5G Traffic Prediction Using Machine Learning and Cloud Integration

Amisha Lalwani(202251013)
Aradhya Verma(202251022)
Dhwani Saliya(202251041)
Guarav Barhate (202251049)

Introduction

With the exponential growth of connected devices, high-definition streaming, and low-latency applications such as autonomous vehicles, remote surgery, and industrial IoT, 5G networks have become a foundational technology for modern communication systems. The efficient management of network resources — including bandwidth, signal strength, and latency — is critical to maintaining seamless Quality of Service (QoS) and meeting the stringent requirements of real-time and bandwidth-sensitive applications.

In 5G architecture, dynamic traffic load prediction plays a pivotal role in enabling network operators to make proactive decisions for resource allocation, reduce congestion, and enhance user experience. Traditional rule-based or static allocation methods often fall short in handling fluctuating network conditions in real time. To address this challenge, our project proposes a machine learning-based predictive model using Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN) specially suited for time-series data and long-term pattern recognition.

The goal of this project is to train a robust LSTM model that learns from historical 5G QoS data and accurately predicts future resource allocation percentages, which reflect the current and anticipated network load. The model leverages various input features such as signal strength, latency, application type, required bandwidth, and allocated bandwidth. These features provide valuable insights into how different applications and devices behave under varying network conditions.

To make this solution scalable and accessible, we also deploy the trained LSTM model on Amazon Web Services (AWS). By integrating our predictive engine into a cloud-based infrastructure, we enable real-time inference capabilities that can be consumed via a REST API by edge devices, applications, or network controllers. AWS services like EC2, S3, and Flask API hosting are used to facilitate model serving, data management, and prediction at scale.

This project not only showcases the power of LSTM in forecasting 5G network behavior but also highlights the real-world applicability of deploying machine learning models in a cloud-native architecture. The outcome is a system that can help telecom operators optimize their network performance, anticipate high-demand periods, and ensure consistent QoS across diverse user demands and application types.

In essence, our work combines deep learning, network traffic analysis, and cloud computing to build an intelligent traffic prediction system for 5G networks — one that is scalable, responsive, and practical for modern smart network environments.

I. Project Objectives

The overarching objective of this project is to explore and implement an artificial intelligence-based approach for optimizing resource allocation in 5G networks. By utilizing Quality of Service (QoS) metrics, the project leverages deep learning models to predict and manage the allocation of bandwidth and other critical resources for diverse applications. The project accomplishes this by training a neural network model on real-world QoS data from 5G environments, enabling it to identify patterns in signal strength, latency, bandwidth usage, and application types to forecast necessary resource distributions.

1.1 Understanding Network Dynamics

5G networks are characterized by their ability to handle a high density of devices, ultra-low latency requirements, and enhanced mobile broadband services. This project aims to gain a deep understanding of how key metrics like signal strength, latency, and bandwidth correlate with optimal resource allocation. It uses historical data to develop models that simulate intelligent, autonomous decision-making.

1.2 Predictive Resource Management

Traditional rule-based resource allocation techniques fall short in real-time dynamic environments. This project seeks to introduce AI-driven prediction models that can proactively determine how resources (such as bandwidth and

signal power) should be allocated to different users or applications. By predicting resource allocation, the model can help network systems prepare for demand fluctuations and performance drops before they occur.

1.3 Enhancing Quality of Experience (QoE)

One of the core objectives is to enhance the user experience by ensuring consistent service quality. This includes minimizing delays for real-time applications (e.g., video conferencing, gaming) and maximizing throughput for bandwidth-heavy services (e.g., streaming, file transfers). AI-driven optimization ensures that even under heavy load, critical applications receive prioritized and adequate resources.

1.4 Foundation for Autonomous 5G Systems

The project also aims to lay the groundwork for self-organizing 5G networks. These systems can manage themselves with minimal human intervention by making intelligent decisions using machine learning. The predictive modeling in this project is a step toward such fully autonomous and intelligent network systems.

II. Use Cases (Comprehensive and Real-World Focused)

The AI-based resource allocation model built in this project has a wide range of practical applications in the real-world deployment of 5G technology. Below are the detailed use cases:

2.1 Smart Cities and Urban Mobility

In smart cities, IoT devices such as traffic lights, pollution monitors, and autonomous vehicles must communicate in real-time. The model can be used to:

- Dynamically allocate bandwidth to prioritize time-sensitive data
- Ensure low-latency communication between autonomous vehicles and city infrastructure
- Prevent congestion in the network by preemptively reallocating resources

2.2 Healthcare and Remote Surgery

5G is critical in healthcare, particularly for remote patient monitoring and telesurgery. The model enables:

- Prioritized bandwidth allocation for real-time medical data streams
- Prediction and avoidance of latency spikes that could disrupt life-critical services
- Dynamic resource adaptation during peak hours in hospital networks

2.3 Industrial IoT and Automation

Factories using Industry 4.0 systems require real-time communication between machines, robots, and sensors. The AI model supports:

- Timely data exchange for precise automation control
- Adaptation to fluctuating network demand from various devices
- Ensuring uninterrupted machine-to-machine communication during production

2.4 Enhanced Mobile Broadband (eMBB) Services

For consumers using AR/VR, UHD video, or high-speed downloads, the system provides:

- Seamless service quality even during peak traffic hours
- Dynamic scaling of network resources to support multimedia-heavy tasks
- Reduction of video buffering and call drops

2.5 Network Management for Telecom Operators

Telecommunication companies can utilize the predictive model to:

- Forecast and mitigate network congestion
- Automate network slicing and spectrum allocation
- Reduce operational costs by improving resource efficiency

2.6 Emergency Response Systems

During natural disasters or emergencies, 5G networks must handle sudden surges in communication. This model can:

- Prioritize emergency signals and route them with high reliability

- Allocate resources to first responders and public safety systems
- Maintain robust communication channels when traditional networks fail

III. Technical Depth and Implementation Perspective

The project integrates multiple layers of technological and theoretical concepts:

- **Data Engineering:** Data preprocessing includes cleaning timestamps, encoding application types, normalizing signal strength and bandwidth, and converting units (e.g., Kbps to Mbps). This standardizes inputs for model training.
- **Feature Engineering:** Selection of key features such as latency, signal strength, application type, required and allocated bandwidth is vital for learning meaningful patterns.
- **Model Development:** A PyTorch-based neural network is used to perform regression, predicting resource allocation percentages from the input features. This includes defining the architecture, loss functions (e.g., MSE), optimizers (e.g., Adam), and training loops.
- **Evaluation Metrics:** While not explicitly detailed in the notebook, metrics like Mean Absolute Error (MAE) or Root Mean Square Error (RMSE) are typically used to evaluate the model's predictive accuracy.
- **Scalability & Future Directions:**
    - Incorporation of real-time data streaming for live predictions
    - Deployment on edge devices for low-latency inference
    - Extension of the model using Recurrent Neural Networks (RNNs) or Transformer architectures for temporal learning.

3.1 Introduction to LSTM

Long Short-Term Memory (LSTM) networks are a powerful type of Recurrent Neural Network (RNN) specifically designed to handle the problem of learning

long-range temporal dependencies in sequential data. Traditional RNNs struggle with the vanishing gradient problem, which makes learning difficult over long sequences. LSTMs, introduced by Hochreiter and Schmidhuber in 1997, address this challenge by incorporating a memory cell and three gating mechanisms—the input gate, forget gate, and output gate.

The memory cell in an LSTM acts as a conveyor belt that carries relevant information throughout the processing of a sequence. The input gate determines which new information should be added to the cell state, the forget gate decides what information should be discarded, and the output gate regulates what information is passed to the next layer. This architecture allows LSTMs to learn which parts of the sequence are relevant for long-term prediction, enabling the model to capture both short-term and long-term patterns in data.

Why LSTM Was Selected for This Project

In the context of 5G network resource allocation, data comes in the form of time series—metrics like signal strength, bandwidth utilization, and latency vary over time and are highly interdependent. These characteristics make LSTM an ideal choice for the project. The reasons are as follows:

Temporal Data Dependency

5G network performance indicators are inherently sequential. For example, a degradation in signal quality over a few seconds may lead to increased latency or dropped packets. LSTM networks excel at understanding such temporal patterns. They can learn how previous network conditions influence current and future performance, thereby enabling proactive resource allocation.

High-Dimensional Time Series Prediction

This project deals with multivariate time series data. Multiple features—such as application type, historical bandwidth consumption, and real-time signal quality—contribute to a prediction. LSTMs are capable of handling such high-dimensional data, learning correlations between variables and changes over time. This is essential for producing accurate and context-aware resource predictions.

Stability in Learning

Unlike simple RNNs, LSTMs are stable during training and are less prone to issues like gradient vanishing or exploding. This stability ensures consistent

model training, which is especially important when dealing with large-scale datasets from a 5G network, where convergence and reliability are critical for deployment.

Generalization Across Scenarios

LSTMs possess strong generalization capabilities. A model trained on a particular environment (e.g., urban traffic patterns) can adapt to different scenarios (e.g., suburban or rural patterns) with minor retraining. This makes the approach scalable and reusable, reducing development and deployment costs.

Real-Time Decision Making

LSTM models, once trained, can perform fast inference, making them suitable for real-time decision-making scenarios in 5G networks. Their ability to forecast resource requirements a few seconds into the future means network orchestration systems can preemptively allocate resources, improving QoS and reducing latency.

Integration of LSTM in the 5G Framework

In this project, the LSTM serves as the core engine of the intelligent resource allocation module. It takes in sequences of historical data, such as:

- Past signal strength measurements
- Application type and priority
- Previous and current latency statistics
- Trends in bandwidth usage and user mobility

Using these inputs, the LSTM model predicts the required resources for the next time interval—specifically, how much bandwidth to allocate to each application or user. These predictions are fed into the resource orchestrator of the 5G architecture, which then executes allocation policies in near real-time.

IV. Technical Depth and Implementation Perspective

This project exemplifies a complete machine learning pipeline integrated into a 5G network framework. The following components outline its technical depth:

Data Engineering

Raw data from the 5G environment undergoes a rigorous preprocessing stage. This includes:

- Cleaning timestamps to ensure uniform time intervals
- Encoding categorical variables such as application type (e.g., streaming, gaming)
- Normalizing numeric inputs like bandwidth (converting Kbps to Mbps)
- Handling missing values and outliers to enhance model robustness

This preprocessing ensures that the input fed into the LSTM model is standardized and meaningful.

Feature Engineering

Careful feature selection enhances model learning. The chosen features—such as real-time latency, signal strength, bandwidth usage, and application requirements—represent a comprehensive view of the network state. These features are engineered to reflect both short-term fluctuations and long-term trends in user behavior and network load.

Model Development

The LSTM model is implemented using PyTorch, a popular deep learning library. It consists of layers optimized for sequence modeling:

- An input layer that accepts sequences of vectors representing network metrics over time
- One or more LSTM layers that learn temporal dependencies
- Fully connected layers that convert the final hidden state into a prediction (e.g., bandwidth percentage)

Training involves minimizing a regression loss function such as Mean Squared Error (MSE), optimized using algorithms like Adam. The model is trained iteratively on sequences of past data to learn patterns and make future predictions.

Evaluation Metrics

Although specific evaluation metrics are not detailed in the notebook, typical metrics include:

- Mean Absolute Error (MAE): Measures average prediction error
- Root Mean Square Error (RMSE): Penalizes larger errors
- R^2 Score: Indicates how well the model explains variance in the data

These metrics are essential to validate the model's performance before deployment.

Scalability and Future Directions

This LSTM-based solution can be extended in several ways:

- **Real-Time Streaming Integration**: Incorporating live data streams using Kafka or MQTT for continuous model inference
- **Edge Deployment**: Running lightweight versions of the model on edge devices for ultra-low-latency predictions
- **Advanced Architectures**: Enhancing accuracy using Transformer-based models that can capture even more complex dependencies

By integrating LSTM into a 5G framework, this project demonstrates the power of sequence learning for intelligent, data-driven network optimization. It lays a foundation for further exploration of AI techniques in telecommunications and contributes to the development of adaptive and resilient 5G systems.

Dataset Overview: 5G Quality of Service

The "5G Quality of Service" dataset, provided by Omar Sobhy on Kaggle, is a comprehensive collection of data focused on analyzing and evaluating the quality of service (QoS) in 5G networks. It serves as a valuable resource for researchers, data scientists, telecom engineers, and students interested in studying modern wireless communication systems, particularly the performance and behavior of 5G networks.

Objective and Use Cases

The primary goal of the dataset is to facilitate the study of various QoS parameters in a simulated 5G environment. By analyzing this data, users can gain insights into network performance issues, predict service degradation,

optimize network infrastructure, and develop machine learning models to enhance user experience.

Potential use cases include:

- **Predictive maintenance and optimization** of network resources.
- **Anomaly detection** in network traffic and performance.
- **QoS classification models** for automation in network management.
- **Reinforcement learning or AI-based network tuning.**

Parameter-wise Explanation of the 5G Quality of Service Dataset

1. Delay

- **Definition**: Time taken for a data packet to travel from the source to the destination, measured in **milliseconds (ms)**.
- **Significance**:
    - A critical metric in real-time services (e.g., video calls, gaming).
    - Lower delay = better user experience.
    - Higher delay may cause lag or buffering issues.

2. Bandwidth

- **Definition**: The maximum rate of data transfer across a given path, typically measured in **megabits per second (Mbps)**.
- **Significance**:
    - Determines how much data can be transmitted in a given time.
    - High bandwidth = faster downloads and streaming.
    - Low bandwidth leads to slower performance and possible congestion.

3. Power

- **Definition**: Signal strength received by a user device, possibly in **decibels-milliwatts (dBm)** or a normalized unit.
- **Significance**:

- Affects the ability of the device to maintain a stable connection.
- Higher power means better signal strength and fewer dropped packets.
- Low power can lead to weak or unstable connections.

4. User Count

- **Definition**: The number of users connected to a specific 5G cell or base station at a given time.
- **Significance**:
  - Indicates network load and congestion.
  - High user count can cause resource sharing, leading to reduced QoS.
  - Helps in understanding the scalability and performance under load.

5. Quality of Service (QoS)

- **Definition**: A **categorical label** representing the overall quality of service experienced by a user or session.
- **Possible Values** (based on common QoS classifications):
  - **Excellent**: Low delay, high bandwidth, strong signal, low user load.
  - **Good**: Acceptable performance with minor fluctuations.
  - **Fair**: Slightly degraded performance; may affect some applications.
  - **Poor**: High delay, low bandwidth, weak signal, or network congestion.
- **Significance**:
  - This is the **target variable** for many ML tasks.
  - Helps in developing models that predict user experience.
  - Can be used to trigger automated optimization in real-time systems.

Dataset Features

- The dataset contains approximately 100,000 rows and includes the following key features (columns), each of which plays a significant role in assessing the quality of 5G service:

Feature	Description
Delay	The time delay in milliseconds (ms) experienced in data transmission. Important for latency-sensitive applications.
Bandwidth	The available bandwidth in megabits per second (Mbps). Affects how much data can be transferred.
Power	Signal power, possibly in dBm or a normalized unit. Impacts signal strength and coverage.
User Count	Number of users connected in the cell/sector. Helps understand congestion levels.
Quality of Service (QoS)	The target variable; a label indicating the overall quality of service experienced. It may be categorical (e.g., "Excellent", "Good", "Fair", "Poor").

Applications in AI/ML

This dataset is ideal for a range of artificial intelligence and machine learning experiments. Example tasks include:

- **Supervised Classification**: Predicting QoS levels based on features like delay, power, and bandwidth using decision trees, SVMs, random forests, or neural networks.
- **Clustering and Anomaly Detection**: Unsupervised techniques can detect unusual network behavior or performance bottlenecks.
- **Reinforcement Learning**: Adjusting power or bandwidth allocation to improve real-time QoS.
- **Optimization Models**: Using the dataset to simulate different user densities and determine optimal configurations.

Detailed Explanation of Implementation Steps

1. Data Preprocessing

Before we can train any machine learning model, especially one as sensitive as LSTM which works with sequences, the raw data must be thoroughly cleaned, transformed, and standardized. Here's how we did that:

1.1 Importing and Organizing the Dataset

We started by reading the dataset from a CSV file that contained Quality of Service (QoS) data for a 5G network. This included metrics like signal strength, latency, application type, and bandwidth information.

To ensure proper chronological order—critical for time-series models—we converted the Timestamp column into datetime format and sorted the dataset based on it. This step maintains the correct sequence of events for training the LSTM model, which relies on learning from past data to predict future outcomes.

1.2 Encoding Categorical Data

The dataset contained a column called Application_Type, which was categorical (e.g., Gaming, Streaming, etc.). Since machine learning models cannot work directly with textual categories, we applied **Label Encoding**. This step replaced each unique application type with a unique integer. For instance, "Gaming" might become 0, "Streaming" might become 1, and so on.

1.3 Cleaning Numeric Data

Several fields in the dataset had string representations with units:

- **Signal Strength** had values like -55 dBm
- **Latency** had values like 34 ms
- **Bandwidth** fields included units like Kbps or Mbps

We used string operations to remove the unit symbols (dBm, ms, Kbps, Mbps) and then converted the strings to floating-point numbers. For bandwidth, a custom function converted all units to a common scale (e.g., Mbps) for consistency.

1.4 Target Variable Preparation

The column Resource_Allocation, which we aimed to predict, was originally in percentage format (e.g., 75%). We cleaned the data by removing the percent sign and dividing the number by 100 to normalize it to a 0–1 scale (i.e., 75% became 0.75).

1.5 Feature Scaling

Machine learning models often perform better when features are on the same scale. Hence, we used **Min-Max Scaling** to normalize all the relevant features (like signal strength, latency, etc.) between 0 and 1. This helps LSTM learn more efficiently by avoiding domination of larger numerical values over smaller ones.

2. Sequence Generation for Time Series Prediction

LSTM models learn from sequences of data, not from single data points. To prepare our dataset for this, we used a sliding window approach.

We created sequences where:

- Each input to the model was a series of **10 consecutive time steps** (e.g., 10 rows of features).
- The target value was the resource allocation value **right after** that 10-step window.

This allows the LSTM to learn how past trends affect future outcomes.

3. Splitting the Data

To evaluate the model properly, we split the data as follows:

- **80% of the data** was reserved for training and validation.
- **20% of the data** was used as a test set for final evaluation.

Then, from the 80% training data:

- **80% was used for training**
- **20% for validation**

This multi-level split ensured the model had a good variety of data to learn from, validate on, and finally test its generalization.

4. Model Architecture

We designed an LSTM-based deep learning model in PyTorch. Here's what it consisted of:

Model Components:

- **Input Layer:** Takes in the feature vectors over 10 time steps.
- **LSTM Layers:** Two LSTM layers with 64 hidden units each to capture the temporal dependencies in the data.
- **Fully Connected Layer:** Maps the final hidden state output from LSTM to a single value (the predicted resource allocation).
- **Activation Function:** A sigmoid-like output to stay within the normalized (0–1) range.

5. Model Training

Training Setup:

- **Loss Function:** We used **Mean Squared Error (MSE)**, which measures how far off our predictions are from actual values.
- **Optimizer: Adam**, a popular gradient-based optimizer that adapts the learning rate for each parameter.
- **Batch Size:** 32 sequences per batch for training.
- **Epochs:** We used 100 epochs with early stopping patience=10.

We ran the training loop for up to 100 epochs, but with a clever twist…

6. Early Stopping

To avoid overfitting and wasting time on unnecessary training, we implemented **early stopping**:

- If the model did not improve on the **validation set** for 10 consecutive epochs, training was halted early.
- This ensured the model trained just enough to generalize well, without memorizing the training data.

7. Model Evaluation

After training, the model was tested using the **20% holdout test data**. We used several metrics to assess performance:

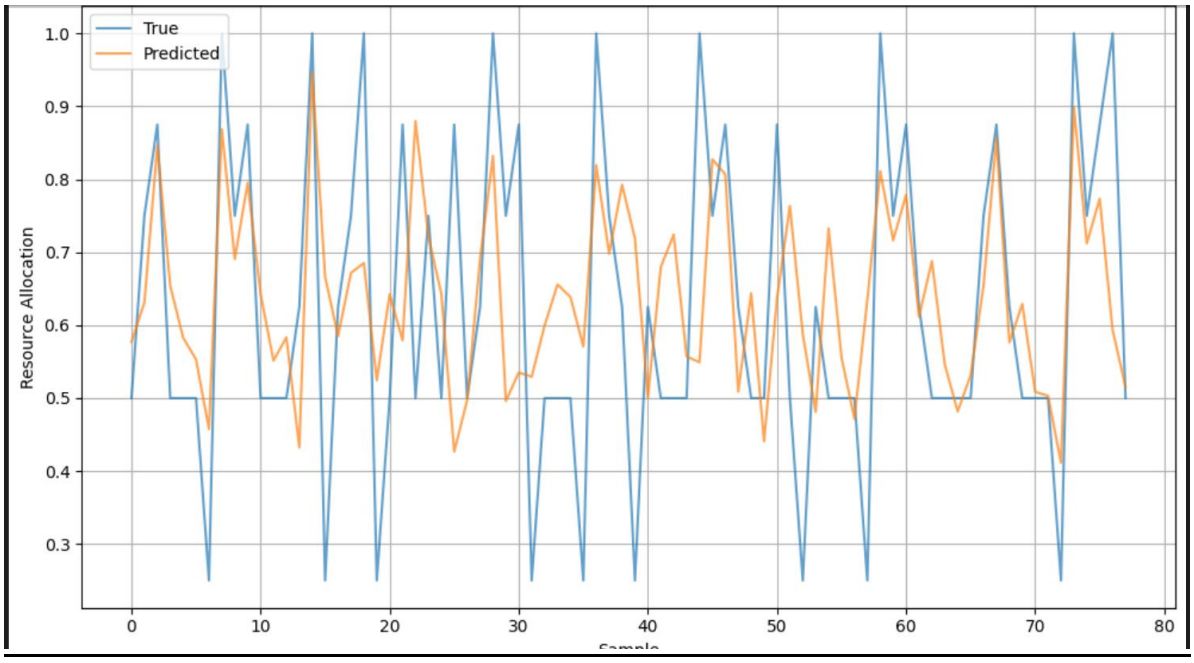
Evaluation Metrics:

- **Mean Absolute Error (MAE):** Average magnitude of errors.
- **Mean Squared Error (MSE):** Penalizes larger errors more.
- **R^2 Score:** A statistical measure of how close the predictions are to actual values (closer to 1 is better).

We also **plotted** the true vs predicted values of Resource_Allocation to visually assess how closely the model's predictions tracked the real data.

OUTPUT

```
Test MSE: 0.0372
Test MAE: 0.1500
Test R² Score: 0.2357
```



Steps to Integrate AWS S3 and Load the Model:

1. Setup AWS S3 and Boto3

- AWS S3 Setup: First, you need an AWS account and access to AWS S3 where the trained model (best_model.pth) is stored. If you don't already have a bucket, you would need to create one in the AWS S3 Console.
- Boto3 Library: Boto3 is the AWS SDK for Python, and it allows your Python code to interact with AWS services, including S3. You need to install boto3 in your environment (use pip install boto3).

2. Configure AWS Credentials

- **Credentials Setup**: For Boto3 to authenticate and access your AWS resources, you need to configure your AWS credentials (access key and secret key). This can be done in the ~/.aws/credentials file, or you can set the environment variables AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY.
- **IAM Permissions**: Ensure that the IAM user or role you are using has the necessary permissions to access the S3 bucket (e.g., s3:GetObject).

3. Import Necessary Libraries

In your Python script, import the required libraries:

- **PyTorch (torch)**: For defining the LSTM model and handling the model's forward pass.
- **Boto3 (boto3)**: For accessing AWS S3 and downloading the model.

4. Define the LSTM Model Architecture

- Define the architecture of the LSTM model that will be used for inference. The model should match the structure used when the model was trained.

5. Set Up S3 Bucket and Download the Model

- **Boto3 Client**: Use the Boto3 client to connect to your S3 bucket, specifying the region where your bucket is located (region_name='eu-north-1' in your case).
- **Download the Model**: You specify the S3 bucket (my-lstm-models), the key (path to the model best_model.pth), and the local file path where the model will be saved (local_path).

- **Exception Handling**: Add error handling to catch issues during the download process (e.g., incorrect bucket name, missing file).

6. Load the Model Weights into the LSTM Architecture

After downloading the model, load the weights into the model using PyTorch's torch.load() and load_state_dict() methods. This step ensures the model is initialized with the trained weights from the .pth file.

- **Loading the Model Weights**: Use the torch.load() function to load the downloaded .pth file into the model's state dictionary.
- **Strict=False**: If the model's architecture does not match exactly with the loaded state (i.e., extra or missing keys), strict=False will allow the model to load successfully by ignoring those mismatches.

7. Set the Model to Evaluation Mode

- Once the model is loaded, set it to **evaluation mode** by calling `model.eval()`. This is necessary because some layers in PyTorch, such as dropout and batch normalization, behave differently during training and inference.

8. Run Inference on the Model

- Once the model is loaded and set to evaluation mode, you can use it for inference. For demonstration purposes, generate a random input tensor (`sample_input`) and pass it through the model.

9. Main Execution Flow

The main block of code orchestrates the process:

- It calls load_model_from_s3() to download and load the model.
- Then, if the model is successfully loaded, it runs inference using run_inference().

Summary of Integration Steps

Here's a quick summary of how AWS S3 was integrated into the project:

1. **Install and Configure Boto3**: Set up the boto3 library to interact with AWS S3 and configure AWS credentials for authentication.
2. **Download Model from S3**: The trained model was downloaded from the S3 bucket using the s3.download_file() method, specifying the bucket name, file key, and local path.

3. **Load Model Weights**: The downloaded model file (best_model.pth) was loaded into a PyTorch model using torch.load() and model.load_state_dict().
4. **Run Inference**: After loading the model, it was set to evaluation mode, and inference was performed using a random input tensor.