Leveraging Large Language Models for Advanced AI Applications

Conf42 Observability 2024

Satya Kale, Sr. SDE Amazon

Gayathri Shivaraj, Sr. PM Amazon

Agenda

- Introduction to Large Language Models
- Understanding LLM Architecture
- Methods for Leveraging LLMs
- Limitations of LLMs
- Real world success stories

Understanding Large Language Models:

1. **What are Large Language Models (LLMs)?**

   Large Language Models (LLMs) are advanced AI models trained on extensive datasets to understand and generate human-like language.

2. **Key Components of LLMs:**
   - **Transformer Architecture**
   - **Pre-trained Parameters**
   - **Fine-tuning**

3. **Capabilities of LLMs:**
   - ***Content Generation and Comprehension:*** Text Generation, Question Answering
   - ***Language Processing:*** Language Translation, Summarization
   - ***Analysis and Recognition:*** Sentiment Analysis, Text Classification, Named Entity Recognition (NER)

4. **Applications Across Industries:**

   Software Development: LLMs facilitate code summarization, natural language code search, and automated documentation generation, enhancing developer productivity and code understanding.

   Learning: LLMs can serve as educational tools for learning programming languages, providing personalized feedback and tutoring to aspiring developers. They can also support the creation of interactive coding exercises and adaptive learning platforms.

Transformer Architecture

- The Transformers architecture, introduced in the paper "Attention is All You Need" by Vaswani et al. (2017), revolutionized natural language processing (NLP)

•Self-Attention Mechanism

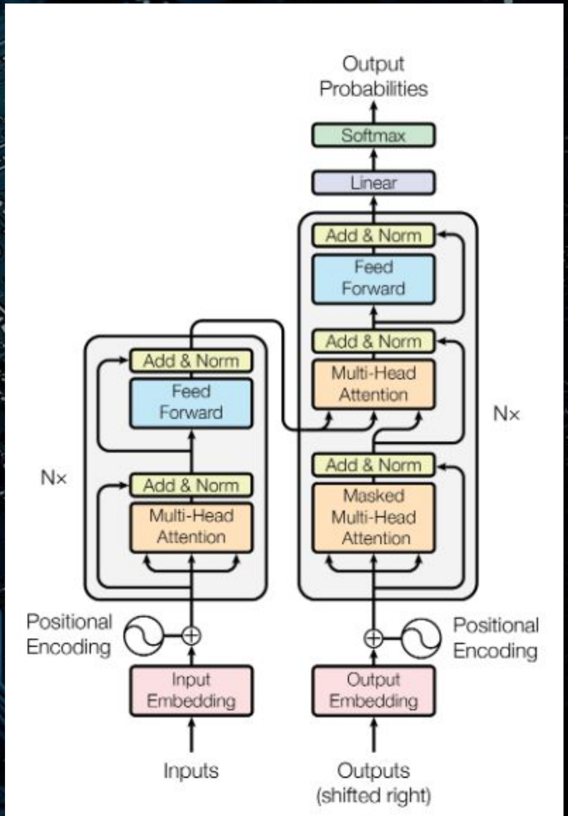
•Positional Encoding

•Feedforward Neural Networks

•Encoder and Decoder

•Multi-Head Attention

•Layer Normalization and Residual Connections



Transformers Architecture [https://doi.org/10.48550/arXiv.1706.03762]

Pre-Trained Parameters

- The model is trained in a self-supervised manner on a large corpus to predict the next tokens given the input
- Learned weights and biases in the model obtained during pre-training on large datasets. Serve as the initial knowledge base for downstream tasks
- Components of Pre-trained parameters:
  - **Word Embeddings**
  - **Transformer Layers**
  - **Output Layer Parameters**

Fine Tuning

- Hyper parameterization
- One-shot/Few-shot learning
- Domain adaptation

Applications that benefit from fine-tuning are Sentiment analysis, Chatbots, Summarization

How Can I Leverage Large Language Models (LLMs)?

- **For Developers**
  - **Integration in Applications**
    - API and SDK based offerings
    - Pre-trained foundational model hosting
    - Custom/Fine tuned model hosting
  - **Quick Playgrounds**
    - AWS Bedrock
    - Quick UIs with Gradio
- **For Non Developers**

  No-Code/Low-Code Platforms – AWS Bedrock, Chat GPT, Claude AI

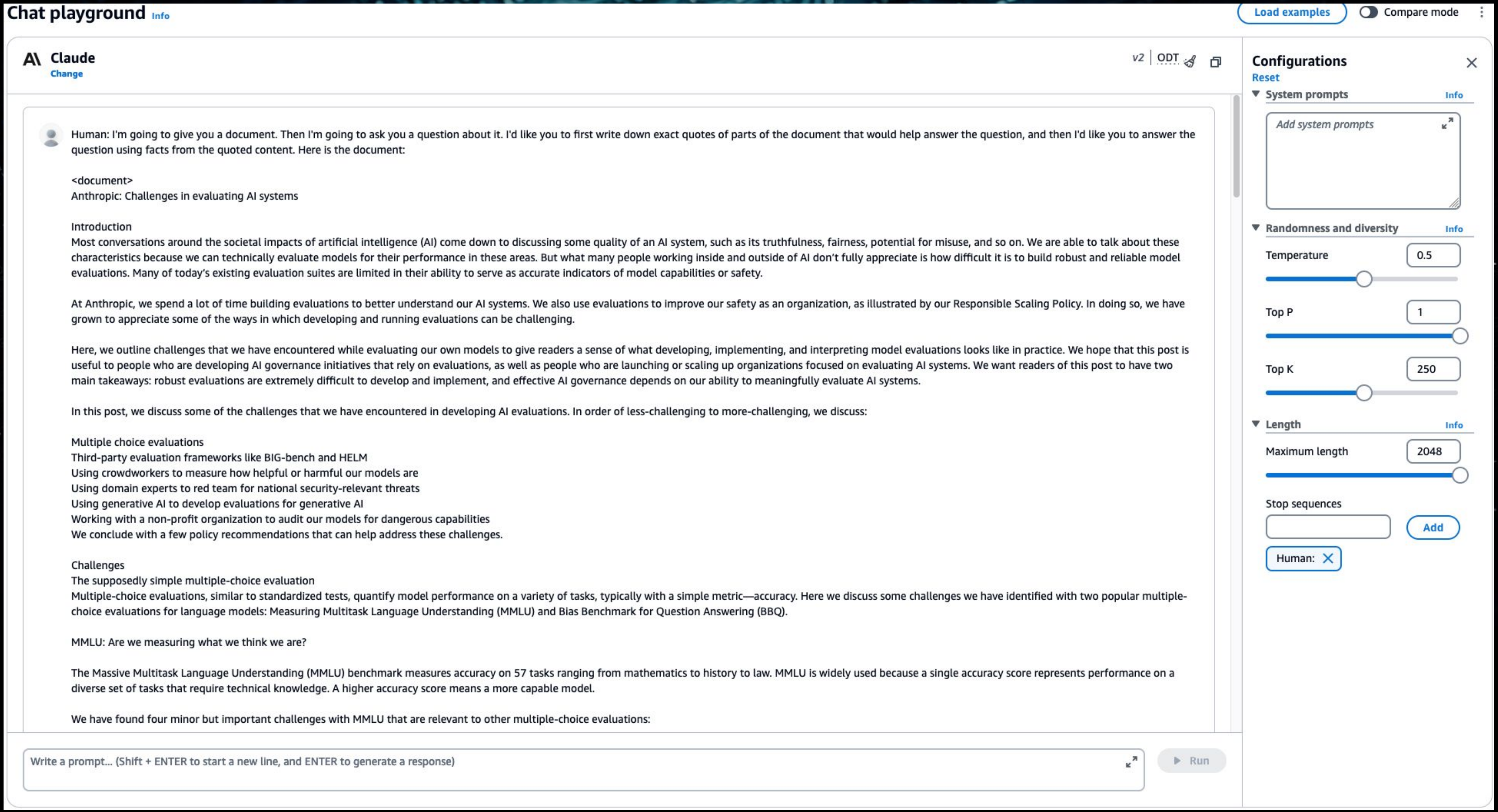
Using Pre-Trained Models via API Integration

- leverage pre-trained models through APIs provided by relevant parties such as AWS Bedrock, Google AI Platform etc

```python
import boto3
import json
brt = boto3.client(service_name='bedrock-runtime')

body = json.dumps({
    "prompt": "\n\nHuman: explain black holes to 8th graders\n\nAssistant:",
    "max_tokens_to_sample": 300,
    "temperature": 0.1,
    "top_p": 0.9,
})

modelId = 'anthropic.claude-v2'
accept = 'application/json'
contentType = 'application/json'

response = brt.invoke_model(body=body, modelId=modelId, accept=accept, contentType=contentType)

response_body = json.loads(response.get('body').read())

# text
print(response_body.get('completion'))
```

Model Type	Model Name
Text	• Jurassic-2 from AI21 Labs • Titan Text from Amazon • Claude v2 from Anthropic • Command R, R+ from Cohere • Llama 2 & 3 from Meta • Mistral 7B, Large from Mistral AI
Image Generation	• Titan Image Generator from Amazon. • Stable Diffusion from Stability AI
Text & Vision	• Claude 3 Haiku, Sonnet from Anthropic
Embedding	• Embed by Cohere • Titan by Amazon

Using Pre-Trained Models via Playgrounds



Continued..

Claude
Change

v2 | ODT

Conclusion
We hope that by openly sharing our experiences evaluating our own systems across many different dimensions, we can help people interested in AI policy acknowledge challenges with current model evaluations.
</document>

First, find the quotes from the document that are most relevant to answering the question, and then print them in numbered order. Quotes should be relatively short.

If there are no relevant quotes, write "No relevant quotes" instead.

Then, answer the question, starting with "Answer:". Do not include or reference quoted content verbatim in the answer. Don't say "According to Quote [1]" when answering. Instead make references to quotes relevant to each section of the answer solely by adding their bracketed numbers at the end of relevant sentences.

Thus, the format of your overall response should look like what's shown between the <example></example> tags. Make sure to follow the formatting and spacing exactly.

<example>
Relevant quotes:
[1] "Company X reported revenue of $12 million in 2021."
[2] "Almost 90% of revene came from widget sales, with gadget sales making up the remaining 10%."

Answer:
Company X earned $12 million. [1] Almost 90% of it was from widget sales. [2]
</example>

Here is the first question: In bullet points and simple terms, summarize the key challenges in evaluating AI systems.

If the question cannot be answered by the document, say so.

Answer the question immediately without preamble.

Assistant:

Relevant quotes:
[1] "Most conversations around the societal impacts of artificial intelligence (AI) come down to discussing some quality of an AI system, such as its truthfulness, fairness, potential for misuse, and so on."
[2] "We have found four minor but important challenges with MMLU that are relevant to other multiple-choice evaluations:"
[3] "Implementing BBQ was more difficult than we anticipated."
[4] "We needed significant engineering effort in order to just install BIG-bench on our systems."
[5] "Methods that work well for evaluating other providers' models do not necessarily work well for our models, and vice versa."
[6] "These experiments are expensive and time-consuming to run."
[7] "There is an inherent tension between helpfulness and harmlessness."
[8] "Red teaming AI systems is presently more art than science; red teamers attempt to elicit concerning behaviors by probing models, but this process is not yet standardized."
[9] "We currently rely on humans to verify the accuracy of model-generated evaluations."
[10] "Initially, we expected this collaboration to be straightforward, but it ended up requiring significant science and engineering support on our end."

Answer:
- Evaluating qualities like fairness is difficult. [1]
- Multiple choice evaluations have challenges like cheating and lack of standardization. [2]
- Implementing more advanced evaluations like BBQ involves unanticipated complexities. [3]
- Third party evaluations require significant engineering work. [4]
- Third party evaluations don't necessarily test models appropriately. [5]
- Real world human evaluations are expensive and time consuming. [6]
- There are tradeoffs between evaluating for helpfulness versus harmlessness. [7]
- Red teaming lacks standardized processes. [8]
- Model-generated evaluations require human verification. [9]
- Third party audits require providing significant internal support. [10]

Configurations
Reset

System prompts — Info
Add system prompts

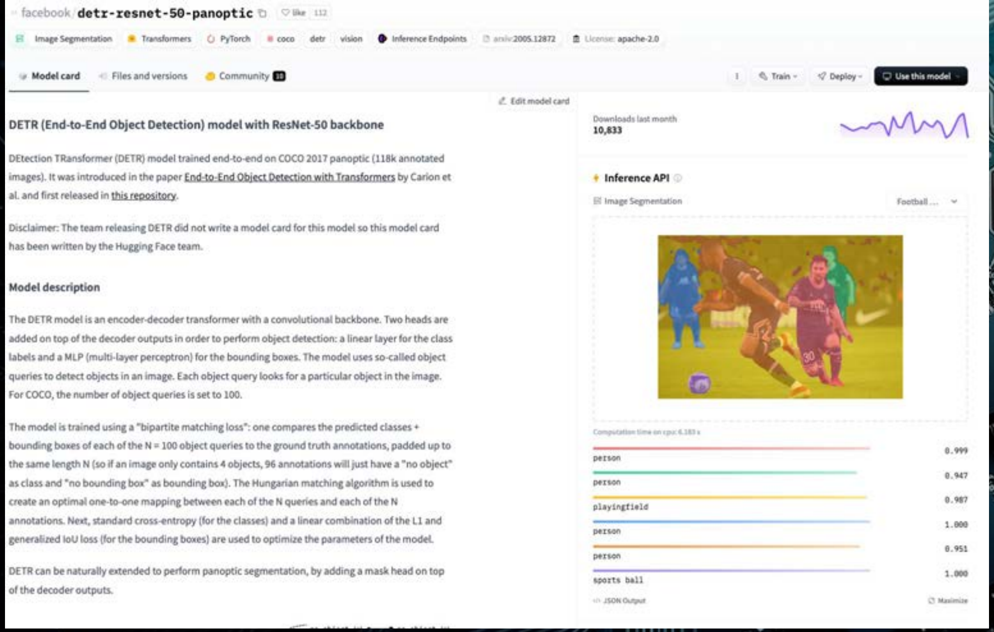
Randomness and diversity — Info
Temperature: 0.5
Top P: 1
Top K: 250

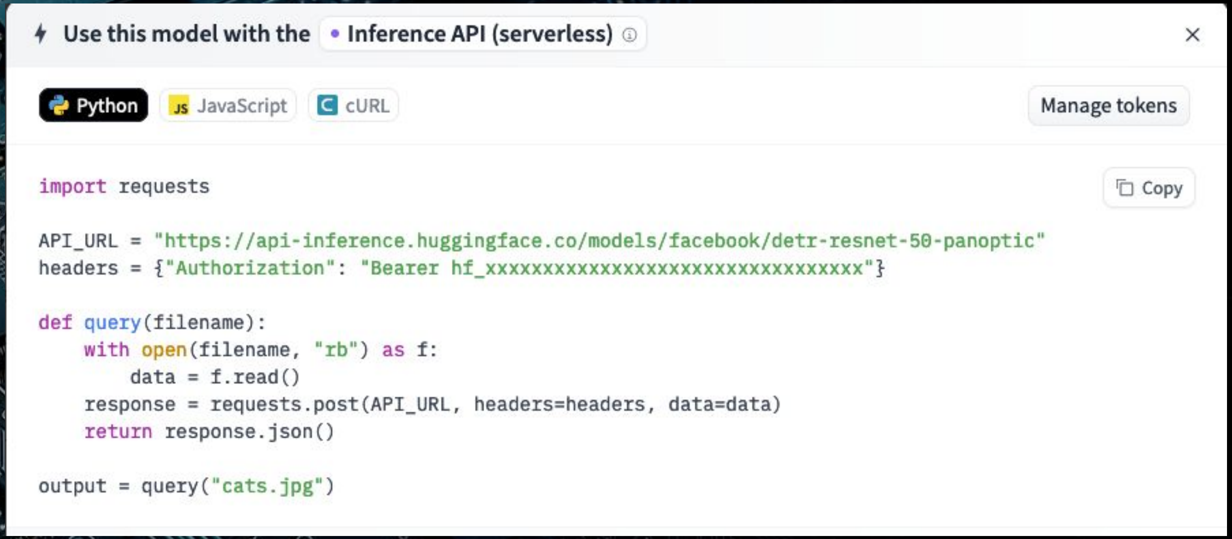
Length — Info
Maximum length: 2048

Stop sequences
Add
Human:

Examples

- Hugging Face(Serverless Inference For Prototyping)

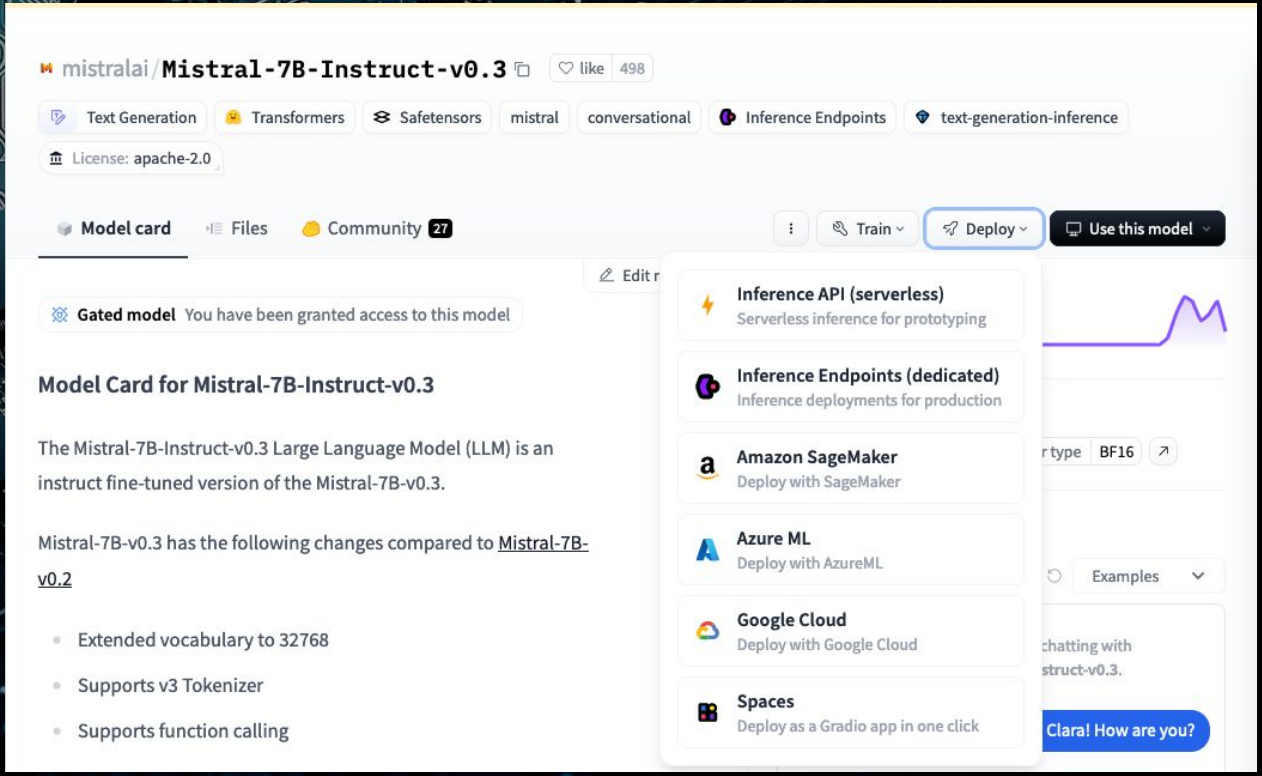




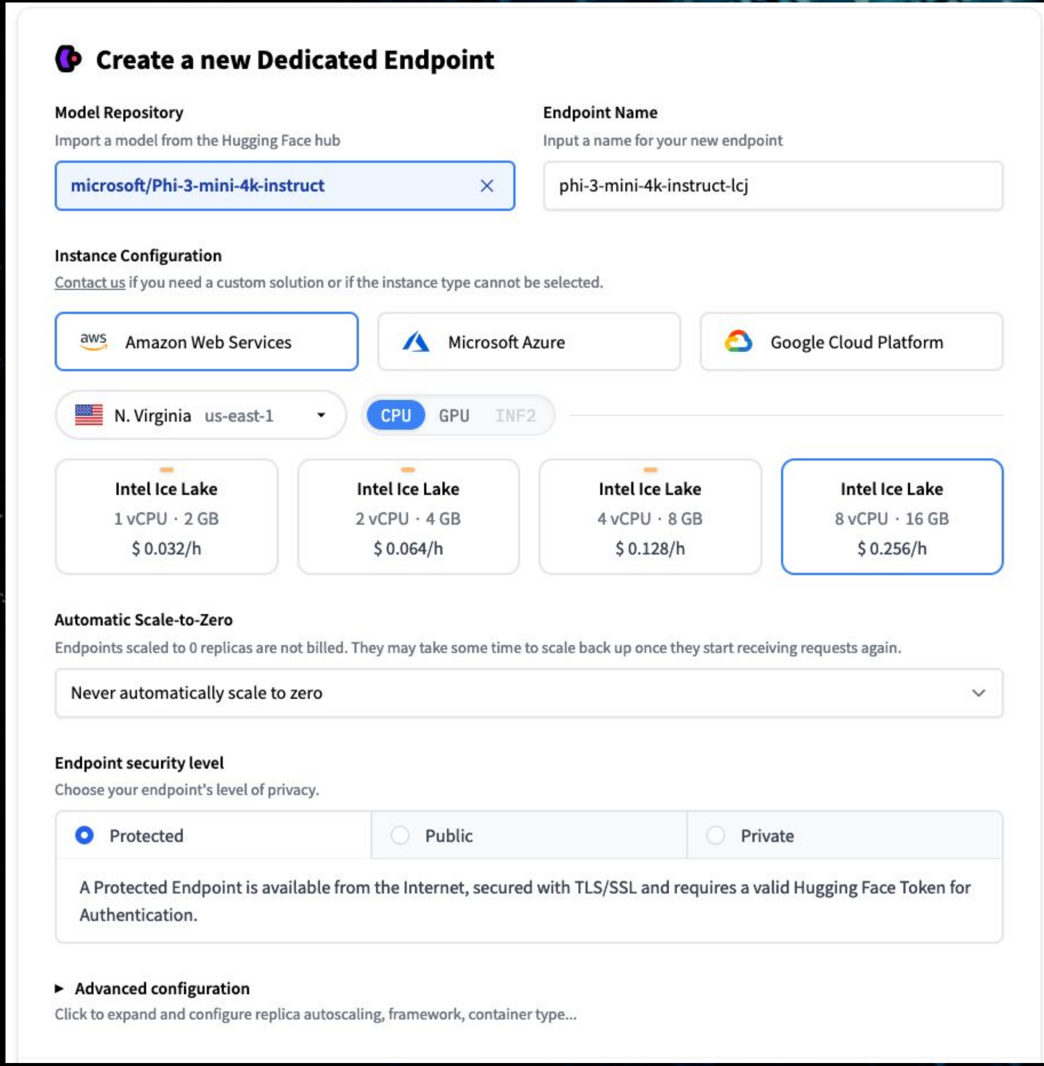
Hosting Models - Hugging Face Model Hub

- Deploying Pre-Trained Models with Hugging Face Model Hub
- Easy integrations with AWS SageMaker
- Vast selection of Models on Hugging Face Model Hub

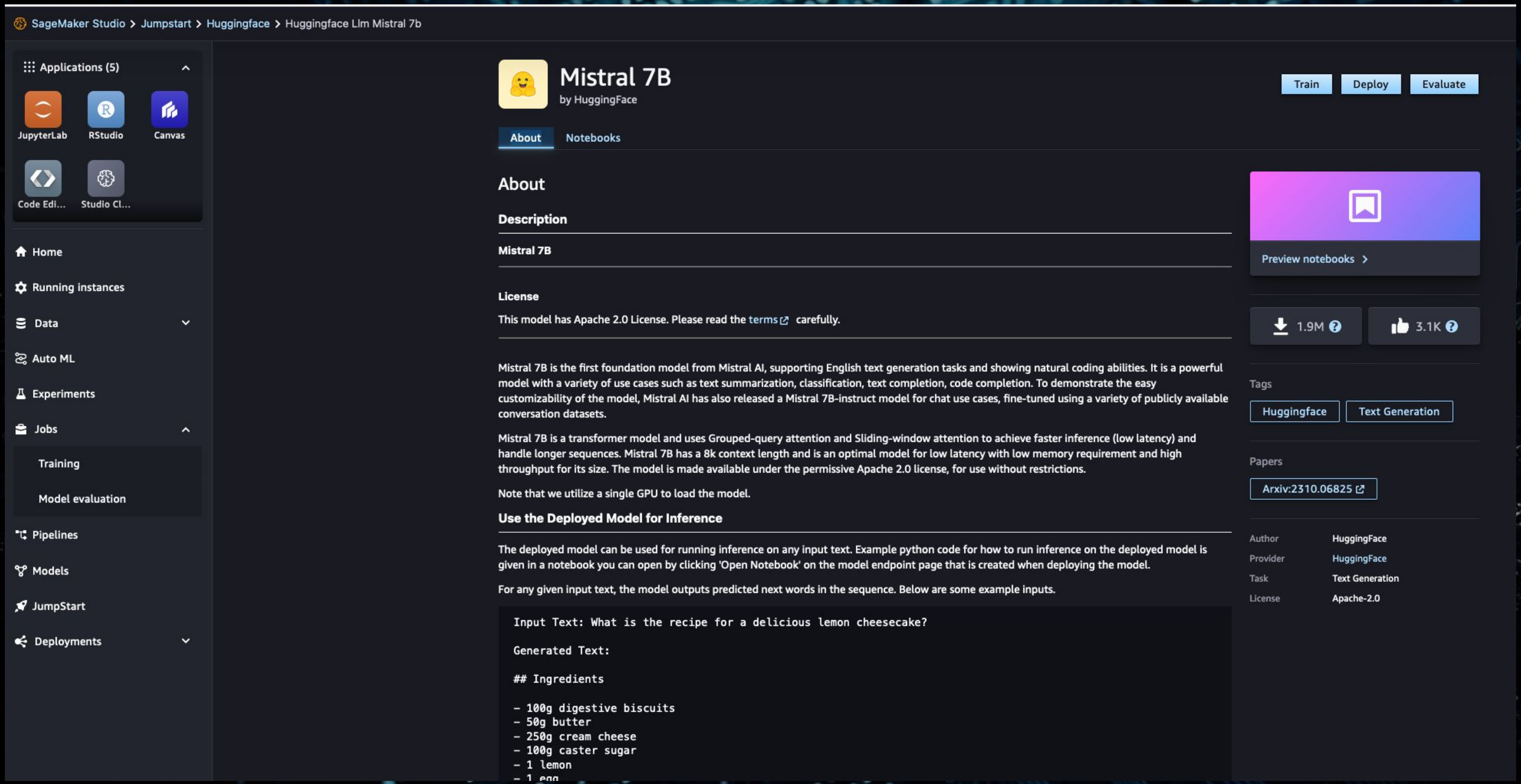
```python
import sagemaker
import boto3
from sagemaker.huggingface import HuggingFaceModel

try:
    role = sagemaker.get_execution_role()
except ValueError:
    iam = boto3.client('iam')
    role = iam.get_role(RoleName='sagemaker_execution_role')['Role']['Arn']

# Hub Model configuration. https://huggingface.co/models
hub = {
    'HF_MODEL_ID':'Salesforce/instructblip-flan-t5-xxl',
    'HF_TASK':'image-to-text'
}

# create Hugging Face Model Class
huggingface_model = HuggingFaceModel(
    transformers_version='4.37.0',
    pytorch_version='2.1.0',
    py_version='py310',
    env=hub,
    role=role,
)

# deploy model to SageMaker Inference
predictor = huggingface_model.deploy(
    initial_instance_count=1, # number of instances
    instance_type='ml.m5.xlarge' # ec2 instance type
)
```



Continued…


Create a new Dedicated Endpoint
Model Repository
Import a model from the Hugging Face hub
microsoft/Phi-3-mini-4k-instruct
Endpoint Name
Input a name for your new endpoint
phi-3-mini-4k-instruct-lcj
Instance Configuration
Contact us if you need a custom solution or if the instance type cannot be selected.
Amazon Web Services
Microsoft Azure
Google Cloud Platform
N. Virginia us-east-1
CPU
GPU
INF2
Intel Ice Lake
1 vCPU · 2 GB
$ 0.032/h
Intel Ice Lake
2 vCPU · 4 GB
$ 0.064/h
Intel Ice Lake
4 vCPU · 8 GB
$ 0.128/h
Intel Ice Lake
8 vCPU · 16 GB
$ 0.256/h
Automatic Scale-to-Zero
Endpoints scaled to 0 replicas are not billed. They may take some time to scale back up once they start receiving requests again.
Never automatically scale to zero
Endpoint security level
Choose your endpoint's level of privacy.
Protected
Public
Private
A Protected Endpoint is available from the Internet, secured with TLS/SSL and requires a valid Hugging Face Token for Authentication.
Advanced configuration
Click to expand and configure replica autoscaling, framework, container type...

Hosting Models – AWS SageMaker Studio



Deploy Custom/Fine tuned Models

- For the foundational models, AWS Bedrock allows customizing the model for some popular LLMs like Llama, Titan and Command

- Import a model with Custom Model Import using Bedrock – Supported for Mistral, Flan and Llama architectures

- Build Sync/Async endpoint on SageMaker via SageMaker SDK by creating the Tar ball of the model artifacts

```python
import sagemaker
from sagemaker import Model

# Specify the location of model artifacts in Amazon S3
model_data = "s3://your-bucket/your-model/model.tar.gz"

# Create a SageMaker model
sagemaker_session = sagemaker.Session()
model = Model(model_data=model_data,
              image_uri="your-custom-container-image",  # Specify the Docker
image URI
              role="your-sagemaker-role",
              sagemaker_session=sagemaker_session)

# Deploy the model as an endpoint
predictor = model.deploy(initial_instance_count=1,
instance_type="ml.g4dn.12xlarge")
```

```python
from djl_python import Input, Output
import os
import torch
from transformers import pipeline, AutoModelForCausalLM, AutoTokenizer

predictor = None

def get_model(properties):
    model_name = properties['model_id']
    local_rank = int(os.getenv('LOCAL_RANK', '0'))
    dtype = torch.float16
    model = AutoModelForCausalLM.from_pretrained(model_name,
low_cpu_mem_usage=True, torch_dtype=dtype)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    generator = pipeline(task='text-generation', model=model,
tokenizer=tokenizer, device=local_rank)
    return generator


def handle(inputs: Input) -> None:
    global predictor
    if not predictor:
        predictor = get_model(inputs.get_properties())

    if inputs.is_empty():
        # Model server makes an empty call to warmup the model on startup
        return None

    data = inputs.get_as_json()['prompt']
    result = predictor(data, do_sample=True)
    return Output().add(result)
```

Limitations of Standalone LLMs

- Potential inaccuracies and hallucinations in generated content.
- The challenge of providing up-to-date information due to training data cut-off.
- Difficulty in handling domain-specific queries with general-purpose LLMs.
- Limited Contextual Understanding
- Ethical and Bias Issues
- Fine-tuning large LLMs require significant computational resources
- Handling of potentially sensitive data underscores the importance of stringent data governance

Introducing Retrieval-Augmented Generation (RAG)

- **What is RAG?**
  - Retrieval-Augmented Generation (RAG) is an advanced AI approach that combines the strengths of retrieval systems with generative models
  - It aims to enhance the capabilities of LLMs by grounding generated responses in factual information retrieved from knowledge bases
- **How RAG Works?**
  - Retrieval Component
  - Generative Component
- **Benefits of RAG**
  - Improved Accuracy
  - Enhanced Contextual Relevance
  - Overcoming Knowledge Cut-off

Implementation Steps for RAG

- **Select a Knowledge Base**
  - Internal company database
  - Ensure the knowledge base is comprehensive, up-to-date, and well-maintained
- **Data Preparation**
  - Clean and preprocess the data in the knowledge base to ensure consistency and quality.
  - Use scalable and efficient storage solutions such as **AWS OpenSearch**
  - Index the data using appropriate techniques to facilitate efficient retrieval **(Vector database with FAISS(Facebook AI Similarity Search) Engine**
- **Develop the Retrieval System**
  - Implement a retrieval system that can efficiently search and retrieve relevant documents or data from the knowledge base
- **Combine Retrieval with LLM**
  - Concatenate or format the retrieved data with the original query to provide context to the LLM

Cost Concerns and Reduction Strategies for LLMs

- **High Computational Requirements**

Model Name	Model ID	Max Total Tokens	Default instance type
Meta-Llama-3-8B	meta-textgeneration-llama-3-8b	8192	ml.g5.12xlarge
Meta-Llama-3-8B-Instruct	meta-textgeneration-llama-3-8b-instruct	8192	ml.g5.12xlarge
Meta-Llama-3-70B	meta-textgeneration-llama-3-70b	8192	ml.p4d.24xlarge
Meta-Llama-3-70B-Instruct	meta-textgeneration-llama-3-70b-instruct	8192	ml.p4d.24xlarge

Instance Type	GPUs	Cost per hour
ml.g5.12xlarge	4 NVIDIA A10G	7$
ml.p4d.24xlarge	8 NVIDIA A100	37$

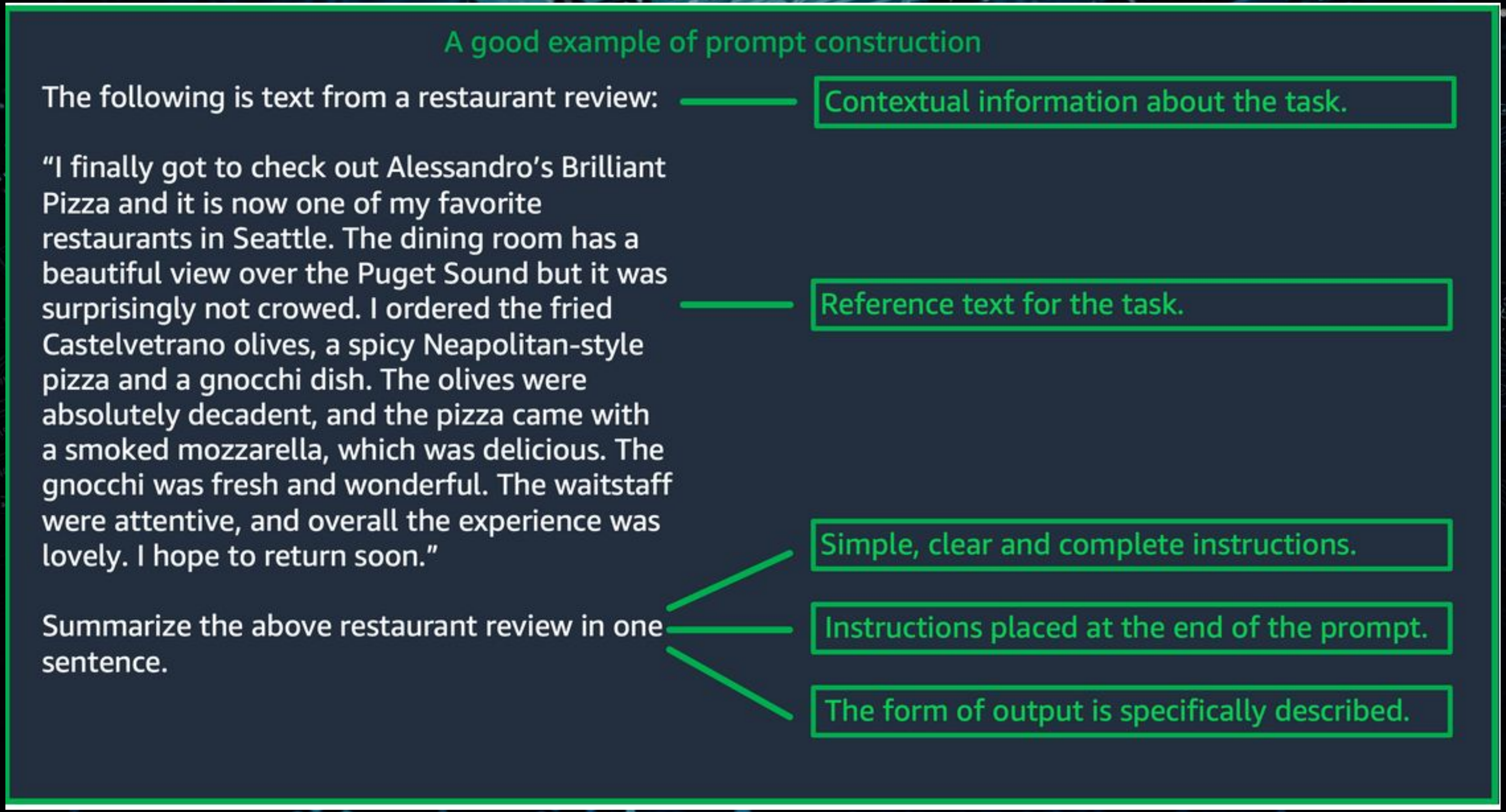
- **Storage and Data Management**
  - **Knowledge base costs**
- **Operational and Maintenance Costs**
  - **Maintenance of LLMs**
  - **Availability**
  - **Low latency**

Cost Reduction Strategies

- **Use Pre-Trained Models by APIs and Services**
  - AWS Bedrock, OpenAI etc.
- **Leverage Foundation Models offered by cloud service providers**
  - AWS SageMaker, AWS Bedrock
- **Optimize Model Size**
  - Model Distillation
  - Quantization and Pruning
- **Efficient Resource Utilization**
  - Auto scaling, Asynchronous invocation, Spot instances, Reserved instances, Caching, Input Batching.
- **Data Management and Storage Solutions**
  - IVF-Flat/IVF PQ Indexing techniques, Compression
- **Model cascading**
  - Deploying a series of models with increasing complexity and computational cost

Improvisation of LLMs

Prompt engineering is about crafting inputs that guide the model towards the desired output



Text Classification Prompt

Prompt template for Anthropic Claude:

```
"""


Human: {{classification task description}}
<text>
{{input text content to be classified}}
</text>

Categories are:
{{category name 1}}
{{category name 2}}
{{category name 3}}

Assistant:"""
```

(Source: Written by AWS)

User prompt:

```
Human:  Classify the given product description into given
categories. Please output the category label in
<output></output> tags.

Here is the product description.

<text>
Safe, made from child-friendly materials with smooth edges.
Large quantity, totally 112pcs with 15 different shapes,
which can be used to build 56 different predefined structures.
Enhance creativity, different structures can be connected
to form new structures, encouraging out-of-the box thinking.
Enhance child-parent bonding, parents can play with their
children together to foster social skills.
</text>

Categories are:
(1) Toys
(2) Beauty and Health
(3) Electronics

Assistant:
```

Output:

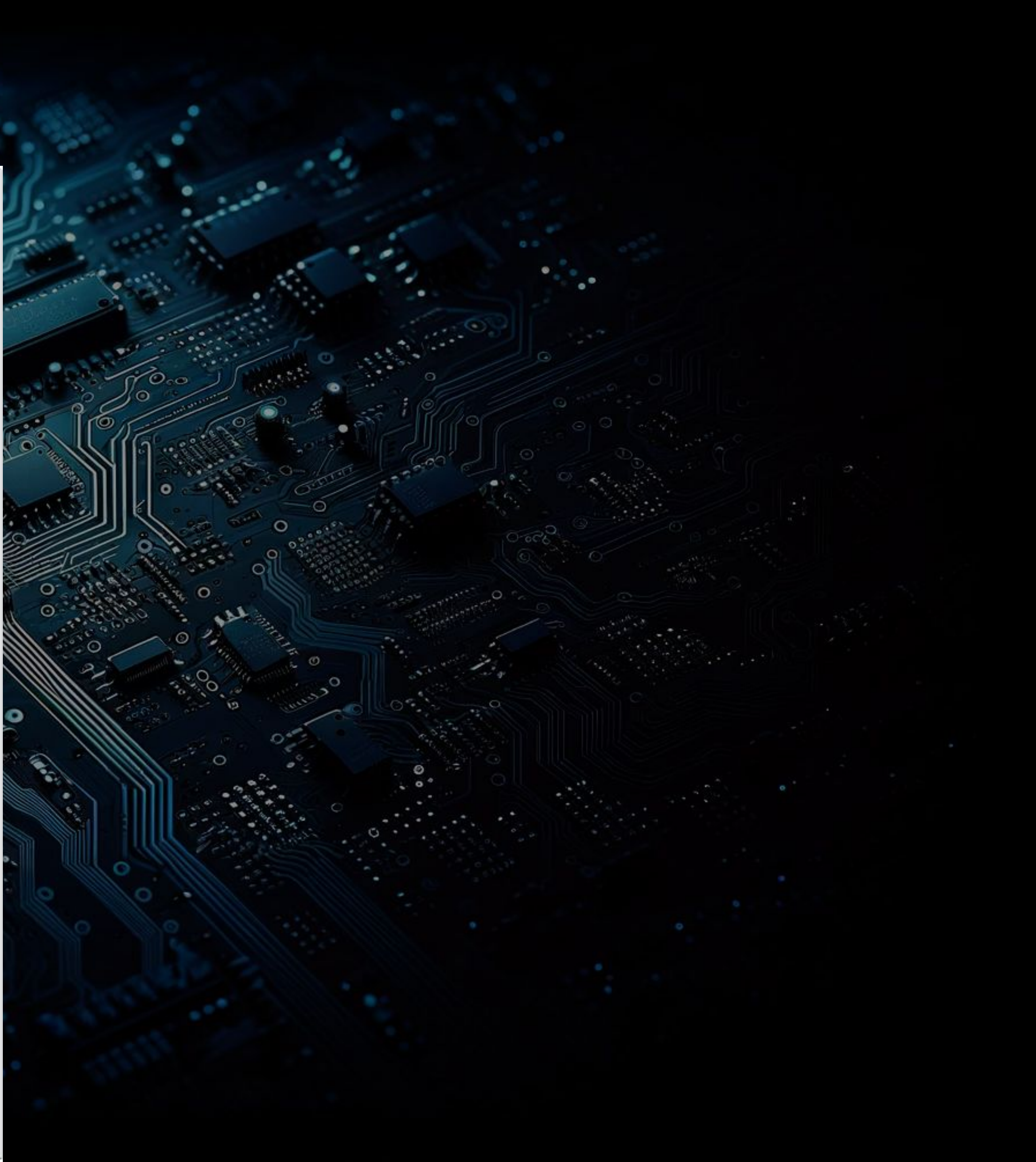
```
<output>Toys</output>
```

Question-Answer prompt

```
Prompt template for Anthropic Claude:
"""

Human: {{Instruction}}
<text>
{{Text}}
<text>
{{Question}}

Assistant:"""
```

```
User prompt:


Human: Read the following text inside <text></text>
XML tags, and then answer the question:

<text>
On November 12, 2020, the selection of the Weeknd to headline
the show was announced; marking the first time a Canadian solo artist
headlined the Super Bowl halftime show. When asked about preparations
for the show, the Weeknd stated, "We've been really focusing
on dialing in on the fans at home and making performances
a cinematic experience, and we want to do that with the Super Bowl."

The performance featured a choir whose members were dressed in white
and wore masks over their faces with glowing red eyes, and were
standing within a backdrop of a neon cityscape. The performance
opened with a white figure dressed the same as the choir being
lowered into the backdrop where the choir was standing while singing
"Call Out My Name". At this time, the Weeknd sat in a convertible
against a skyline backdrop designed to resemble the Las Vegas Strip.
For the next part of the performance, the backdrop then split open
to reveal the Weeknd, who then performed "Starboy", followed by "The Hills".

Next, performing the song "Can't Feel My Face", the Weeknd traveled
through a labyrinth constructed behind the stage, joined by dancers
dressed in red blazers and black neckties similar to his,
but with their faces covered with bandages, in keeping with
the aesthetic of his fourth studio album After Hours (2020).
The dancers would wear these bandages throughout the performance.
In the labyrinth section of the performance, camerawork was visually unsteady.
The next songs performed were "I Feel It Coming", "Save Your Tears",
and "Earned It". For the "Earned It" performance, the Weeknd
was accompanied by violinists. For the finale of the show,
the Weeknd took to the field of the stadium with his dancers to perform
"Blinding Lights". He and the dancers entered the field by performing
"House of Balloons / Glass Table Girls". The performance ended
with an array of fireworks.
</text>
Based on the text above, what songs did the Weeknd play
at the Super Bowl halftime show?

Assistant:
```

```
Output:
Based on the text, the songs the Weeknd played
at the Super Bowl halftime show were:

- Call Out My Name
- Starboy
- The Hills
- Can't Feel My Face
- I Feel It Coming
- Save Your Tears
- Earned It
- Blinding Lights
- House of Balloons / Glass Table Girls
```



Text Summarization Prompt

```
Prompt template for Anthropic Claude:
"""

Human: Please read the text:
<text>
{{text context}}
</text>

Summarize the text in {{length of summary, e.g., “one
sentence” or “one paragraph”}}

Assistant:"""
```

```
User prompt:


Human: Please read the text:

<text>
In game theory, the Nash equilibrium, named after the mathematician
John Nash, is the most common way to define the solution
of a non-cooperative game involving two or more players.
In a Nash equilibrium, each player is assumed to know
the equilibrium strategies of the other players,
and no one has anything to gain by changing only one's own strategy.
The principle of Nash equilibrium dates back to the time of Cournot,
who in 1838 applied it to competing firms choosing outputs.
</text>

Summarize the text in one sentence.

Assistant:
```

```
Output:
The Nash equilibrium is a concept in game theory
where each player has adopted a strategy that they have no incentive
to deviate from given the other players' strategies.
```

Code generation Prompt

Prompt template for Anthropic Claude

```
"""

Human: Write a function in {{programming language}} to
{{task or purpose}}

Assistant:"""
```

User prompt:

```
Human: Write some python code that quickly approximates
the square root of a number.

Assistant:
```

Output:

````
Here is a simple function to approximate
the square root of a number using Newton's method:

```python
def sqrt(n, iterations=10):
x = n
for i in range(iterations):
x = (x + n/x) / 2
return x
```

To use it:

```python
print(sqrt(25))
# 5.0

print(sqrt(15, iterations=20))
# 3.872983346207417
```
````

Leveraging LLMs as a Software Engineer and Tech Professional

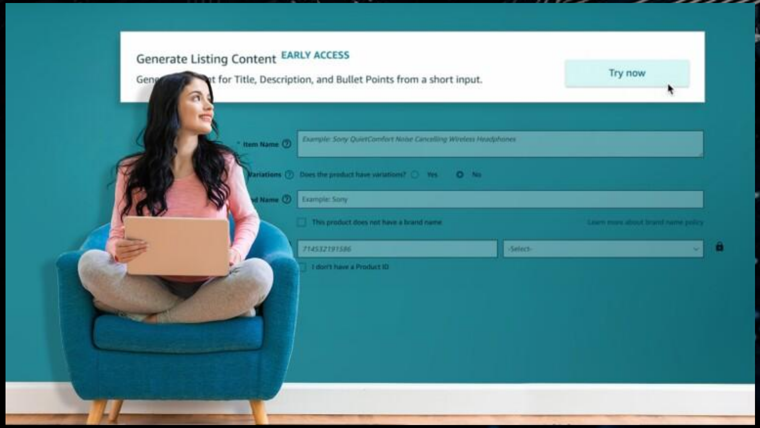
- **Automated Code Generation:**
  - Use LLMs to generate boilerplate code, automate repetitive coding tasks, and suggest code completions.
  - **Example:** GitHub Copilot, powered by OpenAI Codex, Amazon Q can help write code snippets and entire functions based on comments and context within your codebase.
- **Code Review and Debugging:**
  - LLMs can assist in identifying bugs, suggesting fixes, and performing initial code reviews to enhance code quality.
  - **Example:** Tools like DeepCode and CodeGuru use AI to analyze code, provide recommendations, and detect potential vulnerabilities.
- **Documentation Generation:**
  - Automate the creation of comprehensive documentation for your codebase, including function descriptions, usage examples, and API documentation.
  - **Example:** LLMs can generate docstrings, README files, and API documentation from code comments and structure.
- **Natural Language Interfaces:**
  - Develop applications that allow users to interact with software using natural language commands and queries.
  - **Example:** Integrating LLMs into chatbots or voice assistants to perform tasks such as scheduling, querying databases, and controlling software applications.
- **Technical Support and Troubleshooting:**
  - Implement AI-driven chatbots and virtual assistants to provide first-level support, answer technical questions, and guide users through troubleshooting steps.
  - **Example:** AI-powered help desks can resolve common issues, reducing the workload on human support teams and improving response times.
- **Data Analysis and Insights:**
  - Use LLMs to analyze large datasets, generate reports, and extract meaningful insights from textual data such as logs and customer feedback.
  - **Example:** Leveraging LLMs for sentiment analysis, trend identification, and summarizing complex datasets into actionable insights.

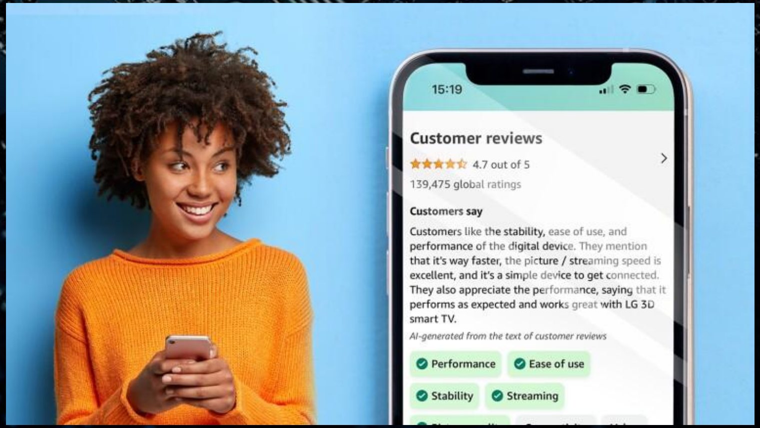
Leveraging LLM In Infringement Detection

- **Trademark and Copyright Violations**
  - Identifying unauthorized use of brand names, logos, and other intellectual properties
- **Counterfeit Detection**
  - Recognizing subtle differences in genuine and fake product listings
- **Obfuscation detection**
  - Spotting subtle variations in brand names intended to bypass detection (e.g., "N1ke" instead of "Nike")
- **Behavioral Analysis**
  - Analyzing seller behavior and history to identify potential infringer

Real World Examples

- Review highlights on product listings by Amazon
- Creation of compelling product titles and descriptions for Product data on Amazon websites
- Amazon Pharmacy staff answer questions more quickly because it scours internal wikis and other sources of information, and then summarizes the findings
- Reduced the human audits for detecting infringements by 80% for famous brands like Apple etc





THANK YOU!