artificial.

Lead ML Engineer - technical task - LLM

We would like you to create an application that can play *20 Questions*, using a State-of-the-art AI model to guess the answer.

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.

Task

This is an exercise designed to test your ability to design an end-to-end solution, rather than optimising for a specific task, so it is recommended not to spend too much time fine-tuning each step of the following process.

1. Create an architecture diagram for a solution to this problem - think carefully about how you would deploy this in a cost effective manner in a cloud environment. Cloud best practices (e.g. serverless where possible and relevant) will be appreciated - AWS preferred but not strictly required. If possible, make reference to the named services you would use at each step.

2. Discuss the following
    a. How do different components interact with each other and why you have chosen this approach - include a short discussion of the other options

considered and why you did not choose them.

b. Explain the decisions behind the model and components you decided might best suit this task

c. Briefly explain any deployment steps necessary and why you chose this method.

3. Produce code and running instructions for this (email zip file or link to a GH repo). We are looking for your ability to write production ready code in Python.
   The resulting submission should be able to be played with a reasonable speed. Some latency is understandable but should not be excessive.

Note:

- The AI should only play 20 questions, and should respond with an apologetic response when asked about something unrelated.
- You are not required to train or fine tune an LLM. You should be able to do this challenge using an LLM available over API (see below) or self hosted. - We will not deploy your submission in our own environment, therefore it is not necessary for you to deploy to the cloud and provide us a link. We will however take careful consideration of submitted code and infrastructure as code - so having a loosely working submission is useful for us to understand how one would deploy this. - We do not expect you to incur any expense in undertaking this assignment

In case you require access to a LLM service

We have provided a free LLM proxy available at the following URL:
`https://candidate-llm.extraction.artificialos.com`

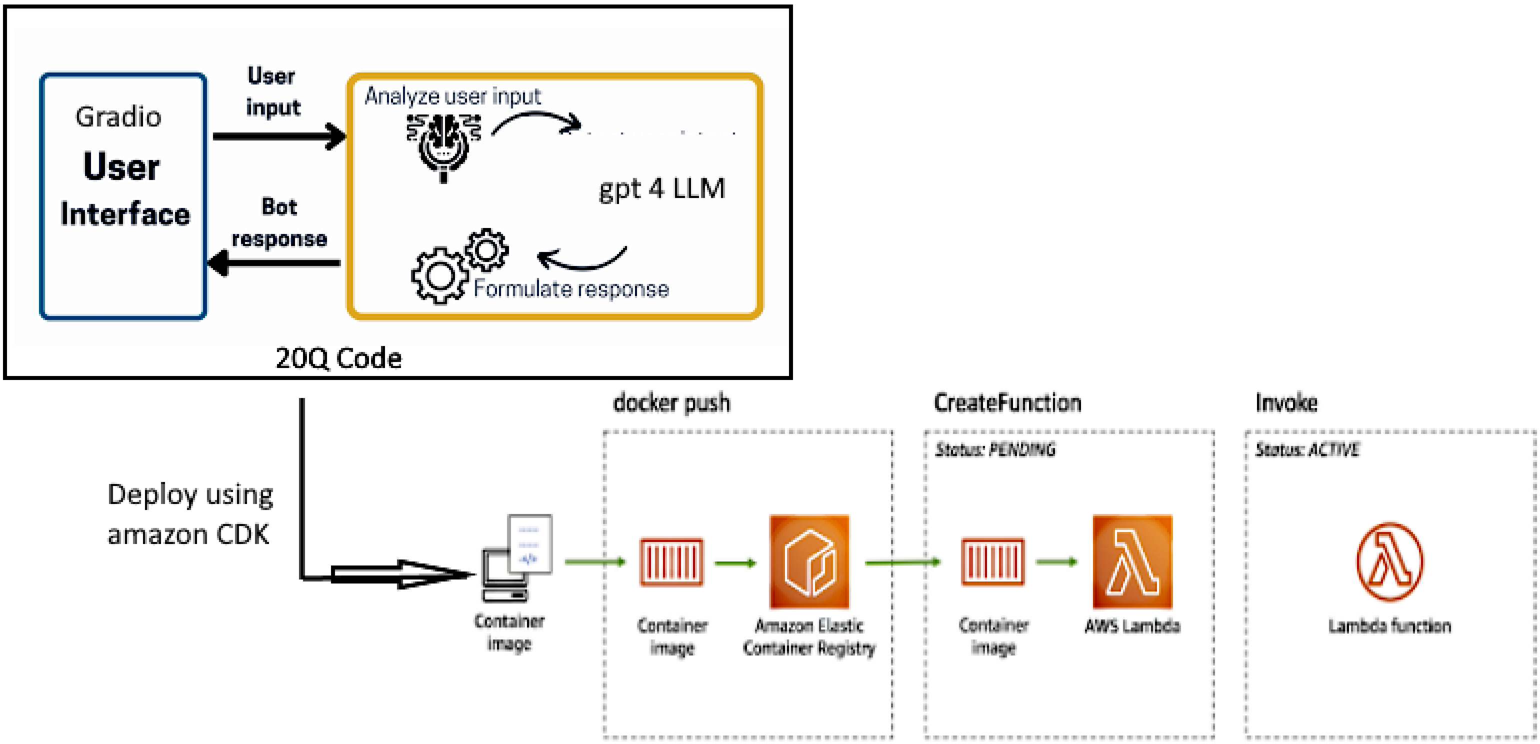
The specification of this API is almost identical to
`https://platform.openai.com/docs/api-reference/`

The Key exception is that your api key must be passed in a header called: `x-api-key` rather than the `Authorization` header.

The required header is below and your personal key will be provided with your assignment via email:

```
x-api-key: <API_CANDIDATE_KEY>
```

Architecture diagram



Architecture Discussion

Gpt-4 as our LLM

We are using gpt-4 as our model as it has been proven to be better at playing 20 questions than gpt 3.5. This is probably due to it’s larger size.

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

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. 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.
- Quick Deployments and Updates:
    - No need to upload code to servers or perform backend configuration for releasing new versions.
    - Enables rapid uploading of code in bits or functions, facilitating quick product releases.
    - Updates, patches, fixes, or new features can be added one function at a time without altering the entire application.
- Reduced Latency:
    - Code can run closer to end users, decreasing latency.
    - Applications can be run from servers near the end user, reducing the travel distance for requests.
    - Cloudflare Workers enable serverless latency reduction.

Programmatic Infrastructure Deployment

Programmatic infrastructure deployment involves managing infrastructure using a descriptive model, employing the same versioning as the DevOps team uses for source code. Major cloud providers, including AWS, Google Cloud, and Microsoft Azure, offer tools for programmatic infrastructure management.

AWS provides AWS CloudFormation templates or AWS CDK, where CloudFormation uses YAML or JSON configuration files to describe resources and their dependencies. Google Cloud recommends Deployment Manager, using YAML templates.

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 is the only provider 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

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
```

Boostrap 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
```