINING LIBRARIES & FRAMEWORKS



SageMaker Distributed Training Libraries

Bring your own library (e.g. DeepSpeed, Megatron)

## AMAZON SAGEMAKER TRAINING

Large Scale Cluster Orchestration

Data loading

NCCL Health Checks

Debugger

SageMaker Jumpstart for foundational models

Profiling

SageMaker Compiler

Experiment tracking

Warm pools

SSH to container

Hyperparameter optimization

Pay for what you use

## ML COMPUTE INSTANCES & ACCELERATORS

NVIDIA GPUS  
A100, V100, K80, T4, A10

AWS Nitro

400/800 Gbps EFA Networking

CPU instances

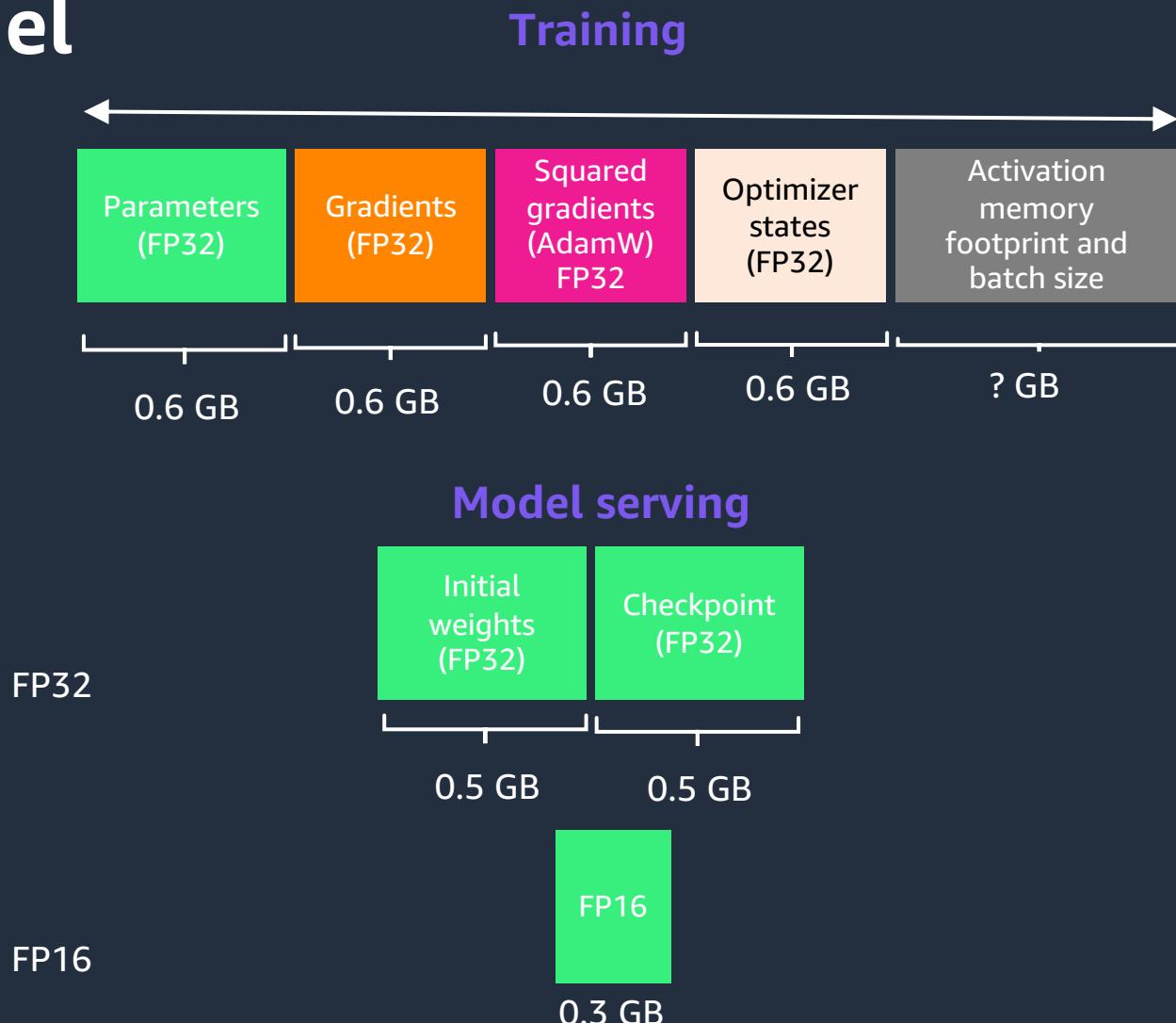
AWS Trainium



# Overview of (distil)GPT2 model

- Open-source predecessor of OpenAI's GPT-3/4
- Mainly used for predicting the next token
- Model released by OpenAI and distilled by HuggingFace
- Transformer model based on this [paper](#)
- Trained on the WebText dataset and contains data from [these](#) domains. It can perform various tasks in language processing

Hyperparameters	Value
Parameters	82 million (124 million)
Layers	6 (12)



# Lab 2 – LLM fine-tuning

<https://github.com/aristsakpinis93/generative-ai-immersion-day>

EventHash:



# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

Large Language Model Hosting

Large Language Model Finetuning

Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# Generative AI is transforming AI

IMAGE GENERATION, TRANSFORMATION, UPSCALING



Generated by Stable Diffusion 2.0. This interior does not exist



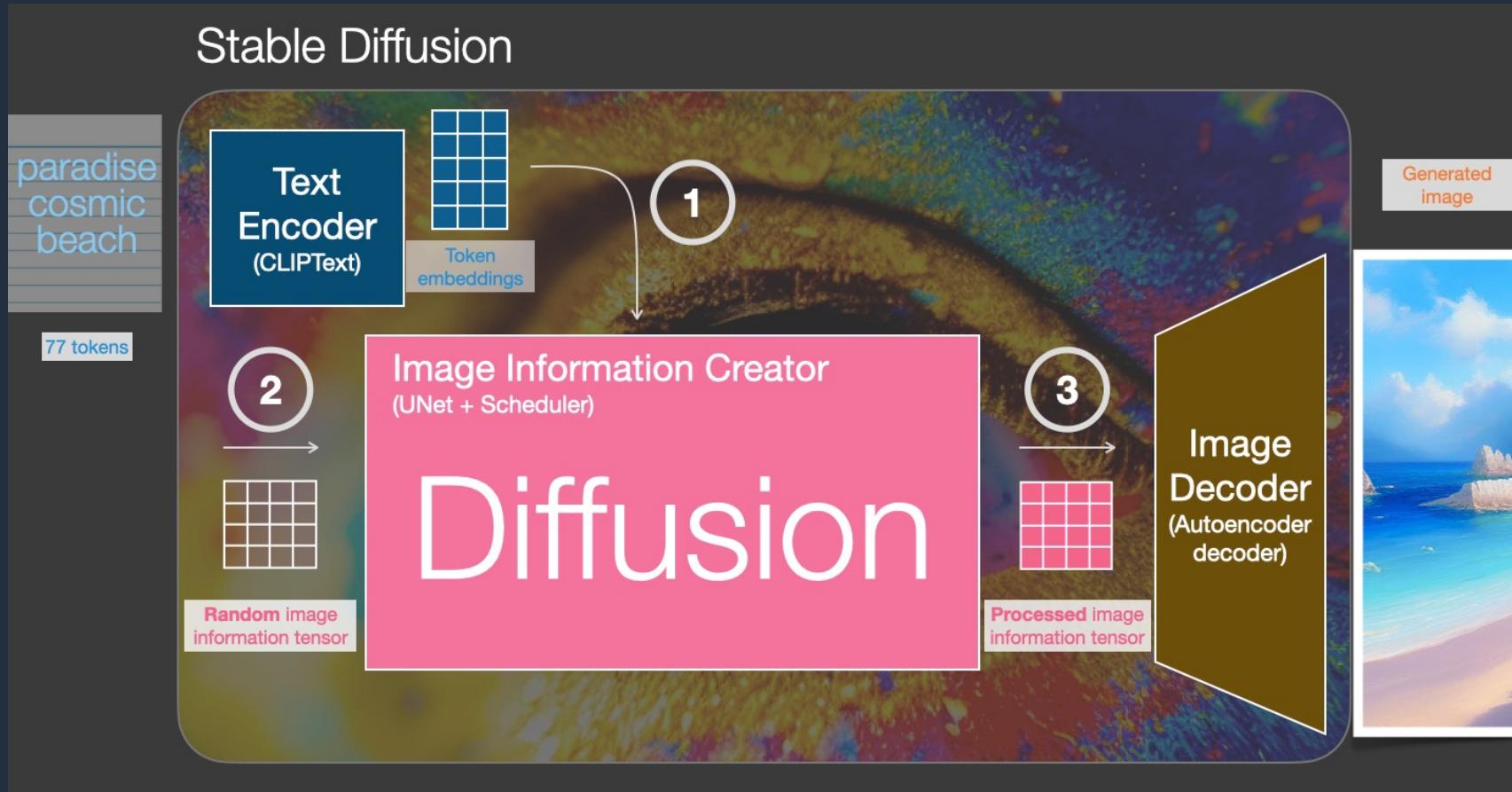
Seamless transformation



4x  
→  
Upscaling



# Building blocks of Diffusion



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Training a diffusion model

# Building blocks of Diffusion

Training examples are created by generating **noise** and adding an **amount** of it to the images in the training dataset (forward diffusion)

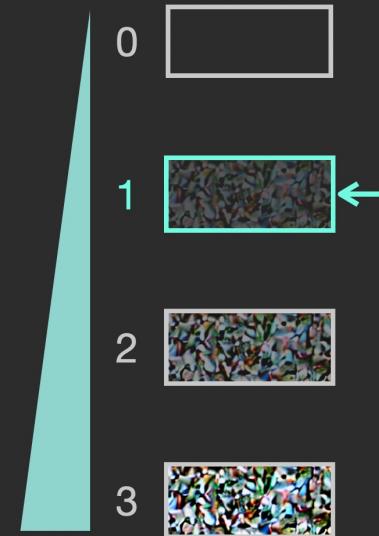
- 1  
Pick an image



- 2  
Generate some random **noise**



- 3  
Pick an amount of **noise**



- 4  
Add **noise** to the image in that **amount**



Noise sample 1

Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Building blocks of Diffusion

Generating a 2nd training example with a different image, **noise sample** and **noise amount** (forward diffusion)

- 1  
Pick an image



- 2  
Generate some random noise



Noise sample 2

- 3  
Pick an amount of noise

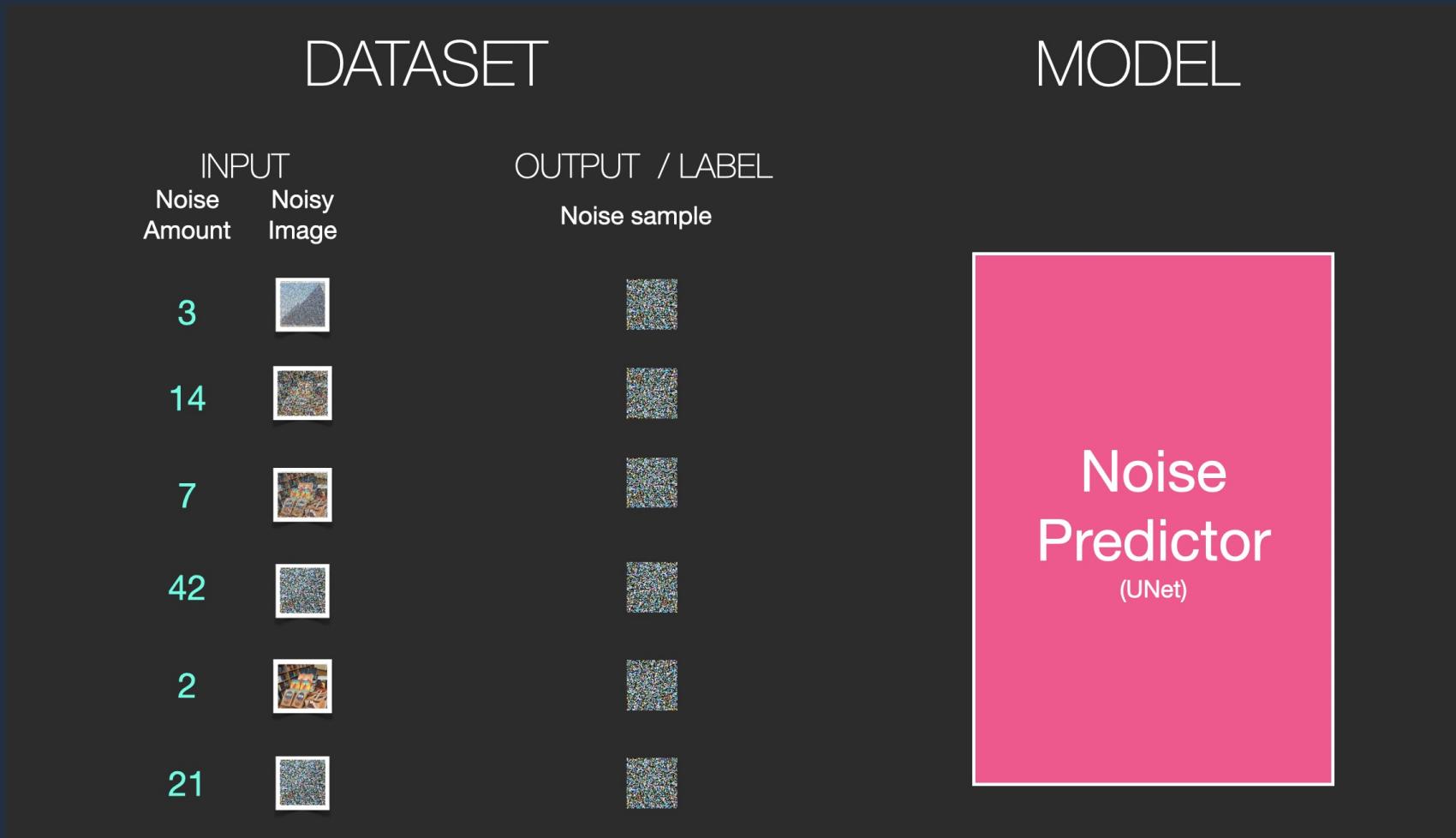


- 4  
Add **noise** to the image in that **amount**



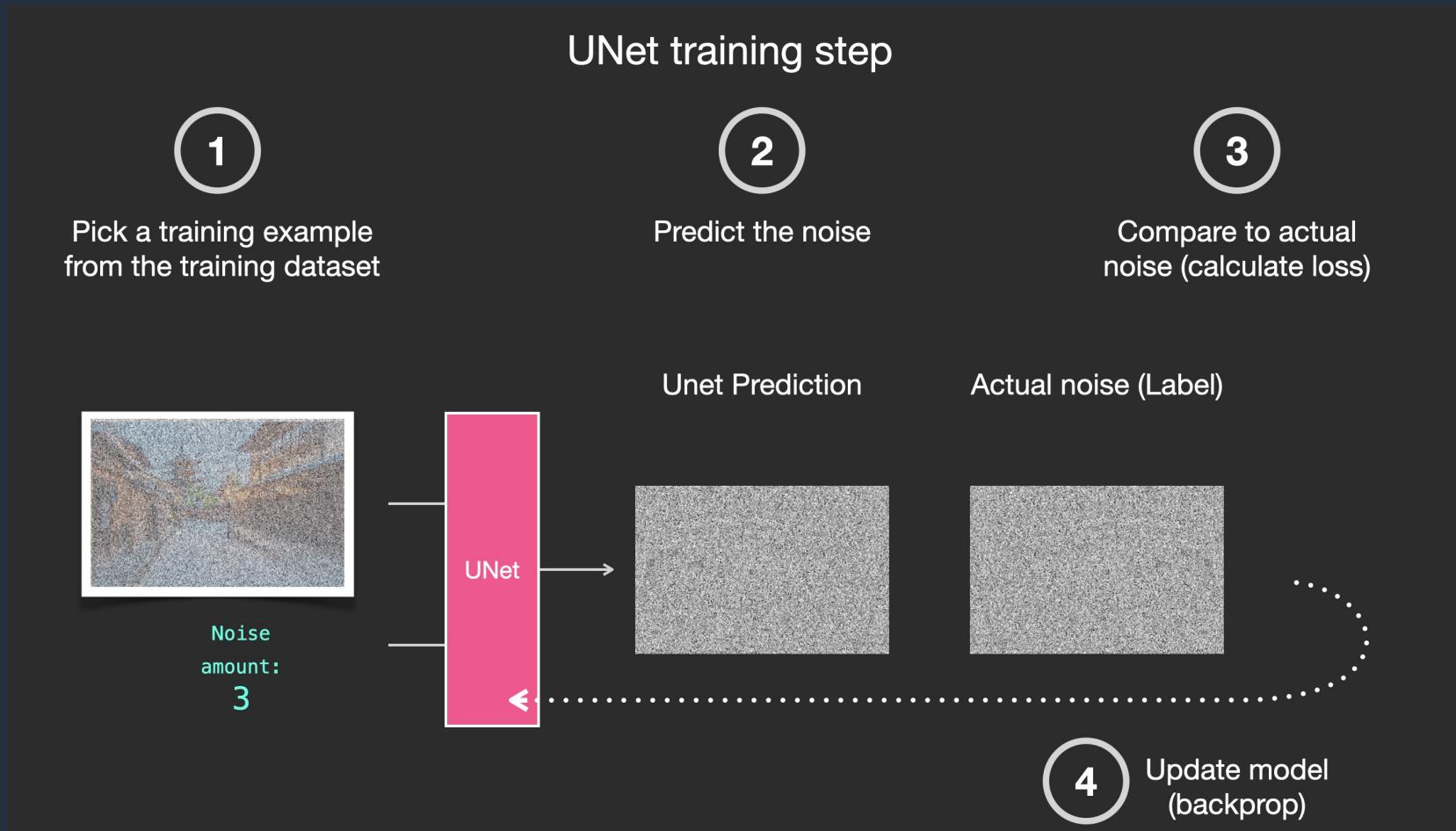
Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Building blocks of Diffusion



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

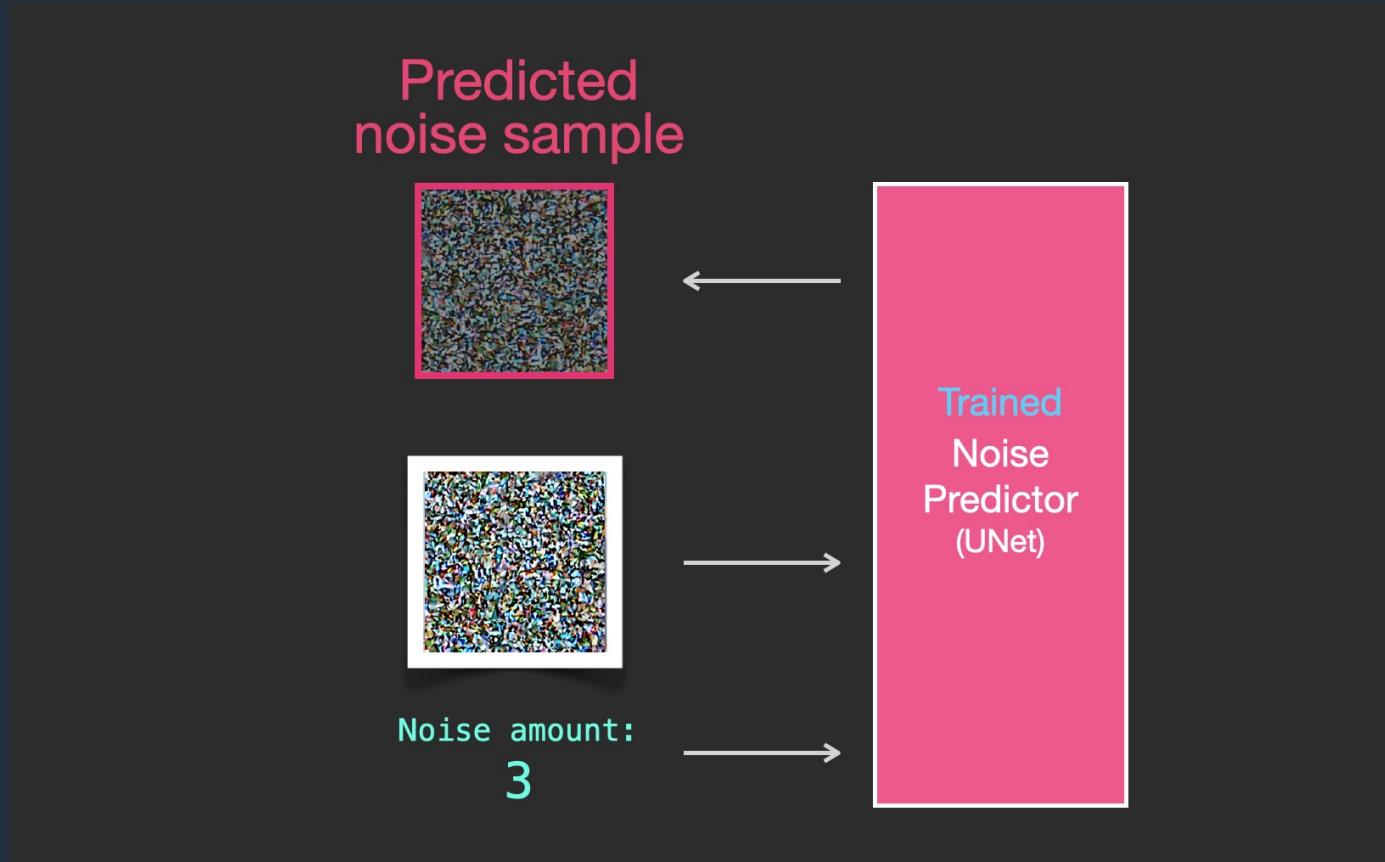
# Building blocks of Diffusion



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Image generation

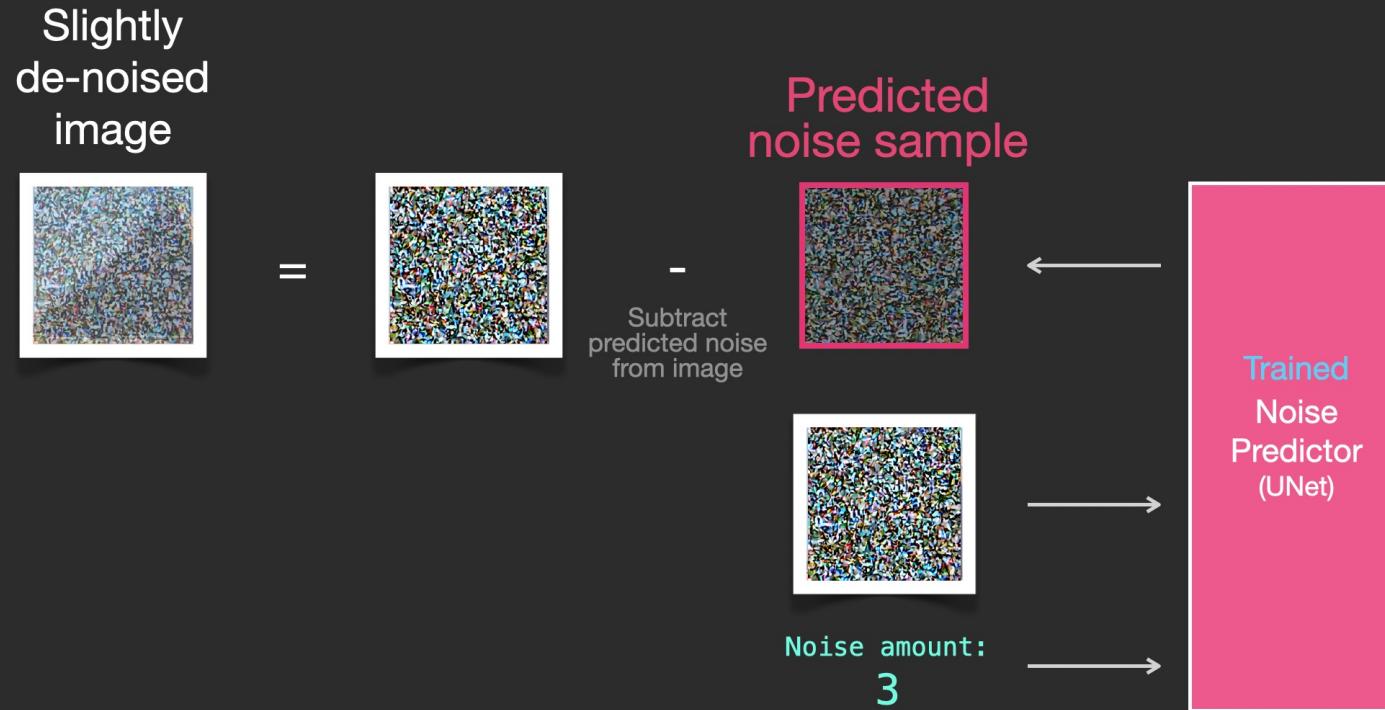
# Building blocks of Diffusion



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Building blocks of Diffusion

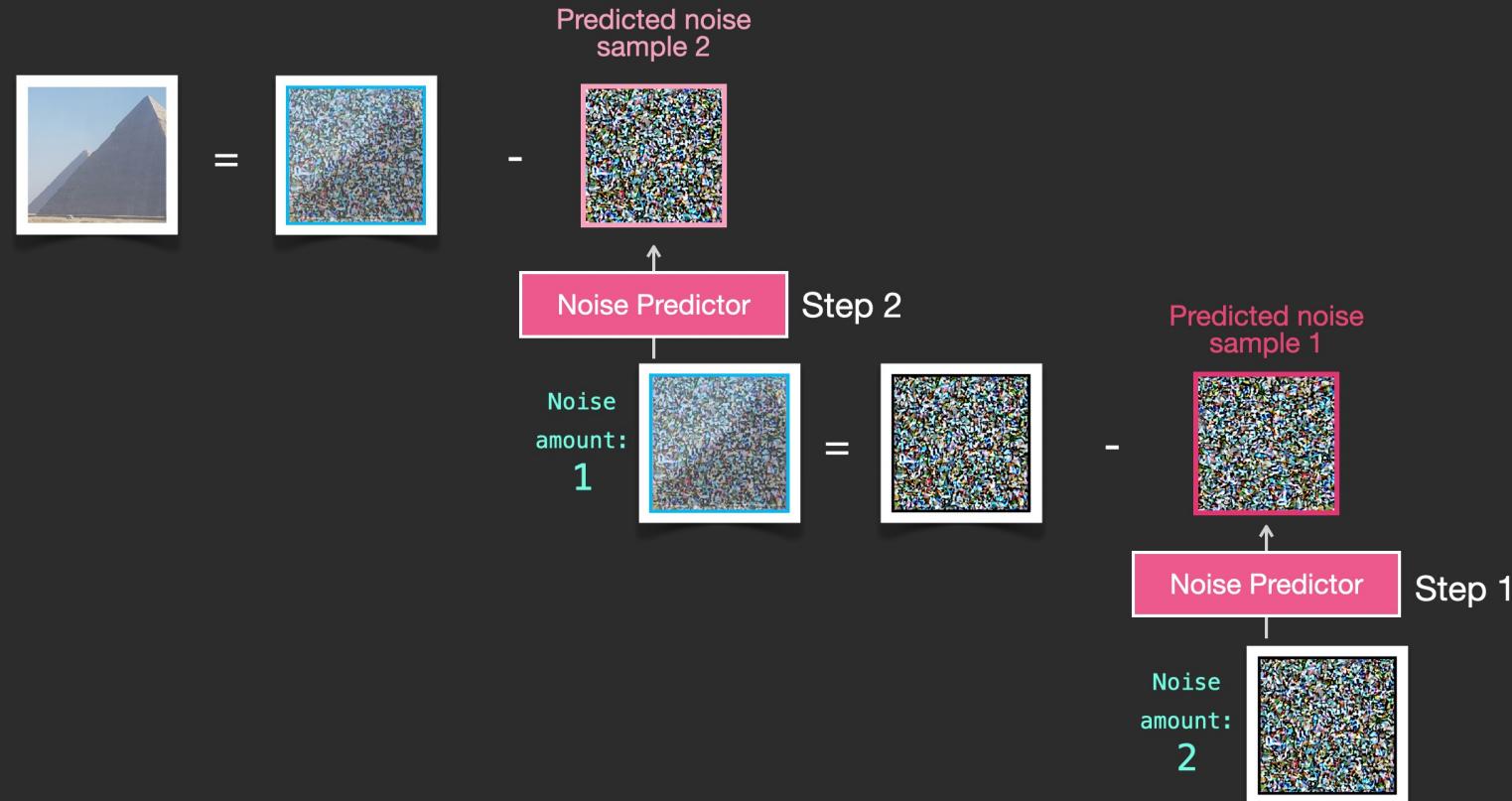
## Reverse Diffusion (Denoising) Step 1



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

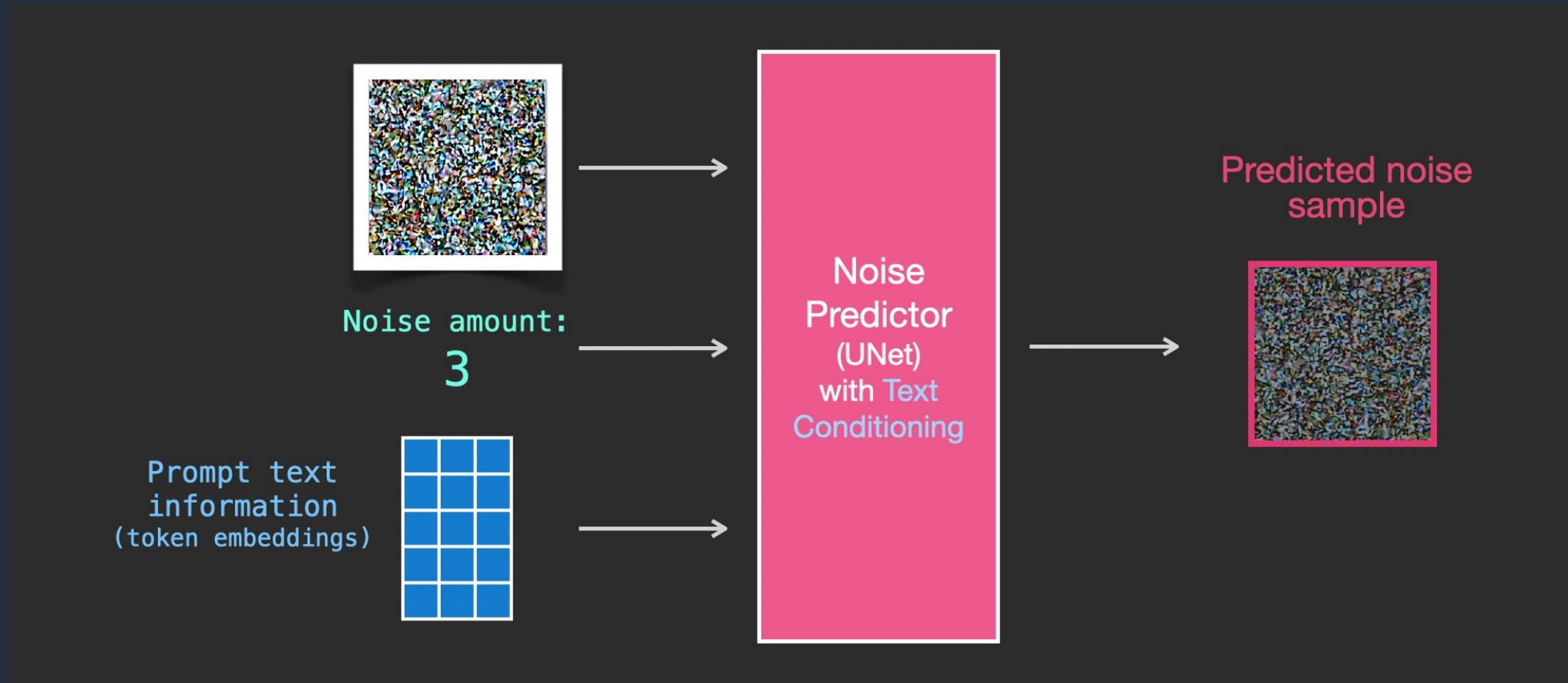
# Building blocks of Diffusion

## Image Generation by Reverse Diffusion (Denoising)



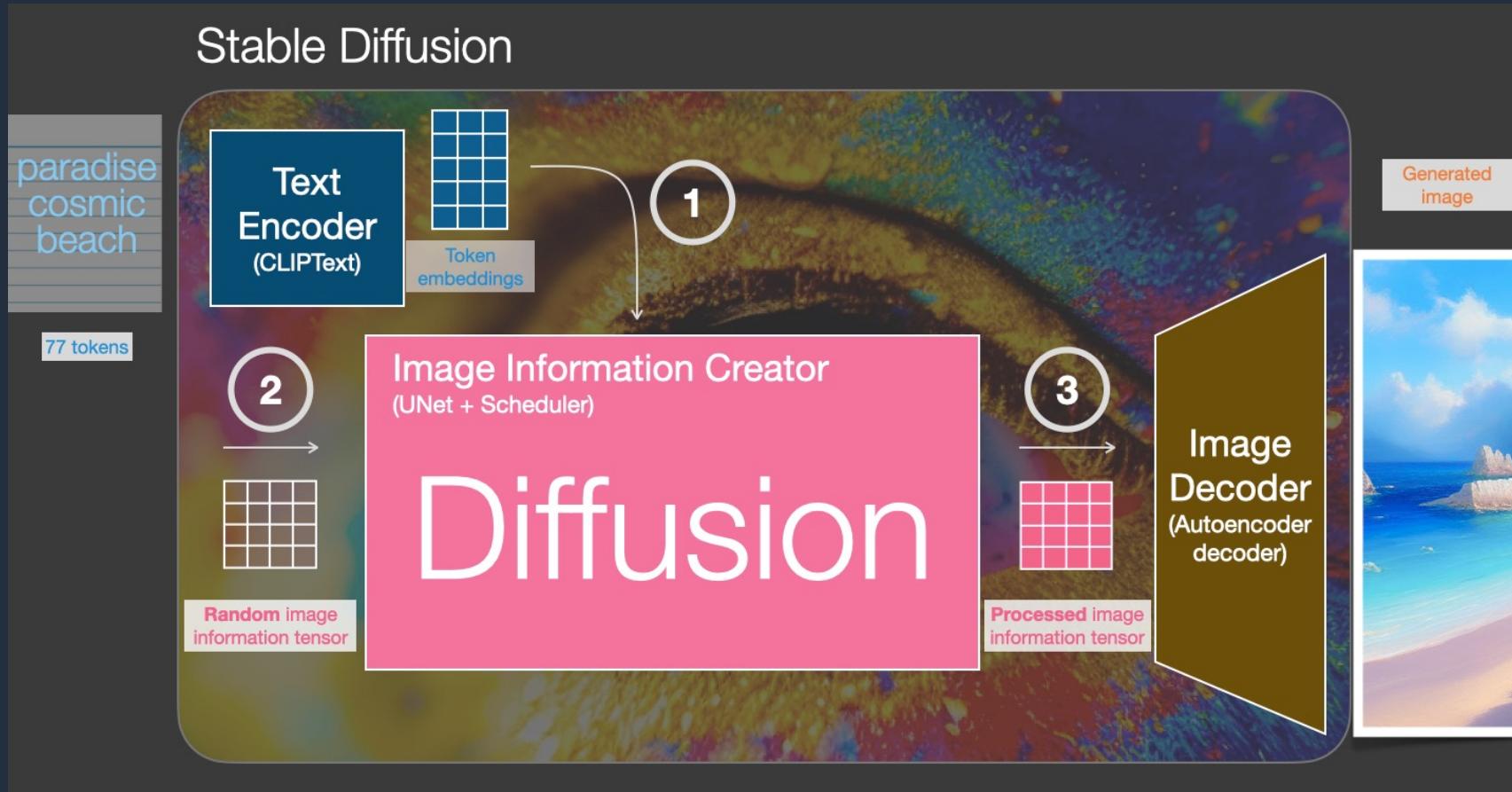
Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Building blocks of Diffusion



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Building blocks of Diffusion



Source: <https://jalammar.github.io/illustrated-stable-diffusion/>

# Lab 3 – Stable Diffusion Deployment & Inference

<https://github.com/aristsakpinis93/generative-ai-immersion-day>

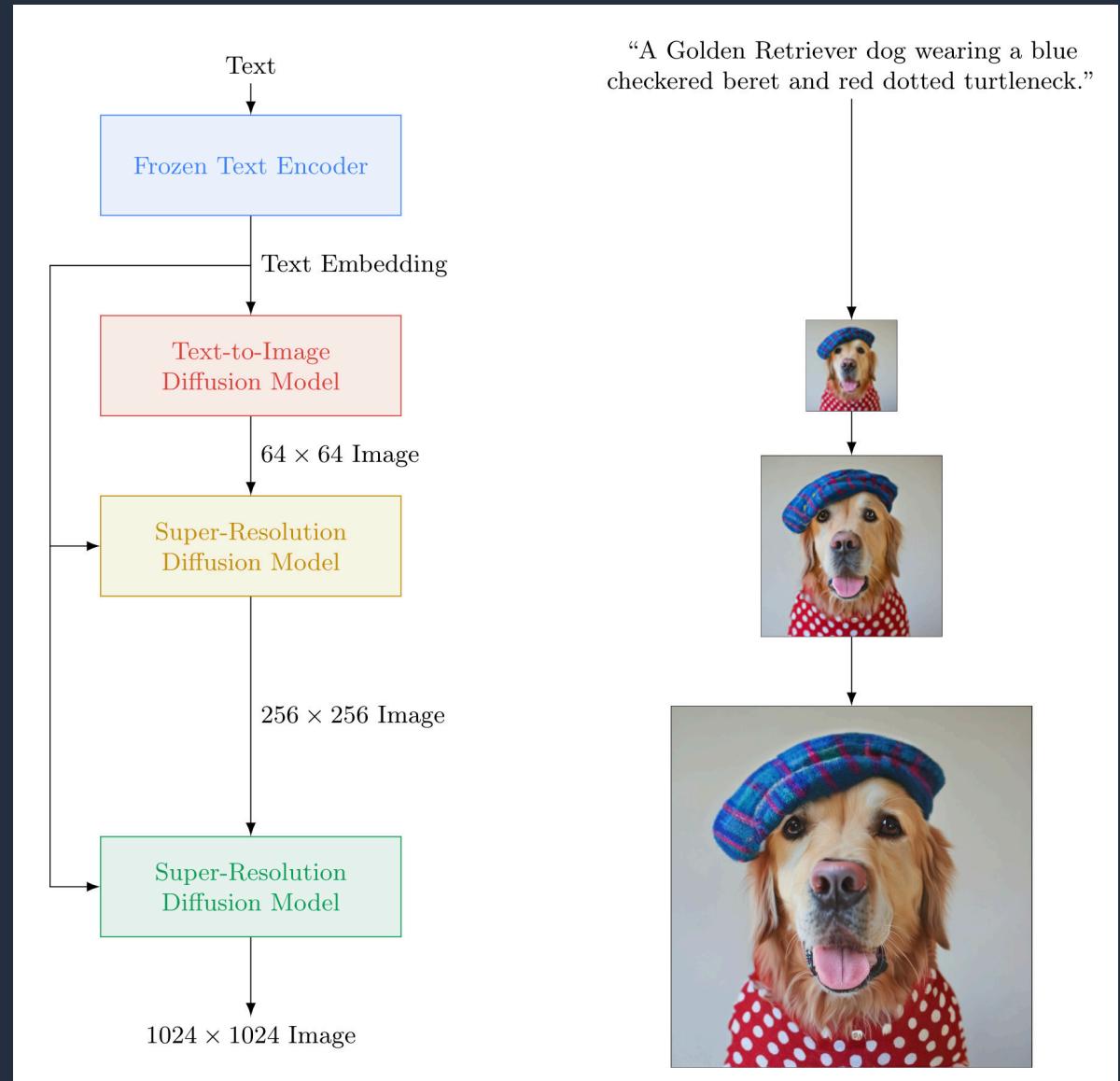
EventHash:



# Other models and related work

# Imagen (Google, 2022)

- Another diffusion model
- Generate at low resolution and use super resolution networks to upscale; doesn't use an autoencoder
- Not available to the public.



# Dall-E 2 (OpenAI, 2022)

Another diffusion model

- Weights are private but there's an inference API

## TEXT DESCRIPTION

An astronaut Teddy bears A bowl of soup

mixing sparkling chemicals  
as mad scientists shopping  
for groceries working on  
new AI research

as a 1990s Saturday morning  
cartoon as digital art in a  
steampunk style

## DALL-E 2



# Midjourney

Heavily stylized diffusion model

- Weights are private but there's public access to inference (no API)



# Runway Gen-2

## Text-to-Video

Mode 02: *Text + Image to Video*  
Generate a video using a driving image and a text prompt

---

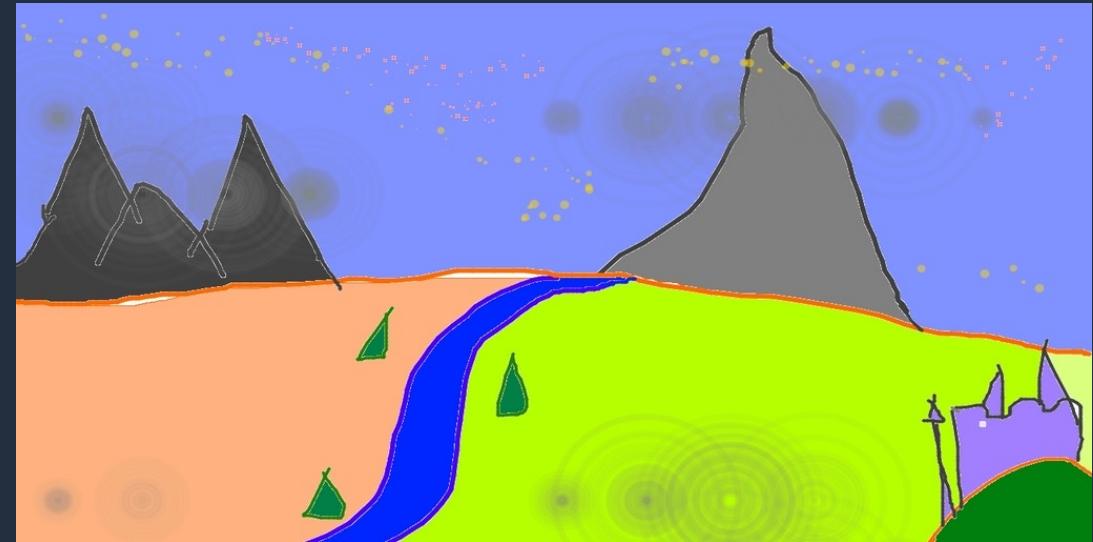
 Input Image

*A low angle shot of a man walking down a street, illuminated by the neon signs of the bars around him.*  
Driving Prompt

 Output Video

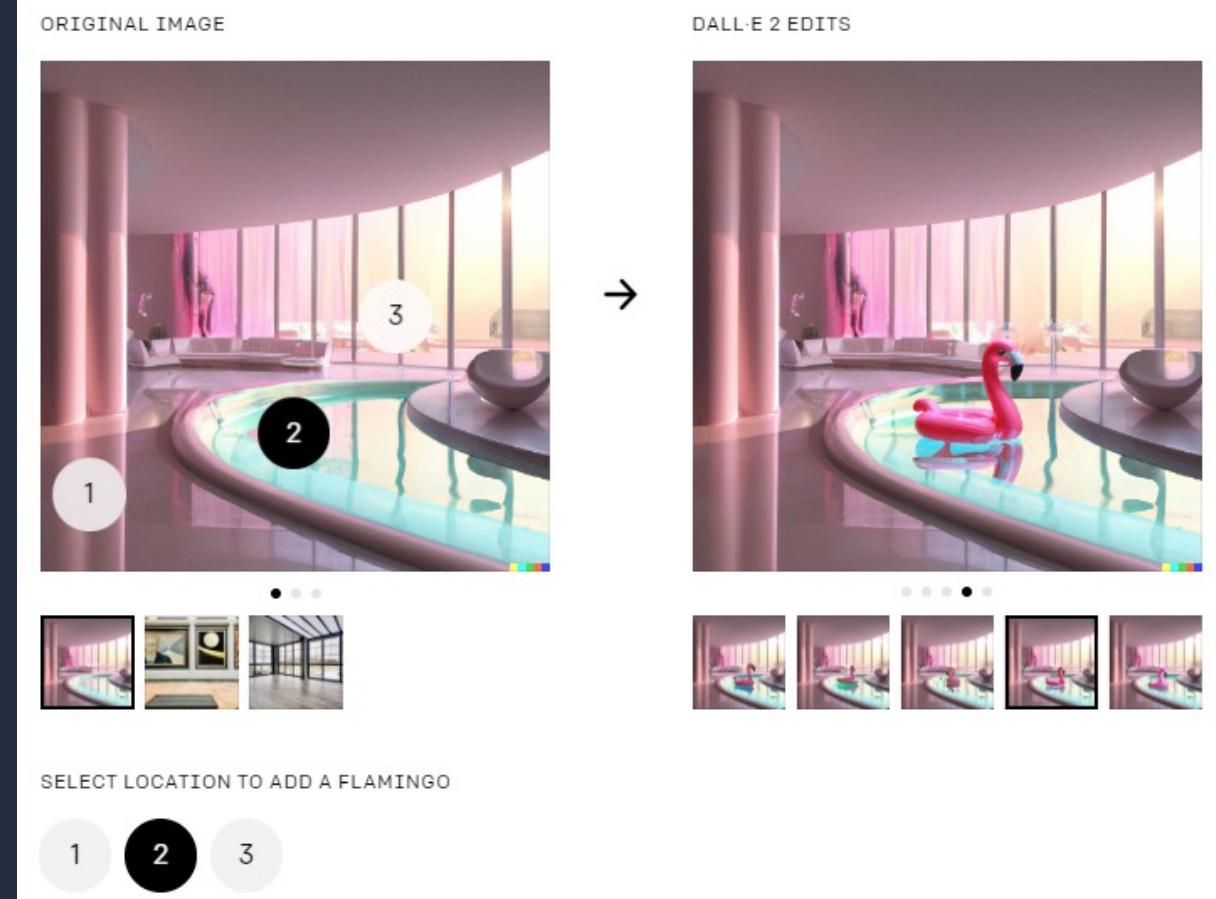
# Image-to-image (img2img)

Add a specified amount of noise to an existing image and start the denoising process



# Inpainting & outpainting

- Masked image-to-image



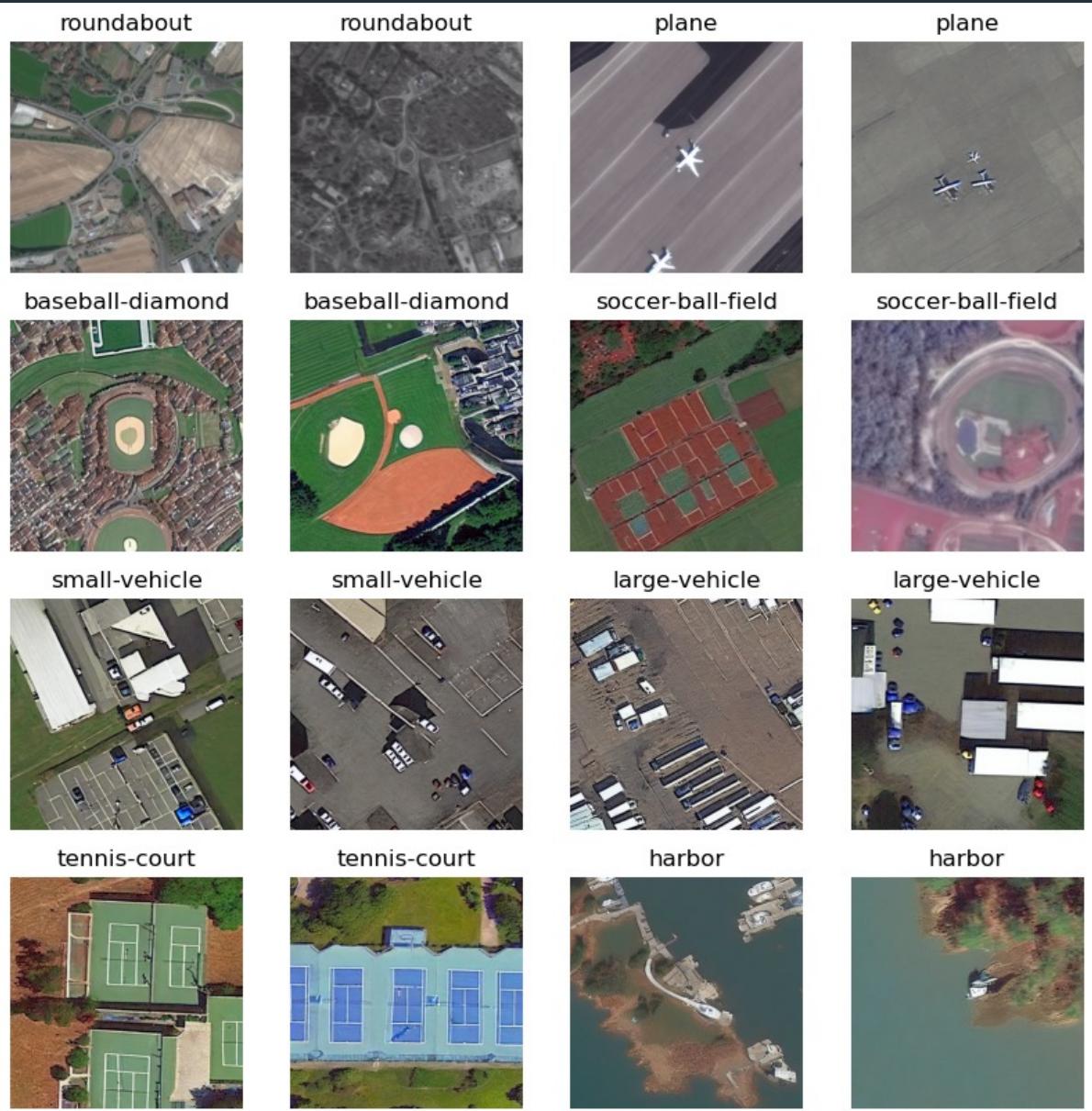
<https://openai.com/dall-e-2/>

# Tuning Stable Diffusion

- Learning a new domain
  - Full scale finetuning
    - Significantly larger data requirement
    - Satellite diffusion model (right) was trained with 2000 image + label pairs

More efficient

- {
- LoRA
  - Hypernetworks

<https://gitlab.aws.dev/wwps-natsec/ai-ml-practice/satellite-diffusion>

"A satellite image showing a {class\_name}"

# Tuning Stable Diffusion

## DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation

[Nataniel Ruiz](#) [Yuanzhen Li](#) [Varun Jampani](#) [Yael Pritch](#) [Michael Rubinstein](#) [Kfir Aberman](#)

Google Research



Input images



in the Acropolis



swimming



sleeping



in a bucket



getting a haircut

*It's like a photo booth, but once the subject is captured, it can be synthesized wherever your dreams take you...*

# AGENDA

Generative AI – What is it and why the hype?

Large Language Models - How the ML works?

Large Language Model Hosting

Large Language Model Finetuning

Visual Foundation Models & Stable Diffusion

Engineering GenAI-powered Applications on AWS

# Successfully building GenAI Applications

## Application Layer

### Built on top

- Proven app development stack
- MLOps ready for Foundation Models

## Model Ecosystem

Easy fine-tuning      Hosting options  
Broadness of choice of Foundation models  
ML Ops

### Vertically Integrated

*"Off the shelf"* applications that can be used with the existing tools and that embed one or more Foundation Models

## Hyperscale Compute

Scalable      Pay as you go      Elastic      Managed      Integrated

Ease of Integration

## Existing IT

## Silicon

Cost Effective      Low Latency      High Throughput      GPU Acceleration

# Amazon SageMaker JumpStart with Foundation Models



## Machine Learning Hub for SageMaker

Browse through ~400 contents including, built-in algorithms with pre-trained models, (New) Foundation Models, solution templates, and example notebooks



## UI as well as API based machine learning

Use the User Interface for single click model deployment or the API for Python SDK based workflow



## (New) Share and collaborate within an organization

Share models and notebooks with others within your organization, and allow them to train with their own data or deploy as-is for inferencing

NEW

Machine Learning / Amazon SageMaker JumpStart

## Getting started with Amazon SageMaker JumpStart

Amazon SageMaker JumpStart is a machine learning (ML) hub that can help you accelerate your ML journey. Explore how you can get started with built-in algorithms with pretrained models from model hubs, pretrained foundation models, and prebuilt solutions to solve common use cases. To get started, see documentation or example notebooks that you can quickly execute.

Reset Filters

Product Type

Text Tasks

- End-to-end Solution
- Text Classification
- Text Embedding
- Text Generation
- Text

Sort By Popularity

FOUNDATION MODEL PREVIEW

Text Generation  
Proprietary Models  
Various Providers



stability.ai

Lighton  
AI for all, everywhere

AI21 labs

co:here

alexa



Stable Diffusion 2

Stabilityai

Model ID: pytorch-textrgen1-alexa20b. AlexaTM 20b is a multitask, multilingual, large-scale sequence-to-sequence (seq2seq) model, trained on a mixture of Common Crawl. It takes a textual description as input and predicts next words in the sequence. This model has BigScience Responsible AI License v1.0. Please read the [terms] ([https://huggingface.co/spaces/bloom\\_1b7](https://huggingface.co/spaces/bloom_1b7))

Fine-tunable

alexa

alexa



Bloom 1b7

Huggingface

Model ID: huggingface-textgeneration-bloom-1b7. This is a Text Generation model built upon a Transformer model from Hugging Face. It takes a text string as input and predicts next words in the sequence. This model has BigScience Responsible AI License v1.0. Please read the [terms] ([https://huggingface.co/spaces/bloom\\_1b7](https://huggingface.co/spaces/bloom_1b7))

Deploy Only

Deploy Only

NEW

# Amazon Bedrock

**The easiest way to build and  
scale generative AI  
applications with FMs**



# Bedrock supports a wide range of foundation models

## FMs from Amazon



Titan Text



Titan  
Embeddings

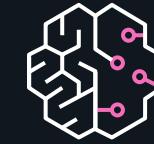
## FMs from AI21 Labs, Anthropic, and Stability AI



Jurassic-2



Claude



Stable  
Diffusion

# Base components and design patterns

- Models
- Prompts
- Memory
- Chains
- Tools
- Agents



# Models

- Language generation models
- Text Embedding Models
- Purpose-fine-tuned models (chat, ...)

```
from langchain.llms.base import LLM

class CustomLLM(LLM):

    @property
    def _llm_type(self) -> str:
        return "custom"

    def _call(
        self,
        prompt: str,
        params: dict
    ) -> str:

        return call_llm_implementation(prompt, params)

llm = CustomLLM()
llm(prompt="How are you?", prompt={"temperature": 0.2})
```

Pre-built implementations available for multiple model providers and hosting options!

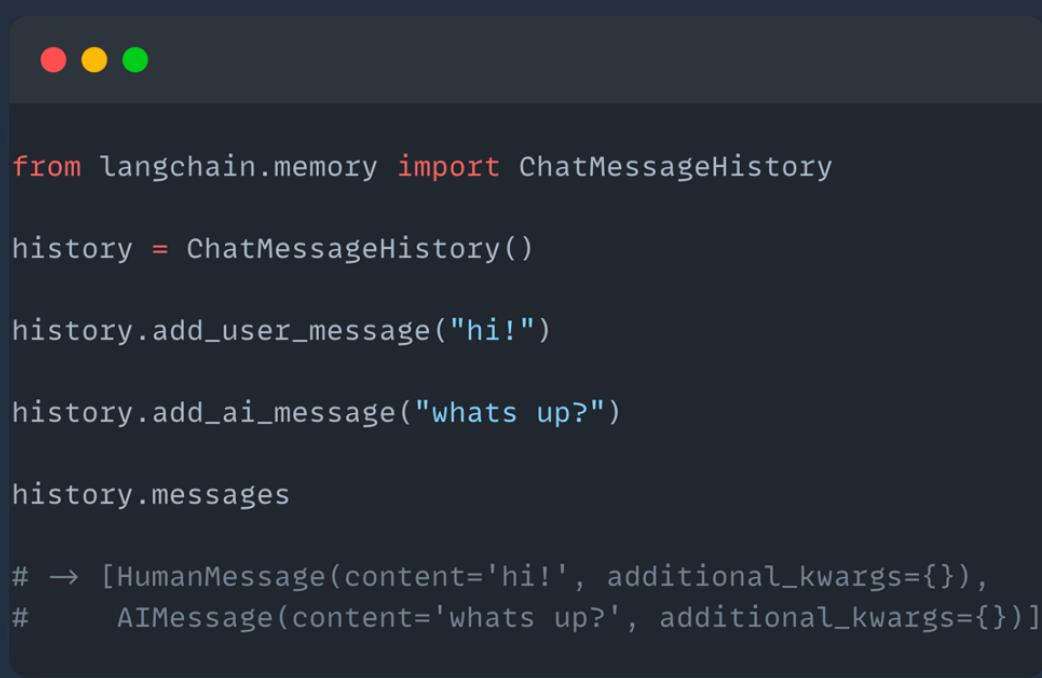
# Prompts: dynamic prompting through PromptTempates

```
from langchain import PromptTemplate

template = """
I want you to act as a naming consultant for new companies.
What is a good name for a company that makes {product}?
"""

prompt = PromptTemplate(
    input_variables=["product"],
    template=template,
)
prompt.format(product="colorful socks")
# → I want you to act as a naming consultant for new companies.
# → What is a good name for a company that makes colorful socks?
```

# Memory: keeping track of conversation history



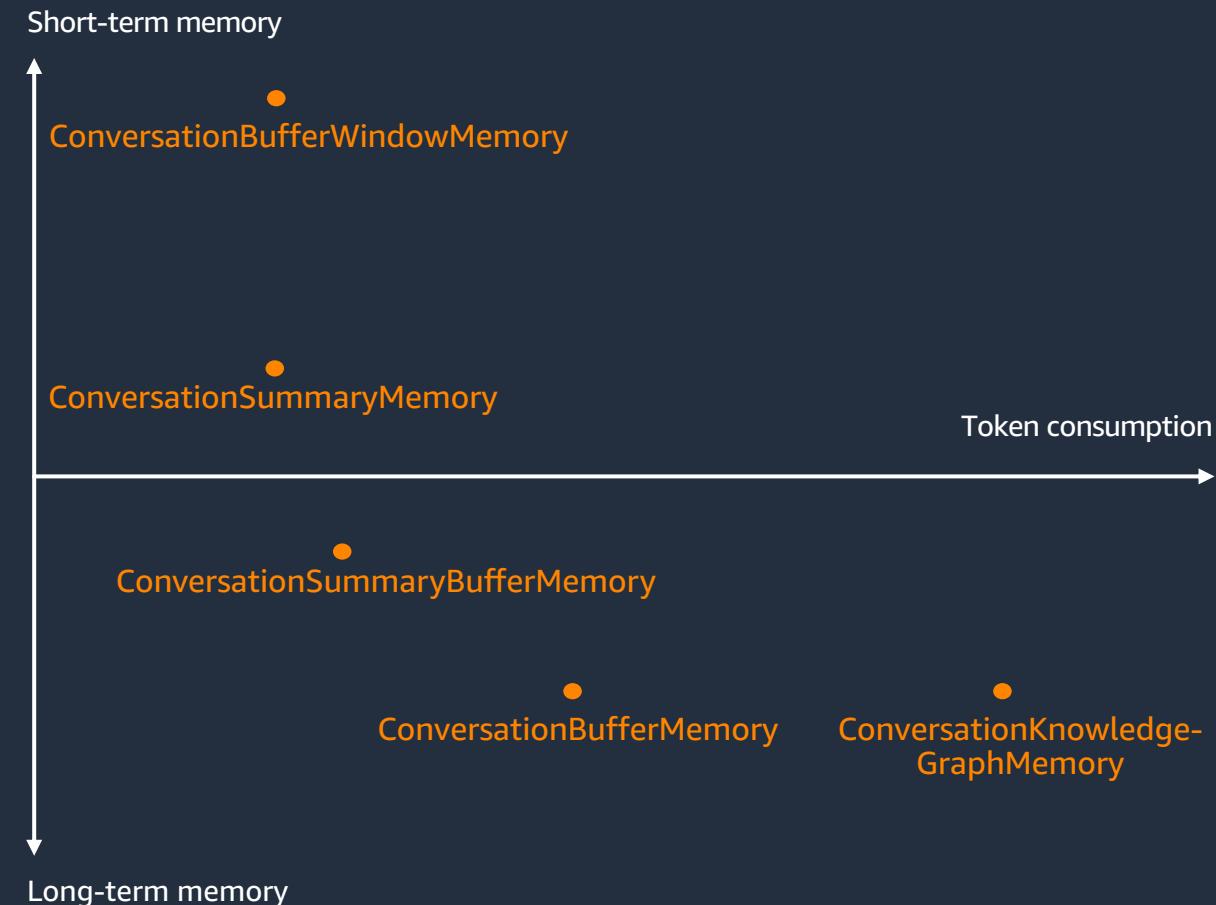
```
from langchain.memory import ChatMessageHistory

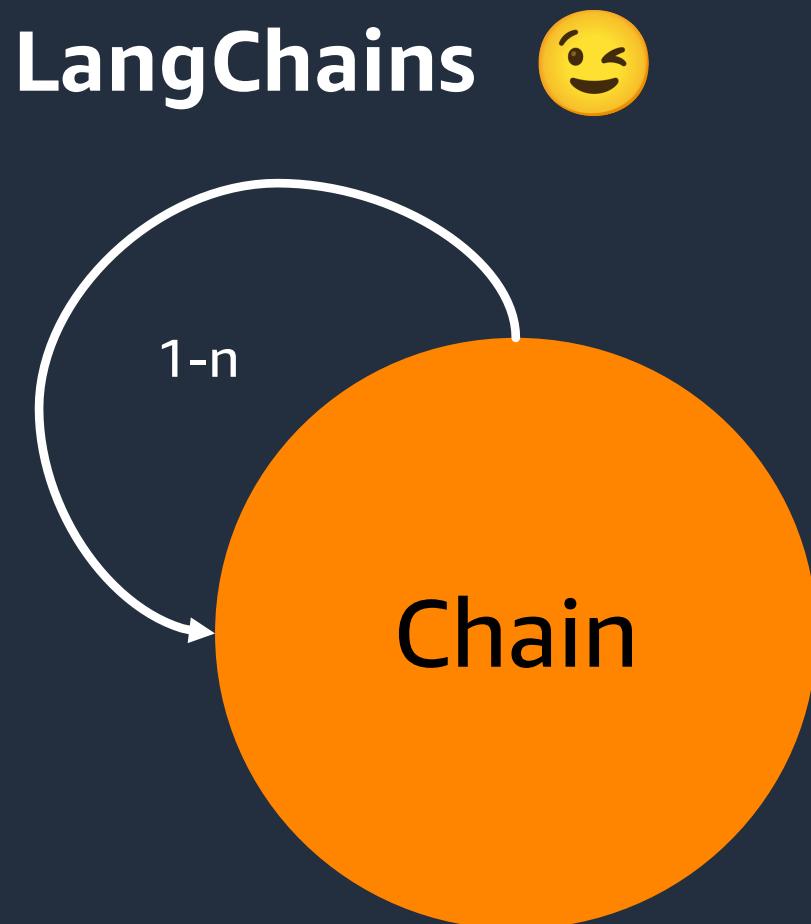
history = ChatMessageHistory()

history.add_user_message("hi!")

history.add_ai_message("whats up?")

history.messages
# → [HumanMessage(content='hi!', additional_kwargs={}),
#     AIMessage(content='whats up?', additional_kwargs={})]
```





## Generic Chains:

- LLMChain = Model + Prompt
- TransformationChain
- SequentialChain
- RouterChain

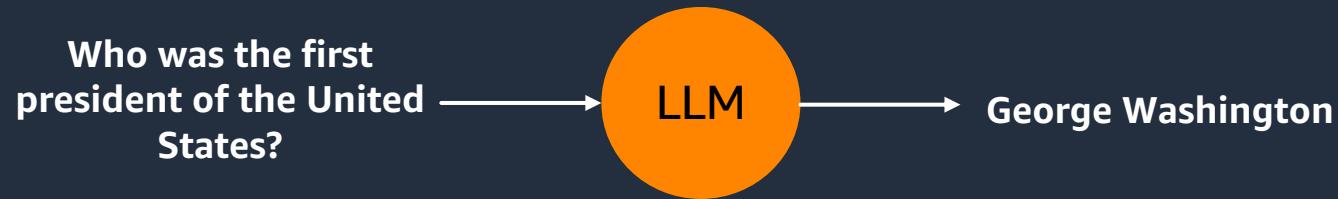
## Index-related Chains:

- Summarization
- StuffDocumentsChain
- QA with sources (RAG)

## Purpose-built Chains:

- LLMMathChain
- APIChain
- Moderation

# LLMChain



Answer the following question: {question}  
If you don't know the answer, just say "I don't know".



Model Node



Decision Node



Business Logic Node

# StuffDocumentsChain



What do all of the  
following articles have in  
common?

Article 1: {article1}

Article 2: {article2}

...



Model Node

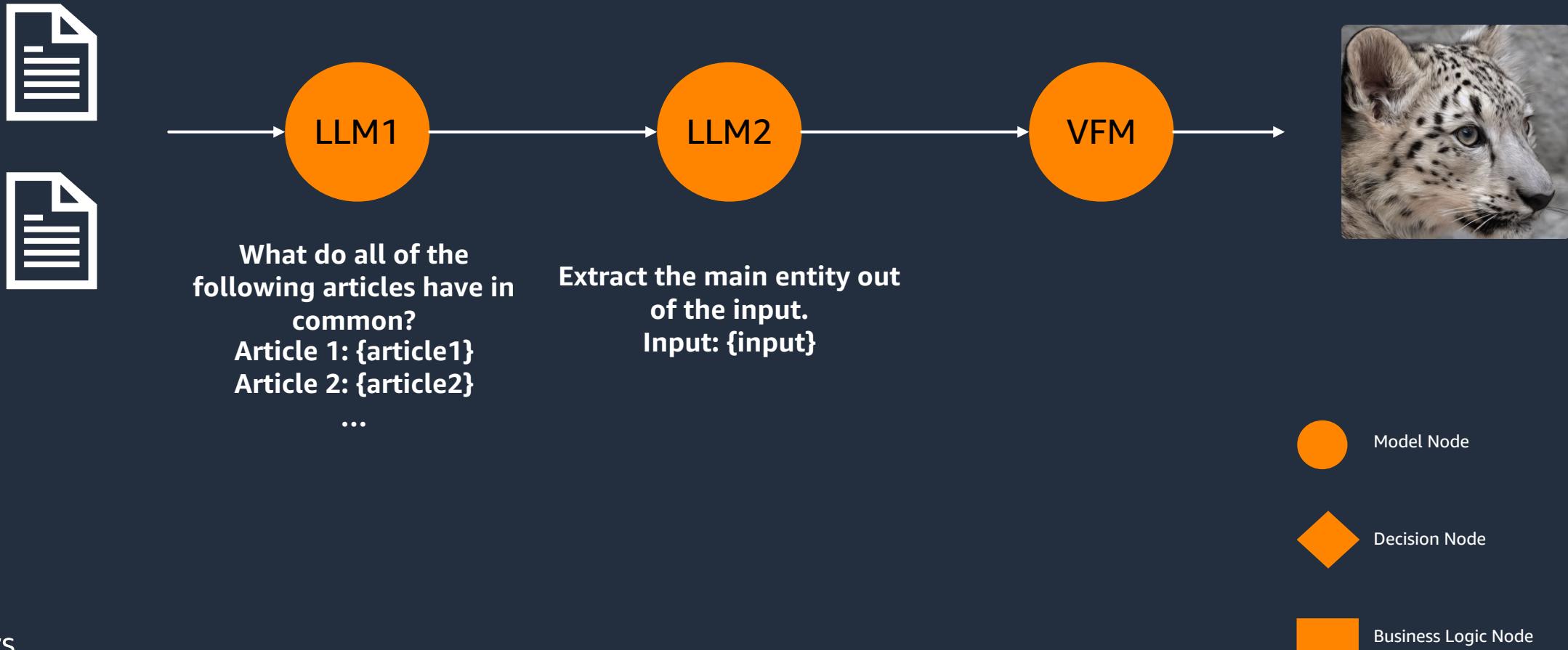


Decision Node

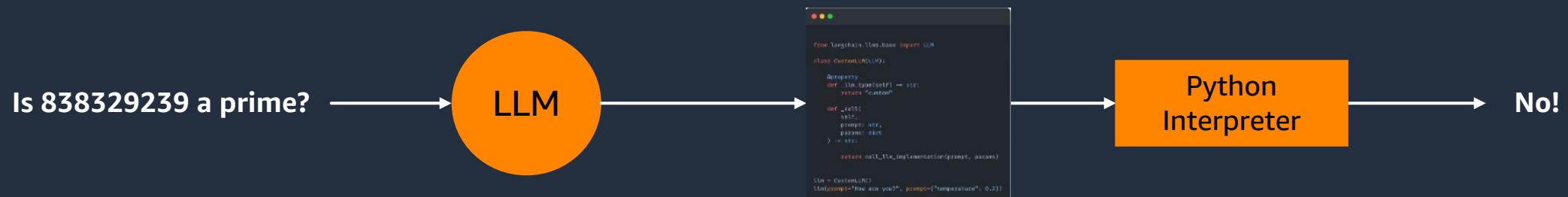


Business Logic Node

# SequentialChain



# TransformationChain



Translate this question  
into Python source code.  
Question: {question}



Model Node

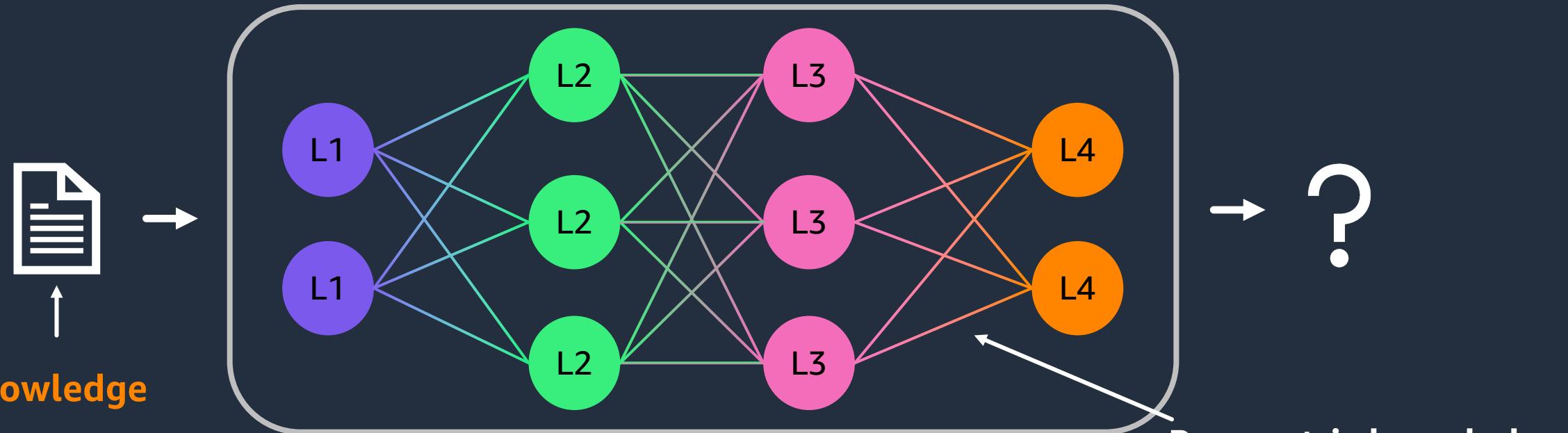


Decision Node



Business Logic Node

# Domain-specific knowledge infusion: forms of knowledge



**Source knowledge**

Adjustable through prompt enrichment (\$)

Dynamic data (live-systems, (semi-)frequently changing knowledge bases, ...)

Minimizes risk of hallucination, adds traceability of results

**Parametric knowledge**

Adjustable through fine-tuning (\$\$\$)

Static data (language foundations , domain-specific vocabulary, writing style, ...)

# Domain-specific knowledge infusion through Tools

*A Tool is “a function that performs a specific duty. (...) The interface for a tool is currently a function that is expected to have a string as an input, with a string as an output.”*

*“These tools can be generic utilities (...), other chains, or even other agents.”*

```
from langchain.tools import BaseTool

class CustomSearchTool(BaseTool):
    name = "custom_tool"
    description = "description of use case of this tool"

    def _run(self, query: str) -> str:
        """Use the tool."""
        result = tool_implementation(query)
        return result
```

```
from langchain.tools import tool

@tool(name="custom_tool", description="...")
def search_api(query: str) -> str:
    """Use the tool."""
    result = tool_implementation(query)
    return result
```

# Tools: RAG

---

**Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks**

---

Patrick Lewis<sup>†‡</sup>, Ethan Perez<sup>\*</sup>,

Aleksandra Piktus<sup>†</sup>, Fabio Petroni<sup>†</sup>, Vladimir Karpukhin<sup>†</sup>, Naman Goyal<sup>†</sup>, Heinrich Küttler<sup>†</sup>,

Mike Lewis<sup>†</sup>, Wen-tau Yih<sup>†</sup>, Tim Rocktäschel<sup>†‡</sup>, Sebastian Riedel<sup>†‡</sup>, Douwe Kiela<sup>†</sup>

<sup>†</sup>Facebook AI Research; <sup>‡</sup>University College London; <sup>\*</sup>New York University;  
plewis@fb.com

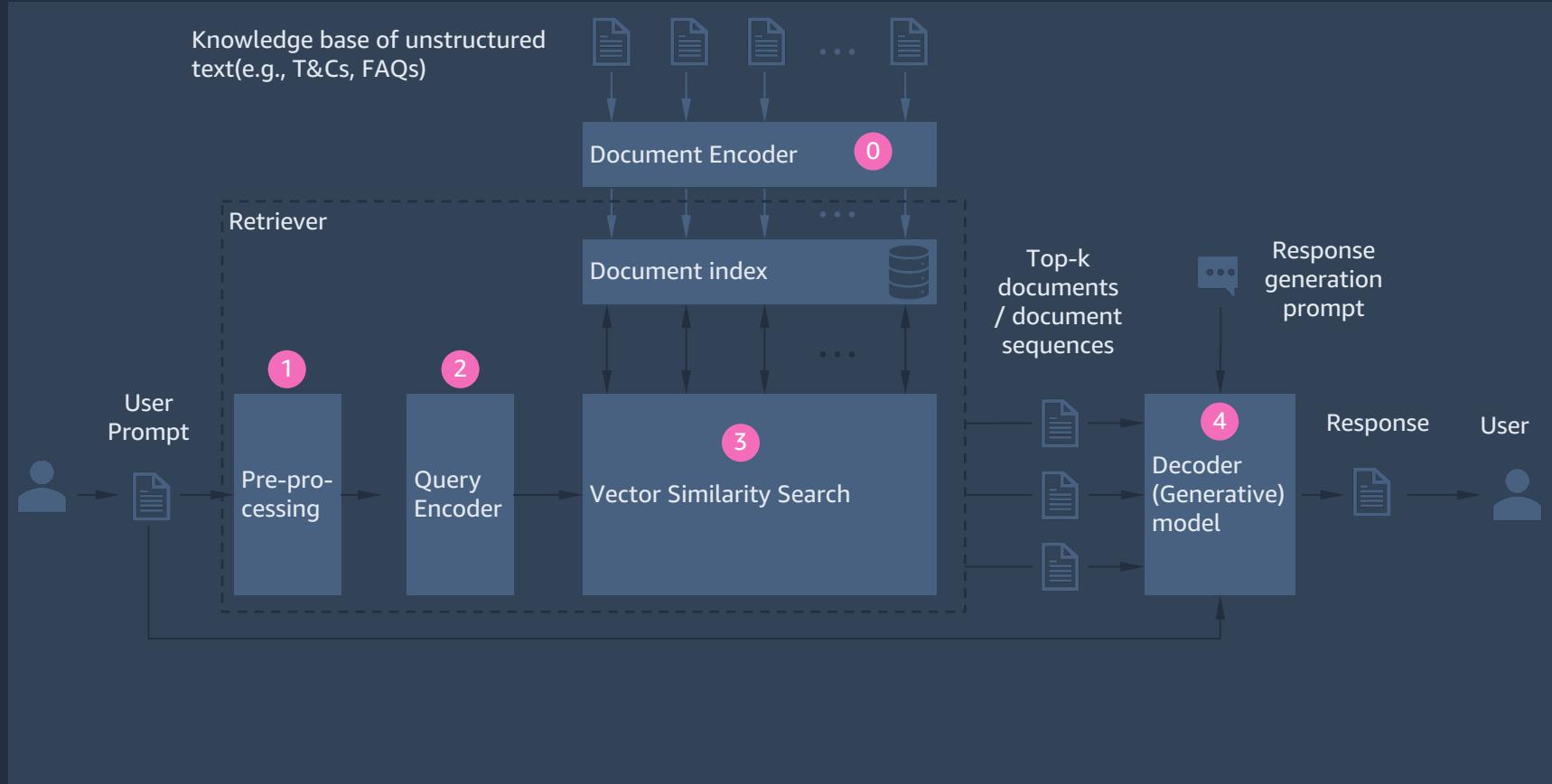
**Abstract**

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric mem-

- Retrieval-augmented generation
- Access to knowledge base of unstructured text
- Implemented through two-step chain powered by two different LLMs

Source: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, Patrick Lewis et al., 2021,  
<https://arxiv.org/pdf/2005.11401.pdf>

# Tools: RAG



## Concept

- 0 Knowledge documents / document sequences are encoded and ingested into a vector database.
- 1 Customer e-mail query is pre-processed and/or tokenized
- 2 Tokenized input query is encoded
- 3 Encoded query is used to retrieve most similar text passages in document index using vector similarity search (e.g., Mixed Inner Product Search)
- 4 Top-k retrieved documents/text passages in combination with original customer e-mail query and e-mail generation prompt are fed into Generator model (Encoder-Decoder) to generate response e-mail

Source: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, Patrick Lewis et al., 2021, <https://arxiv.org/pdf/2005.11401.pdf>

# Tools: RAG

## Document embedding & vectorstore ingestion

```
from langchain.document_loaders import WebBaseLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.vectorstores import OpenSearchVectorSearch

# load documents
loader = WebBaseLoader("https://aws.amazon.com/s3/faqs/")
documents = loader.load()

# split & tokenize
text_splitter = CharacterTextSplitter.from_tiktoken_encoder(chunk_size=100, chunk_overlap=10)
texts = text_splitter.split_documents(documents)

# initialize embeddings model wrapper
embeddings = CustomLLM()

# embed documents into vectorstore
docsearch = OpenSearchVectorSearch.from_documents(
    docs,
    embeddings,
    opensearch_url="http://localhost:9200"
)
```

## Retrievers

```
query = "Which kind of data can I store in Amazon S3?"
docs = docsearch.similarity_search(query, k=3)
print(docs[0].page_content)
# → [{page_content: (...), (...),
#       {page_content: (...), (...),
#       ...
#       }

# wrap into retriever construct
retriever=docsearch.as_retriever()
docs = retriever.get_relevant_documents(query)
```

## RetrievalQAChain

```
from langchain.chains import RetrievalQA

# generative LLM
llm = CustomLLM()

# define RetrievalQA chain
qa = RetrievalQA.from_chain_type(llm, chain_type="stuff", retriever=retriever)

# run chain
res = qa.run(query)
print(res)
# → "You can store virtually any kind of data in any format in Amazon S3."
```



# Tools: Requests



```
from langchain.agents import load_tools

requests_tools = load_tools(["requests_all"])
print(requests_tools)
# → [RequestsGetTool(name='requests_get', description='A portal to the internet. Use this when you need to
#       get specific content from a website. Input should be a url (i.e. https://www.google.com). The output
#       will be the text response of the GET request.', (...)),
#       RequestsPostTool(name='requests_post', (...)),
#       RequestsPatchTool(name='requests_patch', (...)),
#       RequestsPutTool(name='requests_put', (...)),
#       RequestsDeleteTool(name='requests_delete', (...))]

requests_get = requests_tools[0]
requests_get.run("https://aws.amazon.com/s3/faqs/")
```

# Tools: Google Search

```
from langchain.tools import Tool
from langchain.utilities import GoogleSearchAPIWrapper

search = GoogleSearchAPIWrapper()

tool = Tool(
    name = "Google Search",
    description="Search Google for recent results.",
    func=search.run
)

tool.run("Who won the 2020 UEFA Champions League Final?")
# → FC Bayern München
```

# Tools: Python REPL

```
from langchain.agents import Tool
from langchain.utilities import PythonREPL

python_repl = PythonREPL()

python_repl.run("print(1+1)")
# → 2

# You can create the tool to pass to an agent
repl_tool = Tool(
    name="python_repl",
    description="""
        A Python shell. Use this to execute python commands.
        Input should be a valid python command. If you want to see
        the output of a value, you should print it out with `print(...)`.
    """,
    func=python_repl.run
)

repl_tool.run("print(1+1)")
# → 2
```

# Agents: MRKL

## MRKL Systems

A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, Moshe Tenenholz

AI21 Labs

May 3, 2022

### Abstract

Huge language models (LMs) have ushered in a new era for AI, serving as a gateway to natural-language-based knowledge tasks. Although an essential element of modern AI, LMs are also inherently limited in a number of ways. We discuss these limitations and how they can be avoided by adopting a systems approach. Conceptualizing the challenge as one that involves knowledge and reasoning in addition to linguistic processing, we define a flexible architecture with multiple neural models, complemented by discrete knowledge and reasoning modules. We describe this neuro-symbolic architecture, dubbed the Modular Reasoning, Knowledge and Language (MRKL, pronounced “miracle”) system, some of the technical challenges in implementing it, and Jurassic X: AI21 Labs’ MRKL system implementation.

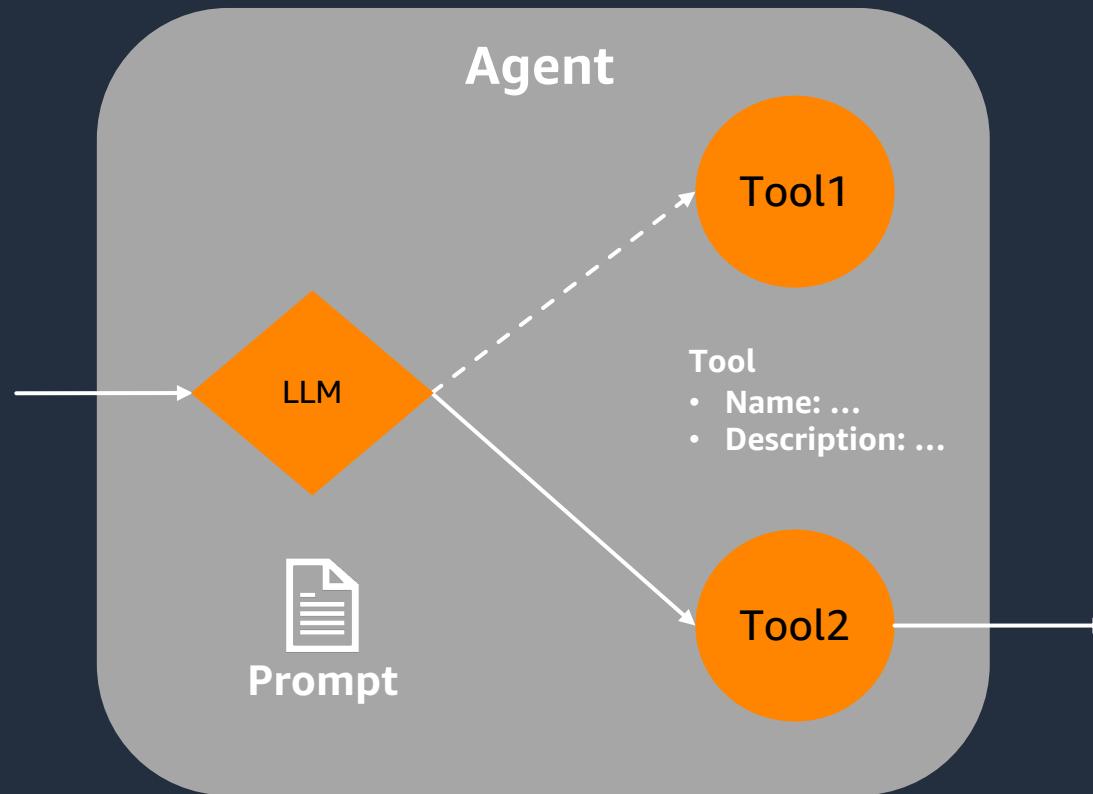
Source: MRKL Systems – A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning, Ehud Karpas et al, 2022, <https://arxiv.org/pdf/2205.00445.pdf>

**Idea:** System routes between 1-n out of multiple tools, which can be *neural* or *symbolic*.

### Benefits:

- Proprietary knowledge
- Up-to-date information
- Interpretability
- Composability
- Robust extensibility
- Safe fallback

# Agents: basic components



# Agents: ReAct

## REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao<sup>\*1</sup>, Jeffrey Zhao<sup>2</sup>, Dian Yu<sup>2</sup>, Nan Du<sup>2</sup>, Izhak Shafran<sup>2</sup>, Karthik Narasimhan<sup>1</sup>, Yuan Cao<sup>2</sup>

<sup>1</sup>Department of Computer Science, Princeton University

<sup>2</sup>Google Research, Brain team

<sup>1</sup>{shunyuy, karthikn}@princeton.edu

<sup>2</sup>{jeffreyzhao, dianyu, dunan, izhak, yuancao}@google.com

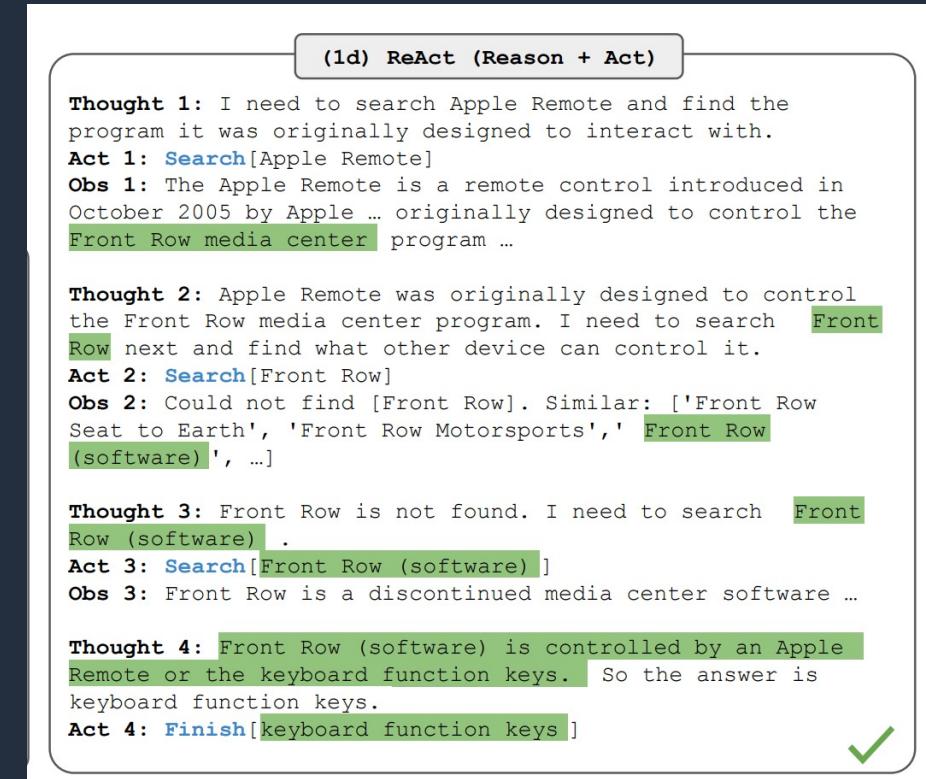
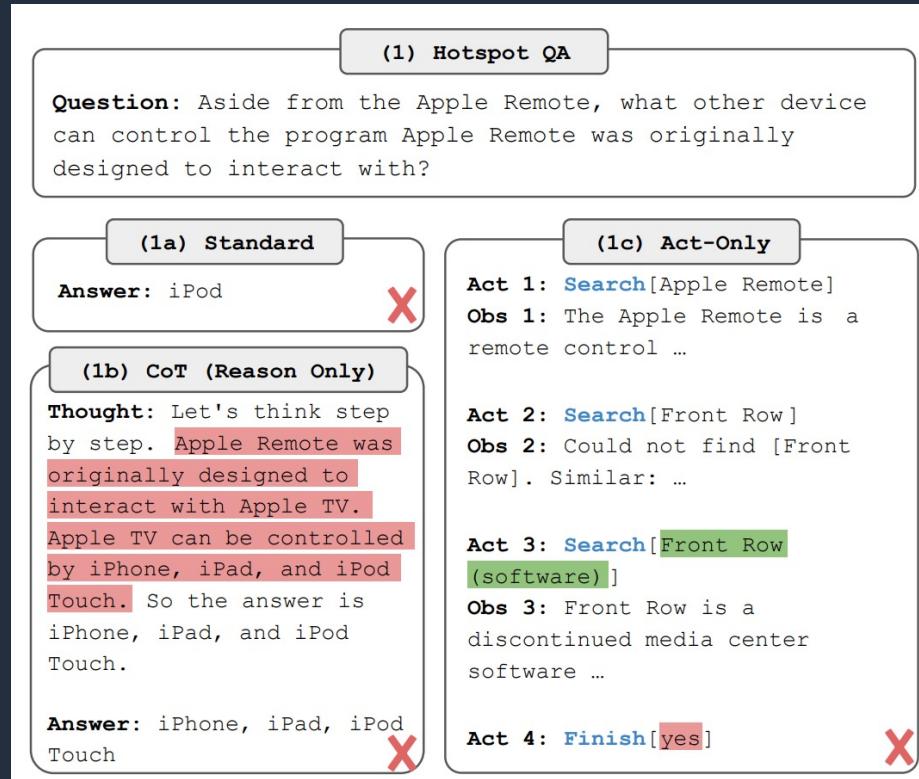
### ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving

- Logical and modular reasoning of GenAI-powered systems (chain-of-thought)
- Execution of task-specific actions (action plan generation)
- Implemented through chain of recursive steps against a powerful LLM

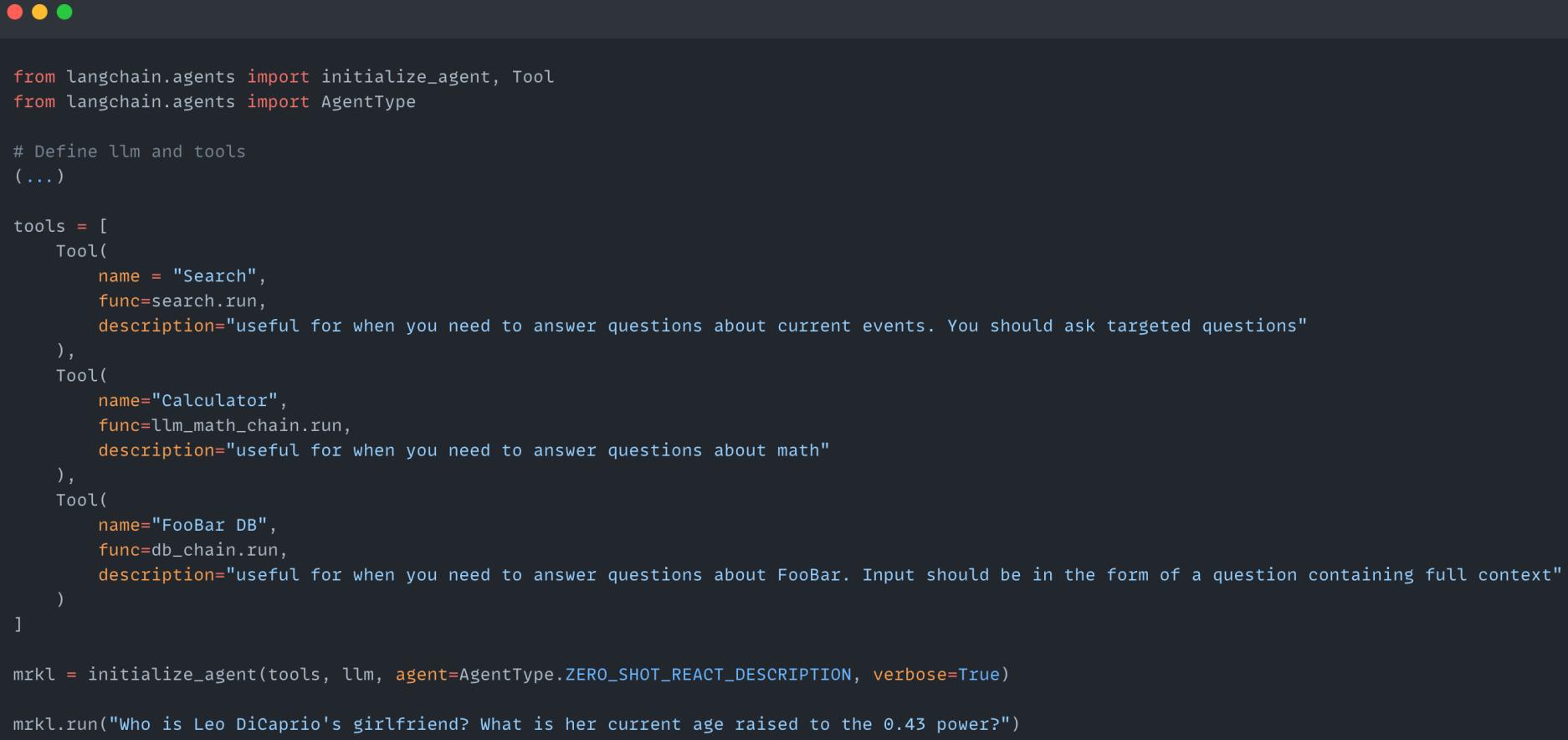
Source: ReAct: Synergizing reasoning and acting in language models,  
Shunyu Yao et al, 2023, <https://arxiv.org/pdf/2210.03629.pdf>

# Agents: ReAct



Source: ReAct: Synergizing reasoning and acting in language models,  
Shunyu Yao et al, 2023, <https://arxiv.org/pdf/2210.03629.pdf>

# Agents: MRKL agent implementation



```
from langchain.agents import initialize_agent, Tool
from langchain.agents import AgentType

# Define llm and tools
(...)

tools = [
    Tool(
        name = "Search",
        func=search.run,
        description="useful for when you need to answer questions about current events. You should ask targeted questions"
    ),
    Tool(
        name="Calculator",
        func=llm_math_chain.run,
        description="useful for when you need to answer questions about math"
    ),
    Tool(
        name="FooBar DB",
        func=db_chain.run,
        description="useful for when you need to answer questions about FooBar. Input should be in the form of a question containing full context"
    )
]

mrkl = initialize_agent(tools, llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)

mrkl.run("Who is Leo DiCaprio's girlfriend? What is her current age raised to the 0.43 power?")
```

# Agents: MRKL agent implementation

```
> Entering new AgentExecutor chain...
I need to find out who Leo DiCaprio's girlfriend is and then calculate her age raised to the 0.43 power.
Action: Search
Action Input: "Who is Leo DiCaprio's girlfriend?"
Observation: DiCaprio met actor Camila Morrone in December 2017, when she was 20 and he was 43. They were spotted at Coachella and went on multiple vacations together. Some reports suggested that DiCaprio was ready to ask Morrone to marry him. The couple made their red carpet debut at the 2020 Academy Awards.
Thought: I need to calculate Camila Morrone's age raised to the 0.43 power.
Action: Calculator
Action Input: 21^0.43

> Entering new LLMMathChain chain...
21^0.43
```text
21**0.43
```
...numexpr.evaluate("21**0.43")...

Answer: 3.7030049853137306
> Finished chain.

Observation: Answer: 3.7030049853137306
Thought: I now know the final answer.
Final Answer: Camila Morrone is Leo DiCaprio's girlfriend and her current age raised to the 0.43 power is 3.7030049853137306.

> Finished chain.
```

# How to get started on AWS?



# Langchain ❤️ AWS: Orchestration Layer



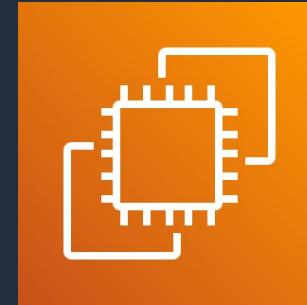
|



AWS Lambda



Amazon ECS



Amazon EC2

...

# Langchain ❤️ AWS: Models

```
from typing import Dict
from langchain import SagemakerEndpoint
from langchain.llms.sagemaker_endpoint import LLMContentHandler
import json

class ContentHandler(LLMContentHandler):
    content_type = "application/json"
    accepts = "application/json"

    def transform_input(self, prompt: str, model_kwargs: Dict) → bytes:
        # example transform
        input_str = json.dumps({prompt: prompt, **model_kwargs})
        return input_str.encode('utf-8')

    def transform_output(self, output: bytes) → str:
        # example transform
        response_json = json.loads(output.read().decode("utf-8"))
        return response_json[0]["generated_text"]

content_handler = ContentHandler()

llm=SagemakerEndpoint(
    endpoint_name="endpoint-name",
    region_name="us-east-1",
    model_kwargs={"temperature":1e-10},
    content_handler=content_handler
)
```

```
from typing import Dict, List
from langchain.embeddings import SagemakerEndpointEmbeddings
from langchain.llms.sagemaker_endpoint import ContentHandlerBase
import json

class ContentHandler(ContentHandlerBase):
    content_type = "application/json"
    accepts = "application/json"

    def transform_input(self, inputs: list[str], model_kwargs: Dict) → bytes:
        # example transform
        input_str = json.dumps({"inputs": inputs, **model_kwargs})
        return input_str.encode('utf-8')

    def transform_output(self, output: bytes) → List[List[float]]:
        # example transform
        response_json = json.loads(output.read().decode("utf-8"))
        return response_json["vectors"]

content_handler = ContentHandler()

embeddings = SagemakerEndpointEmbeddings(
    endpoint_name="endpoint-name",
    region_name="us-east-1",
    content_handler=content_handler
)
```

```
from langchain.llms.bedrock import Bedrock
from langchain.embeddings import BedrockEmbeddings

llm = Bedrock(model_id="amazon.titan-tg1-large")
embeddings = BedrockEmbeddings(model_id="amazon.titan-e1t-medium")
```

# Langchain ❤️ AWS: RAG



Amazon  
SageMaker  
Processing

...



Amazon Kendra



Amazon OpenSearch  
Service



Amazon RDS  
Postgres



AWS Marketplace

...

Preprocessing

Document store & retrieval

# Langchain ❤️ AWS: DynamoDBChatMessageHistory

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  MemoryTable:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: MemoryTable
      AttributeDefinitions:
        - AttributeName: SessionId
          AttributeType: S
      KeySchema:
        - AttributeName: SessionId
          KeyType: HASH
    BillingMode: PAY_PER_REQUEST
```

```
from langchain.memory.chat_message_histories import DynamoDBChatMessageHistory

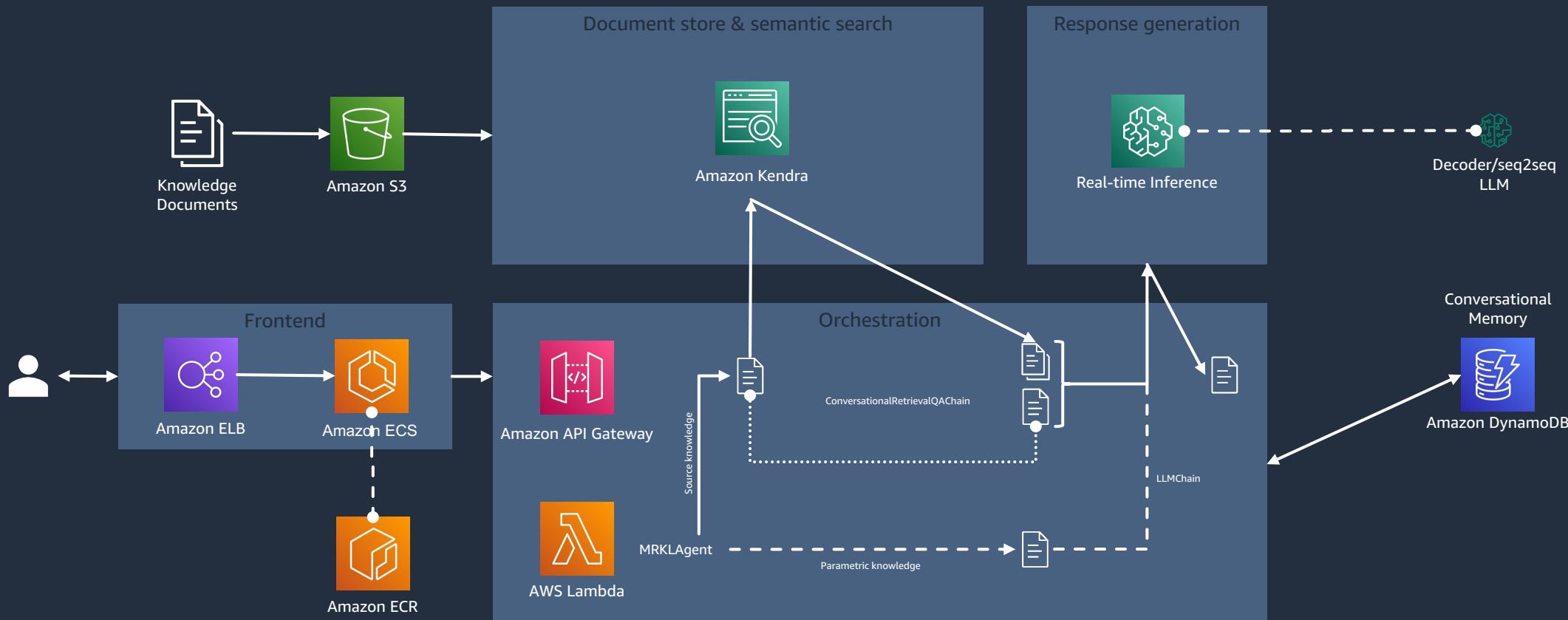
history = DynamoDBChatMessageHistory(table_name="MemoryTable", session_id="0")

history.add_user_message("hi!")

history.add_ai_message("whats up?")
history.messages

# → [HumanMessage(content='hi!', additional_kwargs={}, example=False),
#     AIMessage(content='whats up?', additional_kwargs={}, example=False)]
```

# Langchain ❤️ AWS: E2E Architecture



# Overview of FLAN-T5-XXL model

- Originating out of the T5 transformer architecture
- Largest version of FLAN-T5 model family
- Fine-tuned LAnguage Net (FLAN) is a technique for instruction tuning to learn how to solve natural language processing tasks in general
- Instruction-fine-tuned encoder-decoder transformer model based on [this paper](#)
- Fine-tuned on several datasets (natural instructions only one of them)
- Available through SageMaker JumpStart (FP32, FP16, INT8)

| Hyperparameters | Value      |
|-----------------|------------|
| Parameters      | 11 billion |
| Layers          | 24         |



# Lab 4 – LLM-powered chatbot with RAG-capabilities and short-term memory

<https://github.com/aristsakpinis93/generative-ai-immersion-day>

EventHash:



# Q&A + CSAT – Please provide feedback!



# Thank you!

Aris Tsakpinis

[tsaris@amazon.de](mailto:tsaris@amazon.de)