Candidate Name: An Nguyen - Senior Machine Learning Engineer

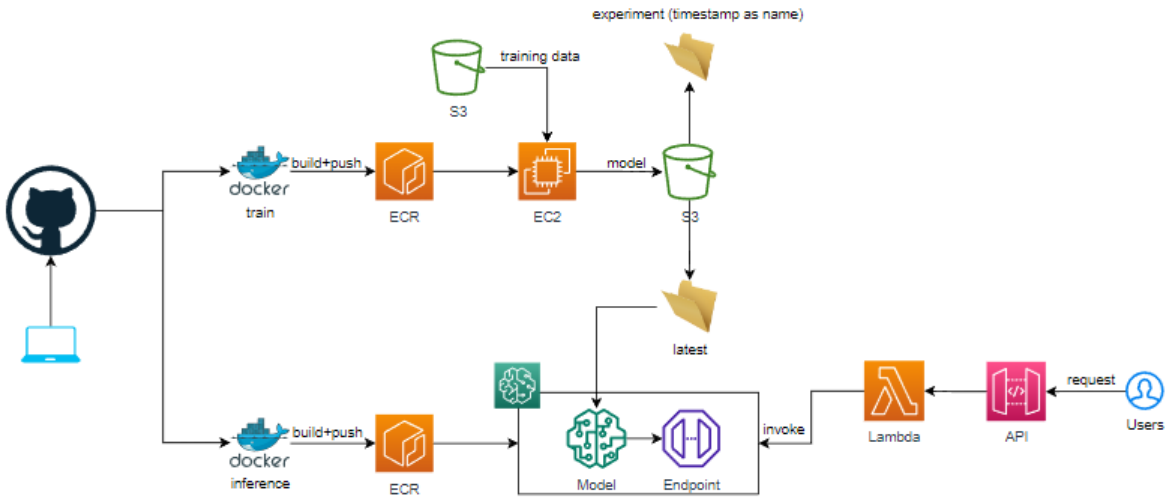
Coding Test

I. Requirement

1. Create a diagram of your architecture.
2. Use Hugging Face GPT models, developing your own open-source model instead of relying on third-party APIs like ChatGPT.
3. Implement comprehensive unit tests.
4. Package files into a Docker image.
5. Set up an MLOps pipeline for training and hosting the model on AWS Sagemaker, utilizing AWS free credits for new accounts.
6. Expose an API endpoint for other developers to access the service.
7. Provision your architecture on AWS using Terraform, AWS CDK, or other Infrastructure as Code tools.

II. Design

Github repo: https://github.com/anguyenbus/chatgpt_detector



- Project is stored in github
- Inside the project, we will have code for train and inference. Train and inference will be packaged into docker separately.
- Docker images will be built using github actions
- After built, the images are stored on ECR.
- Training: we can either spin up an EC2 for training, or sagemaker for training model. Training is not the focus of this project, then we will not go deeper to the selection.

    - Input data: This is not a focus. So, for the purpose of illustration, a very small data set is stored in the docker image. In practice, we can store data in S3 bucket.
    - Output of the training will be saved with versioning
        - If a model is deployed, the S3 bucket prefix is latest
        - Other models will be saved with timestamp (as name of the directory) in S3 for further references
- Inferencing: we will use sagemaker to
    - Create model (linked to S3 URL where model is stored)
    - Create endpoint config
    - Create inference endpoint
    - Endpoint will be invoke by a Lambda function
    - API can run lambda
    - User can send request to lambda

III. Implementation

- Cloud environment: AWS
- Infrastructure as code: Terraform
- Unit testing: pytest
- CICD: github actions
- Model: Huggingface Roberta Open API detector
- Flask API served as API gateway

IV. Testing

The simplest way to test is with curl:

curl -X POST -H "Content-Type: application/json" -d '{"input_text": "this is the input test"}' https://3iyy9gezd5.execute-api.ap-southeast-2.amazonaws.com/test/api-chatgpt-detector

The expected result is:

Expected return

```
{"input_text":"this is the input test","predicted_class":0}
```

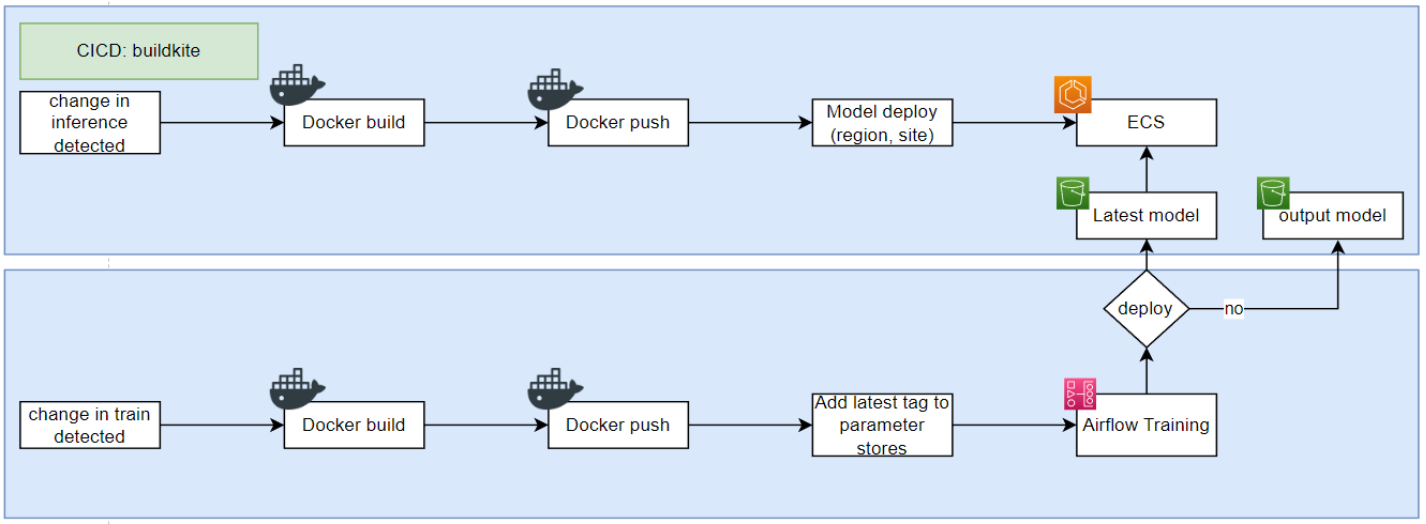
V. Suggestion

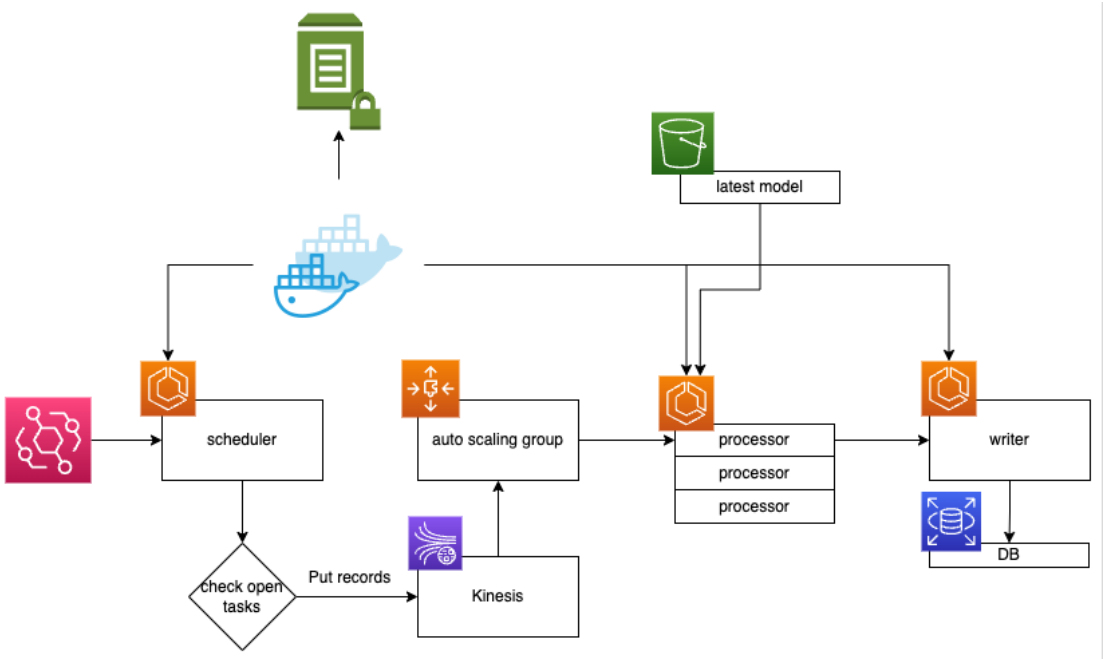
I will suggest a design where the batch process is included. Before I start, I would like to mention the techstack I use and the reason why.

Github Repo: https://github.com/anguyenbus/mle-project

1. Buildkite as CICD: I have more experience with buildkite from working at Reejig. For the project above, I used github actions because it is free.
2. Airflow: I will use apache airflow to schedule training and inference.
3. Terraform: I use terraform to provision infrastructure

4. Model: the model in the project below is a very simple linear regression model.
5. Parallel processing: I trigger multiple ECS instances, and process queues from Kinesis.
6. Django API: I use Django API to serve the inference model.
7. Slack: inference results will be channelled to slack for team members to monitor.
8. Parameter store: I use parameter store to store the latest tag of docker images.\
9. EvenBridge: trigger event
10. JavaScript: I use a bit of javascript to create buildkite file for multiple region purposes





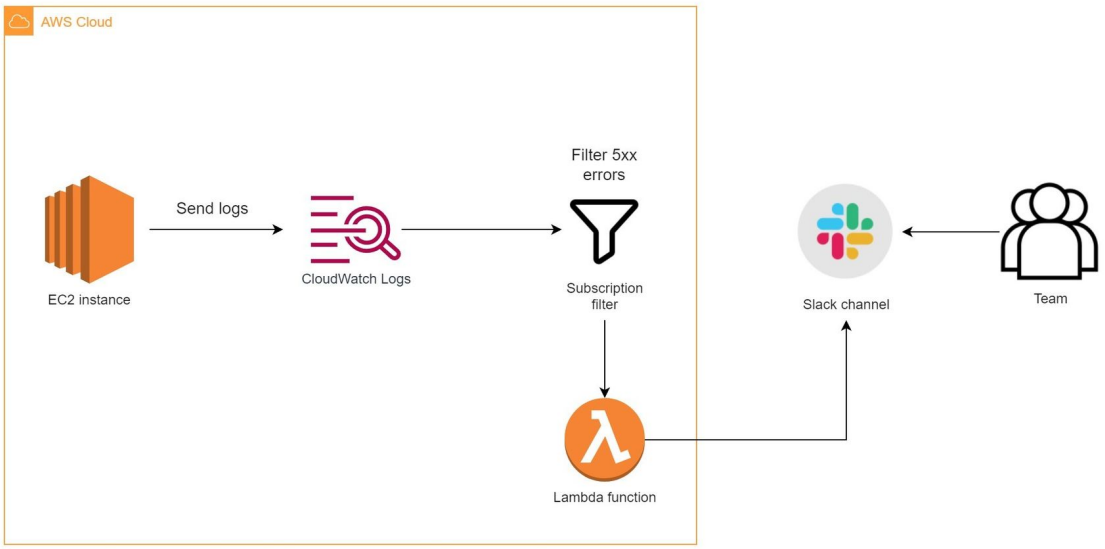
Suggestion 1: In case if we have too much data, that it can take a very long time to process all of it, we can have batch processing. In this particular example, we can use EventBridge to trigger weekly batch processing. Here you see we have ECS service for scheduler, processor, and writer:

- Scheduler will check if there are open tasks, if yes then it will push the records to Kinesis. Kinesis then, via auto scaling group, spins up around 8 ECS instances called processors.

- Each processor will download the latest models from S3 and predict. The results will be written to the database by our ECS writer.

Suggestion 2: In cases that we have multiple sites within a region, or even multiple regions. We can use javascript to populate yml file for different sites/regions.

Suggestion 3: output results to slack using webhook.



VI. Appendix

I have experience working with ElasticSearch, which might be relevant to our current project, although I'm not entirely sure. Nonetheless, I believe sharing this experience will contribute to our discussion and give you a better understanding of my skills.

In my role as a machine learning engineer at Reejig, I actively participated in dynamically constructing and optimizing ElasticSearch queries. This effort significantly improved the relevance of research results and enhanced the user experience by 50%. Moreover, I ensured effective knowledge transfer to my team and other software engineers by explaining complex concepts in the simplest possible manner.

Candidate Name: An Nguyen - Senior Machine Learning Engineer

```
# main query to ES
query = {
    "size": size,    similar to LIMIT
    "_source": "id", columns, etc ["id","current_role"]
    "query": {
        "bool": {
            "should": should_conds,
            "must": must_conds,
            "must_not" : must_not_conds,
            "minimum_should_match": minimum_should_match
        }
    }
}
```

In addition to my experience with ElasticSearch, I have also worked extensively with Apache Airflow as a job scheduler for regular training and inferencing tasks.

Furthermore, I had the chance to assist data scientists in optimizing their code by replacing pandas operations like group_by and apply with numpy operations (vectorization). This optimization resulted in a significant reduction of time and cost, reaching 3X to 4X lower.

I am always eager to learn new technologies and strive to enhance engineering practices. These are essential aspects that I seek in any new working environment.