

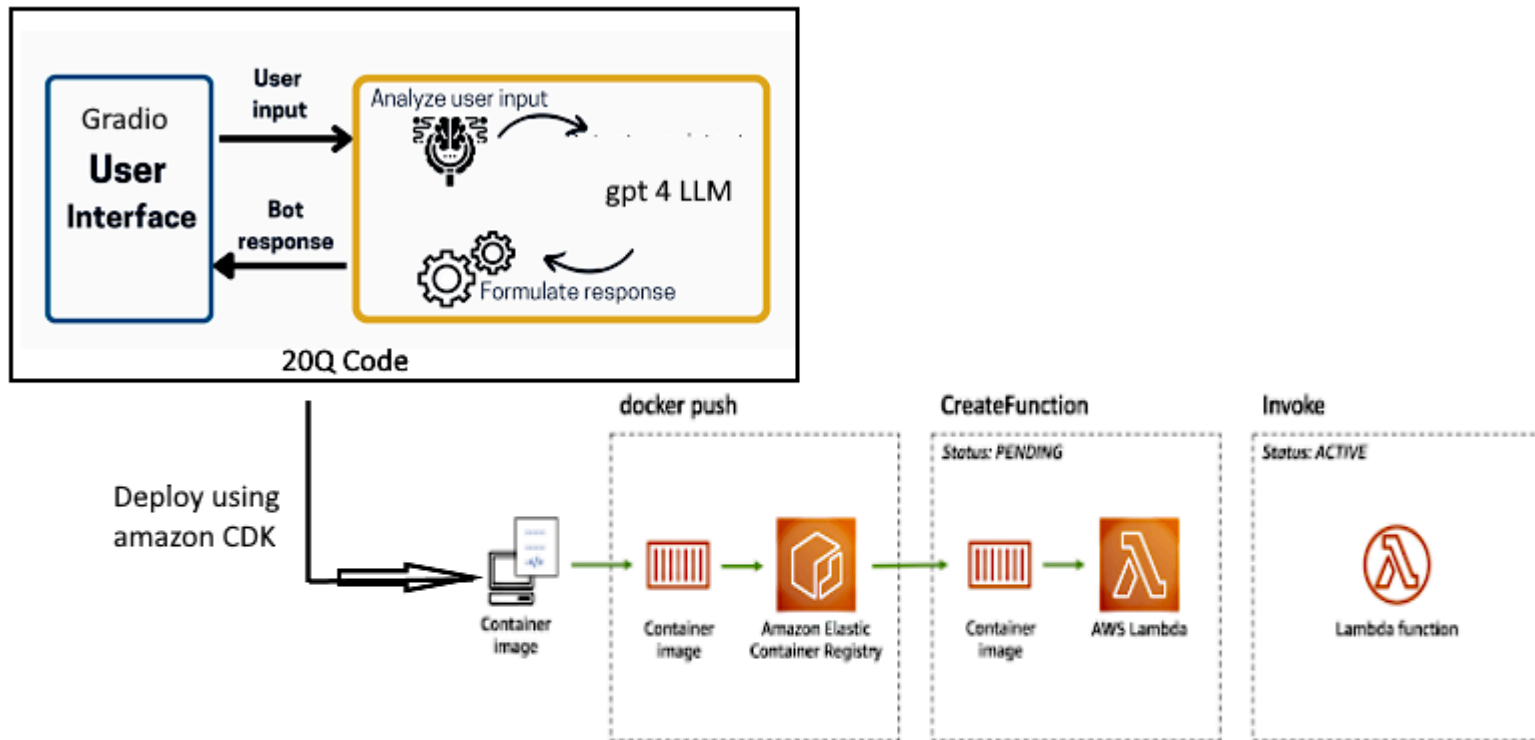
20q LLM AWS experiment

Rules of the game:

The game is 20 questions - whereby you, the user, think of something (i.e. a dog, a coffee, a display, a football) and the AI has to guess what you are thinking of.

The AI can ask up to 20 yes/no questions to find out what you are thinking of.
You must answer yes or no truthfully.

Architecture diagram



Architecture Discussion

Gpt-4 as our LLM

We are using open ai's gpt-4 as it has been proven to be better at playing 20 questions than gpt 3.5. Gpt-3.5 struggles to follow the game, often repeating questions. Due to it's larger size, gpt-4 is able to pay attention to more complex patterns in data, allowing it to consider all of the questions in the chat history.

User interface: gradio

Gradio is an open-source Python package that allows you to quickly create easy-to-use, customizable UI components for your ML model, any API, or even an arbitrary Python function using a few lines of code. You can integrate the Gradio GUI directly into your Jupyter notebook or share it as a link with anyone.

Serverless computing

We will be making use of serverless computing in our deployment. Serverless computing is a method of providing backend services on an as-used basis. A serverless provider allows users to write and deploy code without worrying about the underlying infrastructure. Serverless computing can offer advantages over traditional cloud-based or server-centric infrastructure, like greater scalability, more flexibility, and quicker time to release, all at a reduced cost. However, serverless computing is/was mostly used for web development and not for machine learning, due to its computing and resource-intensive nature.

Advantages of Serverless Computing:

- No Server Management Required:
 - Developers don't need to deal with servers; server management is handled by the vendor.
 - Reduces the need for DevOps investment, lowering expenses.
 - Frees up developers to focus on application development without server constraints.
- Cost Efficiency:

- Developers are charged only for the server space they use.
- Pay-as-you-go model, similar to a phone plan.
- Code runs only when backend functions are needed, automatically scaling up as required.
- Dynamic, precise, and real-time provisioning.
- Inherent Scalability:
 - Serverless architectures scale automatically as the user base grows or usage increases.
 - Functions run in multiple instances, with servers starting up and shutting down as needed.
 - Contrast with traditional architectures where fixed server space may be overwhelmed by sudden increases in usage.

AWS CDK

When deploying machine learning Infrastructure such as virtual machines and databases, it's great to use Infrastructure as Code (IaC). IaC is an approach to managing and provisioning cloud infrastructure in a more automated, efficient, and scalable way. It involves defining and managing infrastructure resources, such as virtual machines, databases, networking components, and more, using code and scripts rather than manual configuration through a graphical user interface or command-line tools. IaC enforces consistency by representing desired environment states via well-documented code in formats such as JSON.

In terms of IaC offerings, AWS provides AWS CloudFormation templates or AWS CDK, where CloudFormation uses YAML or JSON configuration files to describe resources and their dependencies. The AWS CDK is a high-level development framework that allows developers to define cloud infrastructure using familiar programming languages like TypeScript, Python, Java, and C#. It abstracts away many of the complexities of CloudFormation templates and provides a more developer-friendly experience.

I have chosen to use AWS CDK because of my familiarity with the python programming language and the fact that CDK is something of a developer-friendly version of Cloud Formation. AWS CDK tends to allow you to define infrastructure in less lines of code too.

Why lambda?

When it comes to comparing the FaaS offerings of the three major cloud providers (AWS, GCP, Azure) they are extremely similar and comparable, both in terms of features and cost.

While AWS Lambda is the more mature and most popular of the three, Azure Functions appears to have some very similar features and in some ways, more options to accommodate edge cases. GCP Cloud Functions has a few less bells and whistles but is still fairly comparable to the other two.

AWS Lambda is the only option to offer highly customized concurrency management options, while Azure and GCP are a little vague on how concurrent executions are handled. Cold starts can affect the performance of FaaS workloads that are very sensitive to delay but there are ways to mitigate it, and it appears to affect the Azure platform more than its two competitors. Additionally, AWS now offers “provisioned concurrency” as an approach to eliminate cold starts with Lambda.

Docker and Lambda

I have used Docker to containerize chatbot functions, then run them on-demand via Lambda.

Deployment

Deployment requires the use of python 3.11.5.

Local deployment

1. Install dependencies

```
pip install -r requirements.txt
```

2. Run application. Running on local URL: <http://127.0.0.1:8080>

```
Python app.py
```

Cloud Serverless Deployment

Run all commands from the root directory.

1. install the AWS CDK CLI

```
npm install -g aws-cdk
```

3. Install dependencies

```
pip install -r requirements.txt
```

3. Bootstrap the CDK

[optional]: export aws profile

```
export AWS_PROFILE=hf-sm
```

Bootstrap project in the cloud

```
cdk bootstrap
```

4. Deploy Gradio application to AWS Lambda

```
cdk deploy
```

Delete AWS Lambda when you are done with the application

```
cdk destroy
```

Revision

What Is Underwriting?

Underwriting is the process through which an individual or institution takes on financial risk for a fee. This risk most typically involves loans, insurance, or investments.

We are looking for a Machine Learning Engineer to join our collaborative team of 4, to focus on building the most advanced and flexible machine learning enabled insurance platform to cater for insurers, brokers and underwriters of all sizes and sectors.

Bringing your years of commercial experience in machine learning techniques and principles as well as knowledge of data science techniques, this role would be particularly of interest to you having come from a previous start up or scale up environment.

The insurance workforce is aging and replacing those employees as they exit the workforce will not be an easy task. At the same time, AI is not the same as it was even five years ago. The economics have substantially improved, and the technology has matured significantly.

Artificial

This SaaS platform revolutionizes insurance underwriting with features like algorithmic underwriting, automated data ingestion, instant risk assessment, a robust contract builder, and an end-to-end underwriting workbench. It enables rapid scaling, automates routine tasks, and employs machine learning for smart decision-making. The platform supports various data formats, ensures quick response times, and streamlines compliance checks, analytics, and communication with underwriters and brokers. In essence, it enhances efficiency, reduces manual work, and improves decision-making in the insurance underwriting process.

Why join artificial?

So I've been looking for a promising startup to join.

- Admiration for the team. Co-ceos have founded companies before
- The insurance industry and specifically, Underwriting industry, is ripe for disruption according to Accenture: [Transforming claims and underwriting with AI](#)
- The work that you are doing here really aligns with my skill set
- Good size series A 9.5 million
-

Relevant work: Robo checking

- Automated fact checking of macro economic claims for a functioning democracy
 - Unemployment
 - Inflation
 - Employment
 - Interest rates
- Pipeline: data ingestion> Data cleaning/filtering using claim detection > slot extraction > slot validation> Processing> Fact check delivered
- Technology used jointbert. BERT for Joint Intent Classification and Slot Filling

Experienced with cloud technologies and deployment

Was responsible for deploying infrastructure for various development teams throughout the firm

- Worked extensively with Amazon Web Services namely; vpc, Lambda, EC2, ecs and s3.
- Using Sonar and DataDog, I introduced code quality reporting and performance monitoring to the CI pipeline
- Used the YAML programming language to create cloud-formation templates to automate the deployment of key AWS infrastructure

Full Fact

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

Logically Deployment

- Merge in code to monolith ci/cd
- Update intelligence schema defining inputs and outputs and api calls
- Work with ml ops to stand up model behind api
- Deploy to dev
- Backfilling then Deploy to staging and test outputs and UI
- Backfilling then Deploy to production

SageMaker endpoint for realtime inference

Amazon SageMaker is a fully managed machine learning (ML) service. With SageMaker, data scientists and developers can quickly and confidently build, train, and deploy ML models into a production-ready hosted environment.

Deployment and serving

Deploying struggles

You've felt the pain, struggle and glory of serving your own fine-tuned open source LLM, however, you ultimately decided to return to Open AI or Anthropic due to cost, inference time, reliability and technology challenges :(You've also given up on renting a A100 GPU (many providers have GPUs fully booked until the end of 2023!). And you don't have 100K to shell out for a 2 tier A100 server box.

deepset production experience

🕶️ Everything was easy when we started deepset Cloud. The biggest model was deberta-large and it had 435 million parameters. Using this model added some latency but it was predictably in the range of a couple of seconds.

🤖 Enter ChatGPT and the era of remote inference LLMs. Each token takes time to generate and all of a sudden, the duration of a request was determined by the user's prompt and how much text the model had to generate.

🤖 Your latency could be anywhere between 5 and 50 seconds. Very hard to optimize! Have some random failures of OpenAI's API on top of that to keep it fun. Sometimes, our ingress timeout just killed the request before GPT-4 answered.

Batch vs online serving

Model serving simply means hosting machine-learning models (on the cloud or on-premises) and making their functions available via API so that applications can integrate AI into their systems. With this action, you can use the machine learning model with just a few clicks. Model serving is critical as businesses cannot deliver AI solutions to an extensive user base without making them accessible.

To generate the predictions, you need to invoke the ML model somehow. A simpler example is batch inference. You can run it daily, hourly, or on-demand and use a workflow manager to orchestrate the process. It would access the data source, run the model and write the predictions to a database. The online inference is a bit more complex. You might wrap the model as a service and expose REST API to serve predictions at request. There are more moving pieces to track.

Model Serving Strategies

Following are some of the common model serving strategies used by ML professionals:

1. **Offline Serving:** The end-user is not immediately exposed to your model in this method. It works by performing a batch inference job on your test dataset, caching the findings in a high-performance database, and then serving those results to end-users. The most significant downside of offline serving is that it is a "cold" deployment.
It usually only works in 'push' workflows, where the end-user only accepts requests from the model server but does not create any requests. Offline serving can be used in 'pull' workflows where the end-user can send requests to the model server. It's challenging to implement because end-users often have unrealistic expectations regarding response times.
2. **Online model as service:** When a model learns from human input automatically, it is said to be online. A neural network with a batch size of one is a canonical example of online machine learning. A complete backpropagation pass is started based on the input each time a user requests to your model endpoint, changing the model weights simultaneously with serving the request.
3. **Model as Service:** The most typical model serving strategy in production environments is to deploy a model as a (micro)service. Clients are given access to

a model interface via an API endpoint in this paradigm (REST or otherwise). Clients send POST or GET queries to the endpoint to get what they need. This is a scalable, responsive model service deployment technique with a flexible deployment strategy.

4. **Edge deployment:** It refers to serving the model directly on the client device rather than on the server because it shifts the computation from the server to the edge (the client device). It's tricky since the client's hardware (a web browser, a user computer, or a mobile device) is severely limited.

Optimisations

- Deep speed optimisations
- Quantisation
- Predicting generation time
- Vector optimisations

Model monitoring

What could go right?

Wondering about what could go right, you should think back to the planning phase of your machine learning project workflow, where you hopefully planned around the user and the business objective.

Primarily, everything that could go right is all that guarantees that the business objective and the user (human/service/system) needs are met.

How do you monitor and measure this? It's very unique for each business use case, and typically depends on:

- What does your business define as success? What were the KPIs set during the planning phase?
- What were the performance expectations before deploying to production?
- If my model returns a prediction to a client, what does it expect in terms of results and how quickly should it be delivered?

What could go wrong?


You can monitor what could go wrong with your machine learning model in production at two levels:

- Functional level monitoring – monitoring model performance, inputs (data), and outputs (predictions).
- Operational level monitoring – monitoring at system and resource level.

ML Ops

Kubernetes

Official definition of Kubernetes



- > Open source **container orchestration tool**
- > Developed by **Google**
- > Helps you **manage containerized applications**
in different **deployment environments**

physical

Volume

virtual

cloud

