Let's take a look at the next problem statement below.

Background Context:

You are working as a data scientist for a healthtech company called BeHealthy. It is a late-stage startup. Be Healthy provides a web-based platform for doctors to list their services and manage patient interactions. It provides various services for patients, such as booking interactions with doctors and ordering medicines online. **(Similar to 1mg, PharmEasy)** Here, doctors can easily manage appointments, track past medical records and reports and give e-prescriptions.

BeHealthy has enrolled thousands of doctors across the major cities in India. You can also assume that millions of patients are using BeHealthy's services; hence, many free-text clinical notes would be present in the e-prescriptions.
BeHealthy would want to convert these free-text clinical notes to structured information for analytics purposes. You have solved such a problem in your previous courses on NLP. You had created a CRF model to recognise the Named Entity in the medical data set.
For example:
Clinical Note: "The patient was a 62-year-old man with squamous cell lung cancer, which was first successfully treated by a combination of radiation therapy and chemotherapy."
Disease: Lung Cancer
Treatment: Radiation Therapy and Chemotherapy

Before solving any business problem, we must always keep in mind why a business needs to resolve the problem. In many cases, data scientists directly jump to the solution, using data sets like Kaggle competitions. But in the real world, you should be able to justify the business need, define KPIs and then design an optimal solution.
A company tries to perform real-time analytics, as it receives a large volume of data throughout the day. So, they must create APIs to run models whenever clinical data arrives.

Business goal:

Now, let's take a look at a business goal for BeHealthy:
Currently, if you need to extract diseases and treatments from free text, a trained person with clinical knowledge must manually look at the clinical notes and then pull this information. A data entry team would manually look at the clinical text and extract information about diseases and their treatment data. A data validation team would validate the extracted information. This process is prone to errors, and as the data increases, the data-entry team's size would need to be scaled up.

Automating this process would result in reduced man hours. The data-entry team would not be required. The data validation team would perform validation on extracted data. It would also significantly reduce manual errors.

Using the above information, answer the following questions:

Q1. **System design:** Based on the above information, describe the KPI that the business should track.

Answer:
Based on the above information, the business should track the following KPIs:

1. Accuracy of the model in identifying diseases and treatments from clinical notes. This KPI would help the business to ensure that the model is performing well and is extracting accurate information from the clinical notes.

2. Time taken by the model to extract diseases and treatments from clinical notes. This KPI would help the business to ensure that the model is processing the data in real-time and providing the information promptly.

3. Reduction in manual errors. This KPI would help the business ensure that the model reduces the manual errors previously made by the data entry team.

4. Reduction in man hours. This KPI would help the business ensure that the model reduces the time and resources required to extract information from clinical notes.

Overall, the KPIs should be designed to align with the business goals and help the business to measure the success of the solution.

Q2. **System Design:** Your company has decided to build an MLOps system. What advantages would you get by opting to build an MLOps system?

Answer:
As explained earlier, BeHealthy has enrolled thousands of doctors and millions of patients. Every day, a significant amount of clinical data is produced, as numerous doctor visits are happening each day. So, we require a solution that will scale commensurate with the huge volume of incoming data.
We would require the following:
Reduced lag in model development and deployment: This can be achieved by acquiring and cleaning large amounts of data, setting up tracking and versioning for experiments and model training runs and
setting up the deployment and monitoring pipelines for the models that get to production.

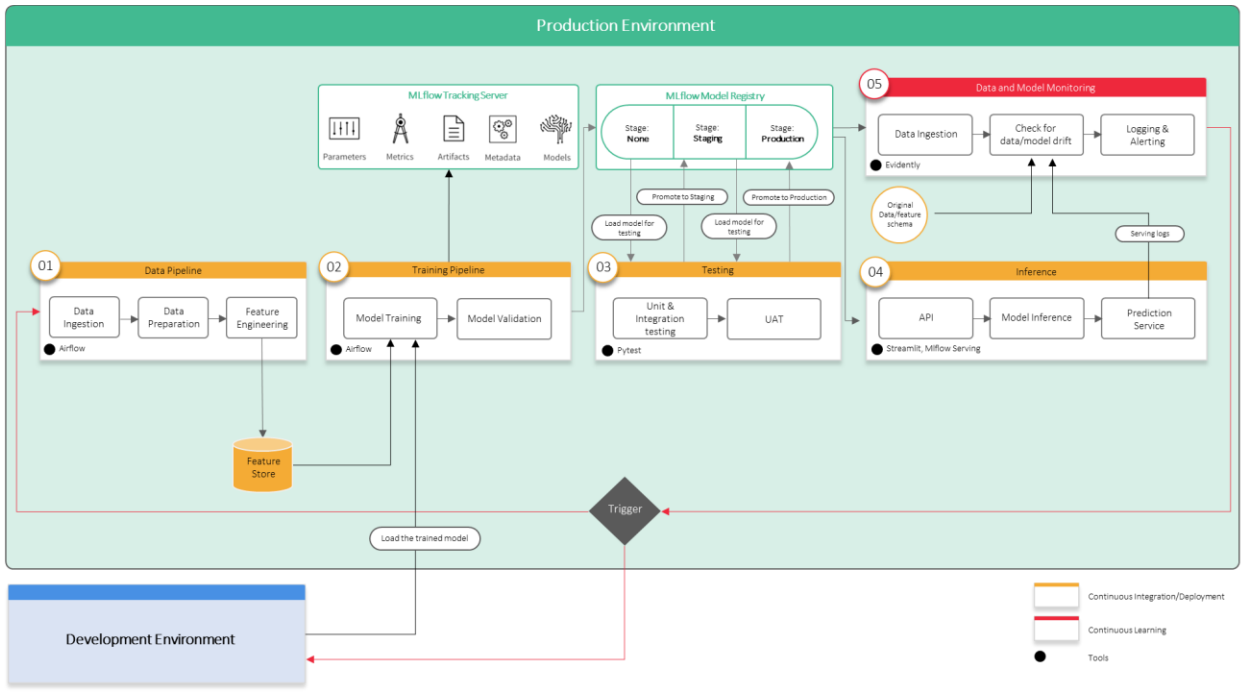
Experiment tracking: This is a part (or process) of MLOps focussed on collecting, organising, and tracking model training information across multiple runs with different configurations (hyperparameters, model size, data splits, parameters and so on). We could try different NLP techniques to perform entity recognition. We could experiment with different classifiers, preprocessing scripts and hyperparameters.
As mentioned earlier, because ML/DL is so experimental in nature, we use experiment tracking tools for benchmarking different models created either by different companies, teams or team members.
Continuous training: ML models in production can have reduced performance due to suboptimal coding and constantly evolving data profiles. As we encounter more and more data, we will encounter new diseases and treatments that may not have been present in the training data; therefore, we will need to retrain our models. Continuous training is necessary to avoid model decay over time.
Extracting medical NER helps create structured data for a downstream analytics use case.
Continuous Montioring: The continuous monitoring pipeline will check the drift level of the input data. If the level of drift is significant, the model will not perform well; therefore, necessary action will need to be taken to contain it.
Real-time analytics: Companies receive a large volume of data throughout the day. So, they must create APIs for running models whenever clinical data arrives.

Learners need not answer in the same manner as given above, but the crux of their answers should be the same. Any other answers are also acceptable. Please check Link1 and Link2 for other acceptable answers.

Q3. **System design**: You must create an ML system with the features of a complete production stack, from experiment tracking to automated model deployment and monitoring. For the given problem, create an ML system design (diagram).

The MLOps tools that you want to use are up to your judgement. You can use open-source tools, managed service tools or a hybrid.

You can upload the design/diagram as an image in the PDF.
Answer:



Here is a detailed explanation of each component in the ML system design and their responsibilities for the given use case.

1. **Data pipeline:**

The data pipeline is responsible for acquiring, processing, and transforming raw data into a suitable format for training the model. The responsibilities of the data pipeline include:

- Extracting clinical notes from the BeHealthy platform's database and storing it in a data storage system like AWS S3 bucket or GCP Cloud Storage.
- Preprocessing the raw data by cleaning and normalizing it, removing stop words, and identifying the Named Entities.
- Creating training, validation, and test datasets and storing them in a data storage system.

2. **Training pipeline:**

The training pipeline trains the machine learning model using the preprocessed data. The responsibilities of the training pipeline include:

- Developing a model architecture and training algorithm.
- Defining hyperparameters, like learning rate, batch size, etc.
- Training the model using the preprocessed data, and saving the best model checkpoints.

3. **Testing pipeline:**

The testing pipeline tests the machine learning model using a separate dataset. The responsibilities of the testing pipeline include:

- Loading the best model checkpoints from the training pipeline.
- Evaluating the performance of the model using the test dataset.
- Logging the evaluation metrics, like precision, recall, and F1 score.

4. **Model inference pipeline:**

The model inference pipeline is responsible for deploying the trained machine learning model to production and predicting new data. The responsibilities of the model inference pipeline include:

- Building an API endpoint using Flask or FastAPI that can receive clinical notes as input and return the identified Named Entities as output.
- Deploying the API on a serverless computing platform, like AWS Lambda or GCP Cloud Functions.
- Scaling the API according to the incoming traffic.

5. **Model monitoring pipeline:**

The model monitoring pipeline monitors the deployed machine learning model's performance and detects drift or degradation. The responsibilities of the model monitoring pipeline include:

- Collecting real-time logs of the API endpoint, including input data and predicted output.
- Comparing the predicted output with the ground truth to detect any drift or degradation.
- Alerting the data science team if drift or degradation is detected, so they can take corrective measures.

Learners need not answer in the same manner as given above, but the crux of their answers should be the same. Any other answers are also acceptable.

Q4. **System design:** After creating the architecture, please specify your reasons for choosing the specific tools you chose for the use case.

NOTE: The company is a series-A startup.

Answer:

The main offering is not related to ML or it is not trying to provide ML solutions to any company. Therefore, it is evident that it does not have to create its own tools on-prem.

Data pipeline:

For the data pipeline, we chose Apache Apark because it is a distributed streaming platform that provides scalable and fault-tolerant messaging between applications. It can handle large volumes of real-time data and is suitable for processing continuous data streams. We also chose Data Version Control (DVC) as an open-source version control system for data science and machine learning projects. It enables reproducibility and collaboration by tracking changes to data, models, and code. With DVC, we can version our datasets and track changes to our data pipeline, ensuring reproducibility and traceability.

Here is a component level breakdown:

- Data storage: AWS S3 bucket
- Data processing: Apache Spark
- Data versioning: DVC
- Data cleaning and pre-processing: Pandas and Numpy

Training pipeline:

For the training pipeline, we chose scikit-learn with DVC for versioning machine learning models and data sets. It provides a way to track changes to machine learning models and data sets over time, making it easier to reproduce experiments and collaborate with team members.

Here is a component level breakdown:

- Machine Learning Framework: Scikit-learn
- Model Versioning: DVC
- Model Tuning: Optuna
- Model Training: Docker containers with GPU support
- Model packaging: Docker

Testing pipeline:
For the testing pipeline, we chose Jenkins, a popular open-source tool that provides continuous integration and continuous delivery (CI/CD) capabilities. It enables automated builds, testing, and deployment of software applications, which is crucial in ensuring that the ML model performs well in a production environment. Additionally, it can be integrated with other tools like GitHub for version control and AWS for cloud deployment.

Here is a component level breakdown:

- Continuous Integration: Jenkins or Travis CI
- Unit Testing: Pytest
- Code Quality and Linting: Pylint, Flake8

Model inference pipeline:

For the model inference pipeline, we chose Flask as it is a lightweight web framework that provides a RESTful API for serving ML models. It can be easily integrated with AWS for deploying the trained model, and can handle multiple requests simultaneously. Additionally, it provides flexibility in choosing the programming language for writing the API code.

Here is a component level breakdown:

- Web Framework: Flask or FastAPI
- Model Serving: Docker containers with GPU support
- API Gateway: AWS API Gateway
- Load Balancing: AWS Elastic Load Balancing
- Auto-scaling: AWS Auto Scaling

Model monitoring pipeline:
For the model monitoring pipeline, we chose Prometheus and Grafana as they are popular open-source tools for monitoring and visualizing system metrics. Prometheus provides a time-series database for storing metrics data, while Grafana provides a user-friendly

interface for visualizing the data. Additionally, both tools can be integrated with other systems like Kubernetes and Jenkins for automated monitoring and alerting.

Here is a component level breakdown:

- Monitoring Framework: Prometheus or ELK Stack
- Dashboard: Grafana
- Alerting: Prometheus Alertmanager or AWS CloudWatch Alarms
- Log Management: AWS CloudWatch Logs or ELK Stack

The specific tools were chosen based on their popularity, ease of use, and scalability for the given use case.

NOTE: This is an exhaustive list, the learners need to provide a more detailed explanation and breakdown. A broader idea for the tools in each pipeline and their respective reason is what we expect.

Q5. **Workflow of the solution:**

You must specify the steps to build such a system end to end.
The steps should mention the tools used in each component and how they are connected to solve the problem.
Broadly, the workflow should include the following:

- Data and model experimentation
- Automation of data pipeline
- Automation of training pipeline
- Automation of inference pipeline
- Continuous monitoring pipeline

Answer:

Here are the steps to build an end-to-end ML system for the given use case:

Data and Model Experimentation:
a. Collect and prepare the data: The first step would be to collect clinical notes data from the BeHealthy platform. The data should be cleaned and preprocessed for model training. Apache Spark can be used to preprocess the data.
b. Develop a CRF model: Next, a CRF model can be developed using Scikit-learn. The model can recognise the Named Entity in the medical dataset, such as diseases and treatments. The model should be versioned using DVC.
c. Experiment Tracking: MLflow can track the CRF model's performance metrics during experimentation.

Automation of Data Pipeline:
a. Data Version Control: DVC can be used to version the input data, preprocess scripts, and processed data.
b. Data Preprocessing: Apache Spark can preprocess the data in parallel and distribute it across multiple machines.
c. Data Storage: The processed data can be stored in a cloud-based service like AWS S3.

Automation of Training Pipeline:
a. Model Version Control: DVC can version the CRF model, its dependencies, and configuration files.
b. Hyperparameter Tuning: Optuna can be used to tune the hyperparameters of the CRF model.
c. Model Training: Docker containers with GPU support can be used to train the model in parallel.
d. Model Packaging: The trained model can be packaged into a Docker container.

Automation of Inference Pipeline:
a. Model Deployment: Kubernetes can deploy the model in a containerised environment.
b. API Development: Flask or FastAPI can be used to develop the API that takes the input data, preprocesses it, and runs the trained model to predict the diseases and treatments.

Continuous Monitoring Pipeline:
a. Log Monitoring: Elasticsearch and Kibana can monitor the application logs generated by the inference pipeline.
b. Model Performance Monitoring: Prometheus and Grafana can monitor the model's performance metrics, such as inference latency and accuracy.
c. Alerting: Alertmanager can alert the relevant stakeholders if performance metrics breach the defined thresholds.

Following these steps, we can build an end-to-end ML system for the given use case.