Here's a structured **ML Engineering Internship Interview** environment with questions covering **core ML engineering concepts** while ensuring they align with the expected knowledge level of an internee.

**📌 ML Engineering Internship Interview Questions**

**📂 Section 1: Machine Learning Fundamentals**

**What is the difference between Supervised, Unsupervised, and Reinforcement Learning?**

Ans: In supervised learning, the label is given and model is trained based on the dataset, e.g., regression, computer vision models, signals classification or simulation models, text to text, text to image, LLMs when used in a fine-tuning scenario, AI models of daily tasks.

Unsupervised learning models are those who’ve no label to be trained to predict on, such as k means clustering, autoencoders, PyCaret, PCA etc.

Reinforcement learning is like our brains, any RL model is trained in an environment, simulated computationally, it learns from it and get rewarded for making right prediction or choice and if it doesn’t then it learns from the environment in which it’s working, all this happens via feedback, which is nothing but just the behaviour on which it is trained, or some simulations of this purpose involves or has the ability to involves human interaction/intervention as well such as OpenAI’s gymnasium. Its example includes self-driving cars, chess playing bot, AI farmers etc.

**Explain the bias-variance tradeoff in machine learning.**

Ans: The bias-variance trade-off in machine learning describes the inherent tension between a model's tendency to make simplifying assumptions and its sensitivity to fluctuations in the training data.

Bias represents the error introduced by a model's overly simplistic assumptions about the underlying relationship in the data. A high-bias model often fails to capture the complexity of the real-world problem, leading to underfitting, where the model performs poorly on both the training and test datasets.

Variance, conversely, reflects the model's sensitivity to small variations or noise within the training data. A high-variance model attempts to fit the training data too closely, including its random fluctuations, resulting in overfitting. This leads to excellent performance on the training data but poor generalization to unseen data.

The goal of the bias-variance trade-off is to find the optimal balance between these two sources of error. A model with too much bias will miss important patterns, while a model with too much variance will memorize noise. Achieving this balance involves selecting a model that is complex enough to capture the true underlying patterns without being overly influenced by the noise in the training data, leading to a model that generalizes well to new, unseen data.

**What is overfitting and underfitting? How can you prevent them?**

Ans: I know the answer very well.

**How does cross-validation work, and why is it used?**

Ans: I know the answer very well.

**What is the difference between batch gradient descent, stochastic gradient descent, and mini-batch gradient descent?**

Ans: Gradient descent is a fundamental optimization algorithm used to train machine learning models by iteratively minimizing a loss function. The three primary variants—batch, stochastic, and mini-batch—differ in how they utilize the training data to compute the gradient and update the model's parameters.

Batch gradient descent (BGD) calculates the gradient using the entire training dataset in each iteration. This approach guarantees convergence to the global minimum for convex loss functions and provides a stable gradient. However, it's computationally expensive and slow, especially with large datasets, as it requires processing all training examples before each parameter update.

Stochastic gradient descent (SGD), in contrast, calculates the gradient using only a single, randomly selected training example per iteration. This makes it significantly faster than BGD, particularly for large datasets, and allows it to escape local minima. However, the noisy updates can lead to oscillations and slower convergence, and it might not converge to the exact minimum.

Mini-batch gradient descent (MBGD) strikes a balance between BGD and SGD. It calculates the gradient using a small, randomly selected subset (mini-batch) of the training data in each iteration. This method is faster than BGD and more stable than SGD, efficiently utilizing vectorized operations for faster computation. While it introduces some noise into the updates, it's less than SGD, and it requires tuning the mini-batch size. Due to its efficiency and stability, mini-batch gradient descent is the most commonly used variant in practice.

**Why do we use feature scaling in ML models? Explain Min-Max scaling vs. Standardization.**

Ans: I know the answer very well.

**What is a confusion matrix, and how do you calculate precision, recall, and F1-score?**

Ans: A confusion matrix is a table that visualizes the performance of a classification model, especially useful when dealing with imbalanced datasets (where one class significantly outnumbers the other). It provides a detailed breakdown of correct and incorrect predictions, enabling a more nuanced evaluation of model performance beyond simple accuracy.

**Confusion Matrix Structure (Binary Classification):**

In a binary classification scenario (positive and negative classes), the confusion matrix consists of four key components:

* True Positive (TP): The model correctly predicted the positive class.
* True Negative (TN): The model correctly predicted the negative class.
* False Positive (FP): The model incorrectly predicted the positive class (a Type I error).
* False Negative (FN): The model incorrectly predicted the negative class (a Type II error).

**Calculating Precision, Recall, and F1-Score:**

These metrics, derived from the confusion matrix, provide a more detailed understanding of the model's performance:

* Precision: Measures the proportion of predicted positive cases that were actually positive (TP / (TP + FP)).
* Recall (Sensitivity): Measures the proportion of actual positive cases that were correctly predicted (TP / (TP + FN)).
* F1-Score: The harmonic mean of precision and recall, providing a balanced measure (2 \* (Precision \* Recall) / (Precision + Recall)).

To calculate these metrics in Python, you can use the sklearn.metrics library:

* `from sklearn.metrics import precision\_score, recall\_score, f1\_score` for individual metric calculations.
* `from sklearn.metrics import classification\_report` for a comprehensive report including precision, recall, F1-score, and support (number of samples) for each class.

**Why do we use train-test splits, and what is the typical ratio?**

Ans: I know the answer very well.

**What are common activation functions in deep learning? Why is ReLU preferred?**

Ans: Activation functions are essential components of deep learning models, introducing non-linearity that enables networks to learn complex patterns. Among the most common are the sigmoid and tanh functions, historically used for binary classification, but they suffer from the vanishing gradient problem and slow training.

The rectified linear unit (ReLU) has emerged as the most widely used, especially in hidden layers, due to its computational efficiency, ability to mitigate vanishing gradients for positive inputs, and promotion of sparsity. Though it can encounter the "dying ReLU" problem (neurons can become inactive if their inputs are consistently negative), variants like Leaky ReLU address this.

For multi-class classification, the softmax function is typically used in the output layer to convert raw scores into probability distributions. ReLU's speed, gradient behavior, and proven effectiveness make it the preferred choice for many deep learning applications.

**Explain the difference between L1 and L2 regularization.**

Ans: Both L1 and L2 regularization techniques are employed to mitigate overfitting in machine learning models.

The primary distinction between them lies in how they handle the features of a dataset. L1 regularization, also known as Lasso, tends to drive the coefficients of less important features to exactly zero, effectively performing feature selection and resulting in a sparse model where only the most influential features are retained.

In contrast, L2 regularization, or Ridge, reduces the magnitude of all feature coefficients, pushing them towards zero but rarely setting them precisely to zero. 3 This approach is particularly useful in scenarios involving multicollinearity, where features exhibit high correlation with the target variable, as it helps stabilize the model without eliminating features entirely.

**💾 Section 2: Data Handling & Processing**

**How would you handle missing values in a dataset?**

Ans: Following ways are there to do the job

* Imputation
* Dropping the column with most of the NaNs.
* Dropping the row with most of the NaNs.

**What are some ways to deal with categorical variables in ML models?**

Ans: These are:

* Using `OneHotEncoding`
* Using `LabelEncoding`
* Using `pandas.Categorical`
* Using `pandas.get\_dumps`
* Using `df[‘another\_column’] = df[‘categorical\_column’].map(DICT\_FORMAT)`

**What is the difference between an imbalanced and balanced dataset? How would you handle class imbalance?**

Ans: I know the answer.

**Explain the difference between one-hot encoding and label encoding.**

Ans: There difference lies in their approach of encoding the categories. The one-hot encoding does it by making a boolean columns for the total categories of the respective dataset. This is however, expensive in terms of dimensionality, its another equivalent is `pndas.get\_dummies`.

In case of label encoding, it encodes the categories into a numeric format in order for an ML model to be applied on the dataset. This doesn’t directly lead to addition of another column in a dataset to be worked with, but most data scientist and analysts remove the original categorical column and then add the new column containing the encoded categories. This is equivalent of `df[‘another\_column’] = df[‘categorical\_column’].map(DICT\_FORMAT)`.

**How would you preprocess text data for machine learning models?**

Ans: The pre-processing of text data includes:

* Cleaning like html tags, numbers, utf-8 encoding to handle Unicode characters.
* Tokenizing of words into a single token using like word-tokenizer, sentence-tokenizer,
* Normalizing (Stemming, Lemmatizing)
* Feature extraction using vectorizing (Bag of words, TF-IDF) and embedding (BERT, llama, Sentence-BERT); though embedding can be that of a domain specific NLP data as well.
* Handling of OOV (Out of Vocabulary) words by replacing them with `<UNKNOWN>` token or using sub-word tokenization by breaking compound words into single words and then tokenizing it.
* Dimensionality reduction like PCA, SVD.
* Text-sequencing for sequence-based model, specifically in deep learning neural networks e.g., LSTM, GRU, RNN using padding (adding for missing token) and truncating (removing tokens when they exceed maximum length).

**What is the difference between normalization and standardization?**

Ans: I know the answer

**If you are working with a dataset that has a large number of features, how would you perform dimensionality reduction?**

Ans: My approach would contain:

* Try to grab the context of the dataset’s feature by using the data-dictionary or my own understanding of the domain, or search for the domain knowledge of the dataset at hand.
* Use `df.corr()` to find the correlation and henceforth the impact of the fields/dimension on the label and then remove the less correlated dimensions.
* If the fields are of categorical variables, then chi-square test is used to determine effects of fields on the label.
* Use model’s own built-in functions to determine the most relevant features, such as xgboost’s `feature\_score`.
* Use dimensionality reduction methods like PCA, SVD, LDA, t-SNE.
* Use feature engineering techniques like auto-encoder, pycarrot.

**What are outliers? How would you detect and handle them?**

Ans: Outliers are those values of the dataset that are not of the same range in which most of the dataset’s values are, either being too low or too high.

Outliers can be detected using IQR method where inter-quartile range is used. Other than that, boxplots can be used, which uses IQR ranges. Boxplot’s upper whisker show extremely large values from the dataset’s other values, while the boxplot’s lower whisker shows extremely low values from the dataset’s other values.

The handling of the outliers depends upon the purpose of the analysis. In my experience with clustering, use the outliers to your advantage by not discarding them totally. However, you can also go for winsorization of the outliers, where outliers are managed in such a way that they’re not removed, but integrated into the dataset.

**What are some common feature engineering techniques?**

Ans: Following techniques are mostly used:

* Creation of new features from existing features.
* Working with categorical data.
* Normalizing, standardization of numerical data.
* Vectorizing, embedding of NLP data.
* In time series, non-stationary series conversion to stationary by using rolling, lag features and also smoothing the time series by using techniques like exponential-smoothing, moving average, Kalman-filter, lowess smoothing, savitzky-golay filter.
* Dimensionality reduction methods to get the highly relevant dimensions as per their impact using methods like PCA, LDA, SVD, t-SNE.
* Employ autoencoders to learn non-linear feature representations.
* Use tools like PyCaret for automated feature engineering, but always check the outputs, and understand what the tool is doing.

**Explain the role of data augmentation in ML models.**

Ans: The role of data augmentation is to add the variability and diversity either in already diverse dataset or in a non-diverse, or a small dataset. The role of this is to expand the dataset, or even add real-life scenarios-based data within the dataset e.g., adding audio modulated sounds in an audio dataset, synthesizing text-data based on the pre-existing text dataset. There’re python libraries to do this job, as per the data type to be work with or at hand.

**⚙️ Section 3: Machine Learning Model Deployment & MLOps**

**What is the purpose of model serialization, and what formats are commonly used?**

Ans: I know the answer very well.

**What is the difference between Flask and FastAPI for deploying ML models?**

Ans: The difference is based on the additional capability of FastAPI to do asynchronous programming as well. Both of these frameworks are popular for making REST API for web apps, but FastAPI’s asynchronous capability allows to make asynchronous calls to API operations, thus enabling the web app for multi-users with multi-operation capability. The same can be done in flask, but not with ease, you’ve to use python’s concurrent capability to add the same capabilities in the flask app.

**How do you monitor a deployed ML model for performance degradation?**

Ans: To ensure a deployed machine learning model remains accurate and useful, continuous monitoring is essential. First, we need to decide what 'good performance' looks like by defining key metrics. For example, if we're predicting customer purchases, we might track accuracy, how often we're right, and latency, how quickly we make predictions. These metrics, along with logs of what the model predicts and what data it uses, are gathered by monitoring tools like Prometheus. We then use visualization tools, such as Grafana, to create dashboards that show these metrics in real-time. These dashboards help us quickly spot any sudden drops in performance or unusual patterns.

Next, we need to watch for changes in the data the model is using, a problem called 'data drift,' and changes in the relationships the model has learned, 'concept drift.' We use statistical tests to detect these shifts, and set up alerts to notify us when significant changes occur. If we see a drop in performance or detect drift, we need to retrain the model with fresh data to bring it back up to speed. We also use techniques like A/B testing, where we compare a new model version with the old one, to ensure any updates are actually improvements.

Finally, we create a feedback loop, incorporating user feedback and business metrics, to continuously refine and improve the model. This process ensures our ML models remain reliable and effective in a constantly changing environment.

**What is CI/CD in the context of ML Engineering?**

Ans: CI/CD in ML Engineering automates the ML pipeline for rapid and reliable model updates.

Continuous Integration (CI) focuses on automated testing of code changes, including unit, integration, and model validation tests. It integrates with version control, automates builds, and facilitates early error detection using tools like Jenkins and GitLab CI.

Continuous Delivery/Deployment (CD) automates model deployment to production, leveraging Infrastructure as Code for environment provisioning. It includes model monitoring with tools like Prometheus and Grafana, and enables automated rollbacks using strategies like blue/green and canary deployments, often on cloud platforms like AWS and GCP.

ML-specific considerations within CI/CD involve automated data validation, model validation, and model versioning to ensure data quality and model performance. Experiment tracking with tools like MLflow and ensuring deployment reproducibility are also crucial.

In summary, CI/CD in ML Engineering automates the entire ML pipeline, from code changes to model deployment and monitoring, enabling rapid iteration, improved model reliability, and efficient productionization.

**What is model drift, and how would you handle it?**

Ans: Model drift, which encompasses both data drift and concept drift, is a phenomenon where the statistical properties of input data or the relationship between features and the target variable shift over time, leading to a decline in model performance. Data drift specifically refers to changes in the distribution of the input data, while concept drift describes the evolution of the underlying relationships the model is designed to learn. These shifts are often triggered by external factors such as economic fluctuations, climate changes, evolving market trends, or other dynamic conditions.

To effectively address model drift, a proactive strategy involving continuous monitoring and adaptive retraining is essential. First, a robust monitoring system should be implemented to continuously evaluate the input data distribution and feature relationships using statistical tests, such as the Kolmogorov-Smirnov test and Kullback-Leibler divergence. This system should also track vital model performance metrics, including accuracy, precision, recall, and F1-score, and visualize model behavior to detect any anomalies. Additionally, external data sources, like market condition indicators, must be monitored to identify potential external influences.

Once drift is detected, based on predefined thresholds for acceptable deviations, automated alerts should be triggered. Following detection, a data versioning process should be initiated to capture the changes in the dataset. The model should then be retrained using the most recent and relevant data to adapt to the updated data distributions and feature relationships.

The retraining frequency should be determined by the observed rate of drift and the criticality of the model's performance. Furthermore, exploring model architectures that are robust to drift, and implementing feature engineering techniques to create features that are less sensitive to changes, should be considered. Finally, establishing a process for human intervention, in some cases, may be required to fully understand and address drift events. This comprehensive approach ensures timely detection and mitigation of model drift, maintaining the model's accuracy and reliability in a dynamic environment.

**What are Docker and Kubernetes, and why are they used in ML deployment?**

Ans: I know the answer

**Explain the concept of feature stores in MLOps.**

Ans: Feature store in MLOps is layer in which features of an ML model are stored, managed and maintained.

It acts as a centralized repository, and it is a purpose-built database, that addresses the complexity and challenges of feature engineering and serving in production ML environments. It ensures consistency, reproducibility, and efficiency in feature handling throughout the ML lifecycle.

There are libraries like Feast, Tecton, Hopswork. The cloud solutions for this purpose includes Azure Machine Learning Studio Feature Engineering, AWS SageMaker Feature Engineering, GCP Vertex AI Feature Store.

Some of the consideration while creating/working with this layer of MLOps architecture include are:

* Cloud vs local feature store as per your need, cloud will be more robust and easier to manage.
* Real time vs batch processed features, as per the model’s requirement.
* Data integration with the feature store (Data warehouse, Data Mart, Data Lake, Data lakehouse, Database(Relational, Non-relational)).
* Check the integrability and compatibility of the ml framework at hand (PyTorch, TensorFlow, sklearn, XGBoost) with the feature store to be used.

**What is an API, and how would you expose an ML model via an API?**

Ans: I know this very well!

**What are the advantages of using cloud platforms (AWS/GCP/Azure) for ML model deployment?**

Ans: I know the answer.

**How does logging and monitoring help in production ML systems?**

Ans: Logging and monitoring are essential for the reliability and continuous improvement of production ML systems. By capturing key metrics, real-time events, and causal precursor chains, logging provides crucial performance indicators, highlighting potential model failures and deviations from expected standards. Specifically, logs detailing failure events or out-of-context predictions, particularly in reinforcement learning where agent understanding is critical, are vital for error analysis.

Monitoring systems then analyze this logged data, alongside other real-time information, to provide continuous performance assessments and detect deviations, triggering alerts for proactive intervention.

This combined data facilitates data-driven model optimization, including hyperparameter and reward function tuning in reinforcement learning, ensuring that the model consistently meets stakeholder expectations and performs optimally in production.

**🛠️ Section 4: Programming & Practical ML Engineering**

**Write a Python function to implement gradient descent for a simple linear regression model.**

Ans: The python function for this purpose is as under:

```python

import numpy as np

from numpy.typing import ArrayLike

from sklearn.metrics import mean\_square\_error

def linear\_reg\_grad\_desc(

x: ArrayLike,

y: ArrayLike,

learning\_rate=0.01,

epochs=1000

):

n = len(x)

errors = []

slope = 0

intercept = 0

for \_ in range(epochs):

y\_pred = slope \* x + intercept

error = mean\_square\_error(y, y\_pred)

errors.append(error)

d\_slope = (-2/n)\*np.sum(x\*(y – y\_pred))

d\_intercept = (-2/n)\*np.sum(y - y\_pred)

slope = slope – learning\_rate \* d\_slope

intercept = intercept – learning\_rate \* d\_intercept

return slope, intercept, errors

```

**What is vectorization in NumPy, and why is it faster than loops?**

Ans: Vectorization in NumPy allows operations to be performed on entire arrays at once, without explicit Python loops. This is significantly faster because Python loops have an O(n) time complexity, where 'n' is the array size. NumPy's vectorized operations, being implemented in optimized C code, achieve a much lower complexity, though not strictly O(1), by avoiding the Python interpreter overhead and leveraging efficient, low-level computations. This results in substantial speedups for numerical operations, which is crucial in machine learning.

**Explain the difference between a Pandas DataFrame and a NumPy array.**

Ans: NumPy arrays and Pandas DataFrames are fundamental but distinct data structures: NumPy arrays are optimized for efficient numerical computations on homogeneous, n-dimensional data, accessed via integer indices, making them ideal for mathematical operations and machine learning algorithms. In contrast, Pandas DataFrames are two-dimensional, labeled tables capable of storing heterogeneous data, offering powerful tools for data manipulation, cleaning, and analysis through both integer and label-based indexing, catering specifically to real-world datasets with diverse data types. Essentially, NumPy excels in numerical processing, while Pandas specializes in flexible data handling

**What are some efficient ways to handle large datasets in Python?**

Ans: Following ways can be used:

* Usage of chunking for large datasets, with `chuksize` of `k` in `pandas.DataFrame.read\_csv` or `pandas.DataFrame.read\_excel`.
* Try to use efficient distributed file systems like HDFS (Hadoop Distributed File System) when working with Apache Spark. Spark's distributed nature allows for efficient processing of data stored across a cluster. Additionally, efficient data formats like Parquet, which work well with HDFS, can significantly improve performance. HDF5 is also an excellent format, that excels when working with node based cluster computing.
* Usage of parallelization in terms of data fetching using library like `Dask`.
* If the dataset is too large, then usage of any database, cloud would be highly recommended.
* If the dataset has too much features, then do dimensionality reduction, or feature engineering.

**Write a Python function to remove duplicate rows from a Pandas DataFrame.**

Ans: Use `df.drop\_duplicate` to remove duplicate from a pandas’ DataFrame.

**How would you optimize an ML model for inference speed?**

Ans: I’ll do:

* By using model serialization so that model doesn’t need to be run again and again just to `fit ` it on certain data and then make predictions out of it then.
* By using caching or memorization in any function which uses model for inferencing.
* By using resource friendly models, if possible.
* By using efficient format of data so that the model gets trained and then get loaded easily to be used as input for inferencing.

**What are the differences between PyTorch and TensorFlow?**

Ans: In my own experience with both of these models, I found that PyTorch models are easy to debug and the training can be further enhanced by using related libraries like `pytorch\_lightning`. In case of TensorFlow, the training is very similar to any ML model like `model.fit` and `model.predict`, but creation of deep learning network is complex, but these .keras model can be easily used on mobile apps as well using `tflite`.

**What are the advantages of using JIT compilation (e.g., TorchScript) in PyTorch?**

Ans: The benefit of using JIT(Just in Time) in PyTorch gives the following benefits:

* Model serialization
* Model portability
* Resource flexibility (GPU, TPU)
* Model exposure
* Low-level conversion to make it fast to interpret by python.

**How does multiprocessing work in Python, and how can it help in ML tasks?**

Ans: The concept of multi-processing in python is of dividing a process into threads. This can help in ML tasks by dividing and parallelization of independent tasks in EDA, data pre-processing

**What is the purpose of a requirements.txt file in an ML project?**

Ans: The purpose of this file is to facilitate the installation of the required packages of the python in a dockerized python app.

**🤖 Section 5: Model Evaluation & Hyperparameter Tuning**

**What is Grid Search and Random Search? When would you use one over the other?**

Ans: Grid search and random search are two sklearn based hyperparameter tuning classes which are used to find the best hyperparameters for the model at hand.

The random search works by randomly making a subset from a range of the given hyper-parameters and then gives the range of best hyperparameters. The grid search works by making all combinations, from the grid of the specified hyperparameters, and search among them all to find the most suitable one.

The previous one is not computationally expensive, but doesn’t necessarily yield the best hyperparameters, it is only meant to be used when you got the perfect idea of the range from which best hyperparameter can be yielded. Grid search on the hand, is used to provide the best performing hyperparameter because of its approach of searching among different combination of the provided parameters and then giving the best results.

Speaking of recommendation among these two in terms of use case and usage, at first, use random search, from the best result yielding range and once you’ve found a particular combination of best hyperparameters, use that as input to grid search; grid search will yield the best from that range. However, if you know very well that random search has already given the best hyperparameters then there’s no need for grid search; because, in such case, the grid search output may not be that different from the input range. Also, before going back and forth between these two, try manually to find the best result acquiring hyperparameters on the model of the concern.

**Explain the difference between validation loss and test loss.**

Ans: Validation loss occurs when the loss occurs during training phase on the validation dataset. This loss helps in further refining the hyperparameters. The rationale behind validation loss is to facilitate early stopping, prevent overfitting by checking the training results on the dataset on which it’s not directly trained on.

The test validation is done after the training phase. This is done to check the model’s performance on the unseen data. Thus, simulating a real-life scenario to assess the model’s quality. There’re several metrics to get the test loss.

**How do you evaluate a regression model? Name at least three metrics.**

Ans: The evaluation of regression is done using the percentage of accuracy score, and error threshold using test loss functions. The test loss functions in `sklearn.metrics` is typically used for the model’s evaluation. These are as under:

* MSE
* MAE
* RMSE

**What are hyperparameters, and how do they differ from model parameters?**

Ans: Hyperparameters are those parameters of the model which effect the model’s performance, and are settled before the model training. Model parameters are those which are learned by the model via being trained on training data and then outputted.

**What are some common techniques for tuning hyperparameters?**

Ans: The most common techniques of acquiring a proper hyper-parameter are:

* Grid Search
* Random Search
* Manually setting the hyper-parameters and checking them for best fit.

**What are ROC and AUC, and why are they important?**

Ans: ROC (Receiver Operating Characteristics) is all about the TPR (True Positive Ratio) against the FPR (False Positive Ratio), which displays the ratio of accurately predicted against the wrongly predicted classes. The closer the curve of these two are, the better the model is.

AUC (Area Under the Curve) is a singe value to represent the overall summary of the binary classifier. It represents the area under the ROC. The AUC value is between 0 to 1, a classifier with AUC of 0.5 is no better than making random guesser.

Both of these are metrics which are used in binary classifiers, be it of deep learning or ML model. The AUC tells the accuracy of the model in terms of the area which is the confidence region while ROC tells the ratio of the accurate prediction vs wrong predictions.

**What is early stopping, and how does it help in model training?**

Ans: The early stopping is used to stop model from training further because a certain metric, say val\_loss, auc etc are not improving. This prevents the resources from being used further for such model.

**How does k-fold cross-validation improve model evaluation?**

Ans: The k-fold cross validation works by making k-subsets of the data which’re nothing but the shuffled-up rows of the same dataset, and pick the row from them all, give input to model to test. The benefit and rationale are to check the model’s accuracy not only on train and test data but also on real life scenarios as well, because these k-fold simulate the real-life scenario based upcoming of the data very well.

**How would you choose between a decision tree and a random forest for a classification task?**

Ans: When choosing between a decision tree and a random forest for a classification task, the decision hinges on several factors related to the dataset and the desired model characteristics. Decision Tree is a ML model for classification of linearly related features of data, random forest is an ensemble model which is used for both classification and regression-based prediction, on no-linearly, complexly related features of data .

A decision tree is a single tree-based model that partitions the data based on feature values, making it relatively simple and interpretable. However, decision trees are prone to overfitting, especially with complex datasets, leading to poor generalization. They tend to perform well on structured, relatively simple data where clear decision boundaries can be defined.

A random forest, on the other hand, is an ensemble method that combines multiple decision trees. By aggregating the predictions of these individual trees, random forests significantly reduce overfitting and improve generalization. They are robust to noise and can handle complex datasets with high dimensionality. Random forests are particularly effective when dealing with data that may have complex, non-linear relationships between features and the target variable.

For random forest classification, the data on which it is implied is usually of complexly related features, e.g., a tabulated form of a non-structured data. The non-structured data could be image data from faces images for instance, the extracted features could be of eye colour, facial colour, skin colour, hair colour, face type and vice versa. The label can be the gender. The extracted features can be acquired by sam, sam2 using ultralytics.

**What is the effect of increasing or decreasing the number of hidden layers in a neural network?**

Ans: That totally depends on the context of data and the architecture at hand. Generally speaking, too much from what is required could lead to overfitting, leaving less room for generalization, also too complex of a neural network yielded via adding more neural network could lead to gaining faulty patterns, removing irrelevant layer(s) in this case can be beneficial. In case of removing the layer due to the cost of computation, do consider the consequences of the layer removal, try to get the full context/understanding of the data and then design the neural network architecture of yours. For example, in classification of the facial expression of the faces as per my experience, 4 layers of conv2d neural layers are enough because each layer will identify the features of the faces from the images. If you reduce the layers then the facial feature won’t be acquired; however, if you add more layers then the image background related features may be captured, which is of no use.

**🧠 Bonus: Case Study Questions**

**You built an ML model with 95% accuracy, but your business stakeholders are not happy with its performance. What could be wrong?**

Ans: The following could be the reasons:

* The 95% accuracy could be an indicator of the overfitting. The 95% accuracy over the training data isn’t an indicator of having same results over the real-life data.
* The metric consideration is also importance, accuracy shouldn’t be considered only. Other metrics like precision, recall, f1-score etc are to be considered as well, or, business specific ones like KPIs.
* There could be issues in dataset as well, it could be an imbalanced one, the old ones which doesn’t align with the real-world data distribution, thus resulting in data drift.
* There could be issue in data pipeline, its scalability and deployment issue and also of deployed model on the cloud.
* The stakeholders had different expectations which were not fulfilled by the model, this issue arises when the elicited requirements are not commuted properly.

**You deployed a machine learning model, but after a few months, its performance degrades. What are possible reasons?**

Ans: The issue could arise due to following factors:

* The data related issues include data drift; the change in data’s distribution, data concept drift; the change in independent and dependant variables/fields, data quality issue; the lack of new to train the ML model at hand.
* The new architecture of a software could also lead to making errors, this could be resulted by interruption of a pre-processing pipeline within the software, thus making the ML model ineffective.
* Model staleness could also lead to this. Model staleness refers to the model’s inability to identify diverse, hidden patterns and insights to make predictions in a comprehensive manner, e.g., a recommendation system which recommends the products of the same company from which a purchase was made.
* When a model is based on certain feedback to create its results more accurate, it becomes stuck in a feedback loop.

**Your deep learning model performs well on training data but poorly on test data. What could be the issue?**

Ans: This is such case of overfitting which has its roots in the dataset on which the model is being trained and tested. This is possible due to the dataset either being too small, or having irregular distributions. The dataset could potentially be imbalanced one as well, thus resulting in high accuracy on training set and poor performance on test set.

**Your ML pipeline takes too long to train. How would you optimize it?**

Ans: The following methods could be of use in this case:

* Try to create a resource friendly ML pipeline, by making a computation friendly ml model.
* In case of making batches of data from the training dataset, do not make too big batches for a model to train on.
* Try to use efficient format like parquet, hdfs for heavy datasets.
* Use parallelization to execute the tasks in a pipeline, thus leveraging the CPU to its full extent.
* Use caching of data within the ML pipeline to prevent the resources from doing the computation of retrieving the dataset batch.