Digital Twin of 5G Network Documentation

version

2025, David Truhlar

May 03, 2025

Contents

Welcome to the Digital Twin of 5G Network!

Getting Started

Welcome to the documentation for the **Digital Twin of a 5G Network** developed as part of a bachelor thesis at the Slovak University of Technology. This section will guide you through setting up, running, and extending the digital twin environment.

Overview

This project simulates and analyzes 5G network behavior by integrating the following components:

- **Open5GS** – a full 5G core implementation.
- **UERANSIM** – emulation of multiple UE and gNBs.
- **Prometheus & Grafana** – monitoring and visualization.
- **LSTM models** – classification of network behavior using deep learning.
- **Sphinx documentation** – modular API docs.

Prerequisites

Before running the project, ensure you have:

- Docker & Docker Compose
- Python 3.9+ and *venv*
- Git
- At least 16 GB RAM (recommended for full stack simulation)

Installation Guide

First, ensure you have the following prerequisites installed:

```
Docker and Docker Compose
Python 3.9+ and `venv`
Git
```

Second, clone the repository:

```
git clone https://github.com/xtruhlar/5GDigitalTwin.git
cd 5GDigitalTwin/Implementation
```

To build docker images

```
cd ./base
docker build -t docker_open5gs .

cd ../ueransim
docker build -t docker_ueransim .

cd ..
```

To set the environment variables

```
cp .env.example .env

set -a
source .env
set +a
```

To run the project, navigate to *Implementation/* and execute the following command

```
docker compose -f deploy-all.yaml up --build -d
```

To add subscribers to Open5GS core, run the following commands

```
docker exec -it mongo mkdir -p /data/backup
docker cp ./mongodb_backup/open5gs mongo:/data/backup/open5gs
docker exec -it mongo mongorestore --uri="mongodb://localhost:27017" --db open5gs /data/back
```

To ensure everything works properly, open http://localhost:9999/ in your browser and login using credentials:

```
Username: admin
Password: 1423
```

To connect UERANSIM gNB to Open5GS, run the following command

```
docker compose -f nr-gnb.yaml -p gnodeb up -d && docker container attach nr_gnb
```

To connect UERANSIM UE to Open5GS, run the following command

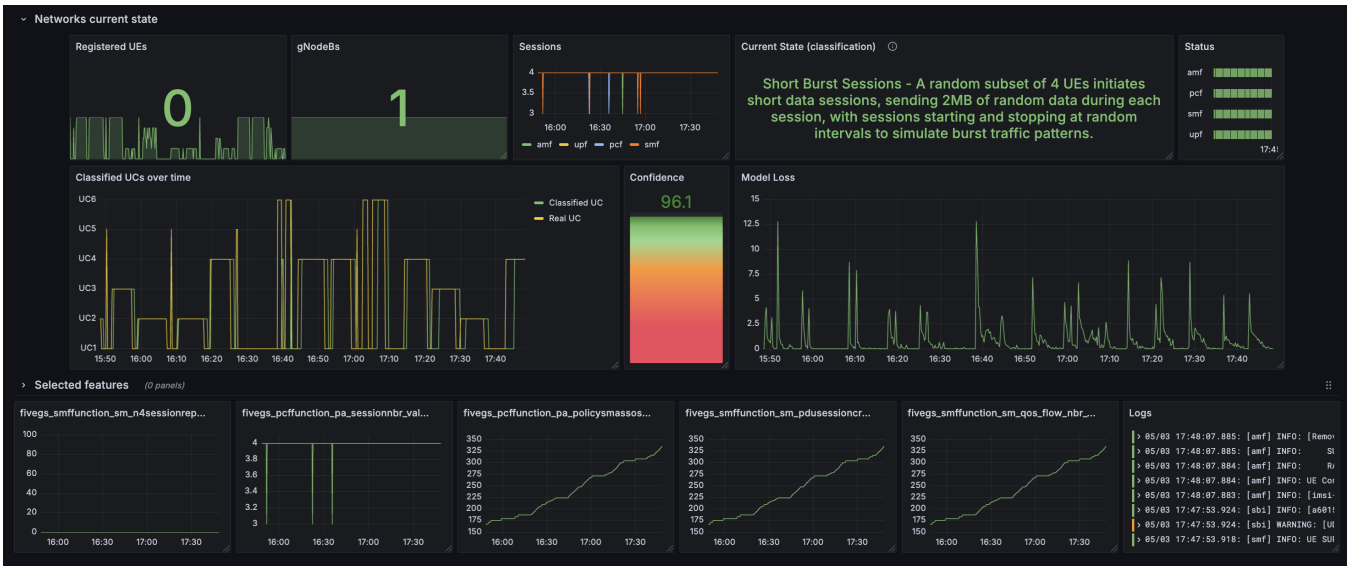
```
docker compose -f nr-ue.yaml -p ue up -d && docker container attach ue
```

Then go to Grafana, open http://localhost:3000/ in your browser and login using credentials:

```
Username: open5gs
Password: open5gs
```

Open menu on the left, click on *Dashboards*. Select *Current state Dash* and you can see the current state of your 5G network.

Example:



Grafana dashboard

Normal Surfing

`uc1.`**`run_uc1`** ()

Run UC1: Normal Surfing scenario.
Simulates user behavior with intermittent UE connectivity and randomized data downloads to mimic typical mobile web browsing patterns.

Scenario Summary

- Starts half of the UEs.
- **UEs may**

- download random chunks of data (5–50MB),
- randomly disconnect or reconnect during the session.

- The scenario runs for a randomized session duration (60–600 seconds).
- Active UEs are defined in *nr-ue{i}.yaml* Docker Compose files.
- Writes the current UC label to *data/current_uc.txt*.

Args

None

Returns

None

Video Streaming

`uc2.run_uc2` ()

Run UC2: Video Streaming scenario.
Simulates a typical video streaming session where all UEs continuously receive data. This scenario helps to evaluate throughput and session stability under constant load.

Scenario Summary

- Starts 4 UEs using Docker Compose.
- Each UE downloads 2MB of random data every second.
- Streaming duration is randomized between 300 and 600 seconds.
- The UC label is logged into 'data/current_uc.txt'.

Args

None

Returns

None

Periodic Keep-Alive

`uc3.run_uc3` ()

Run UC3: Periodic Keep-Alive scenario.
Simulates multiple UEs that periodically send HTTP requests (keep-alive pings) to a remote server. This pattern reflects real-world background traffic in mobile applications (e.g. chat apps, weather updates).

Scenario Summary

- Starts 4 UEs as containers using Docker Compose.
- Each UE sends periodic HTTP GET requests (via *curl*) to a predefined URL.
- The interval between pings is randomized between 30–35 seconds.
- Simulation duration is randomly set between 300–600 seconds.
- Scenario type is logged into 'data/current_uc.txt'.

Args

None

Returns

None

Short Burst Sessions

`uc4.run_uc4` ()

Run UC4: Short Burst Sessions scenario.
Simulates multiple UEs initiating brief data sessions at random intervals. This pattern mimics sporadic, high-frequency user actions (e.g. short API requests, fast-loading web content) in mobile networks.

Scenario Summary

- Randomly selects a UE out of 4 available.
- Starts a short Docker container session for the selected UE.
- Simulates a 2MB data transfer using *dd* inside the container.
- After 2–4 seconds, the UE container is stopped.
- Waits 3–6 seconds and repeats until the scenario duration ends.
- Marks current use case in ‘data/current_uc.txt’.

Args

None

Returns

None

Load Registration Anomaly

uc5.**run_uc5** ()
Run UC5: Load Registration Anomaly scenario.
Simulates a stress event in the 5G network by concurrently connecting multiple UEs. This tests the network’s ability to handle sudden, simultaneous registration attempts — a common anomaly in overloaded environments.

Scenario Summary

- Starts 4 UE containers simultaneously using *subprocess.Popen*.
- Waits for all containers to fully initialize.
- Holds all UE sessions active for a short time (5 seconds by default).
- Stops all containers simultaneously after the wait period.
- Logs scenario activity to ‘data/current_uc.txt’.

Args

None

Returns

None

Authentication Failure Alert

uc6.**run_uc6** ()
Run UC6: Authentication Failure Alert scenario.
Simulates an authentication failure event in a 5G network by repeatedly starting a misconfigured UE (User Equipment) that fails to register due to incorrect credentials or malformed configuration. This scenario is useful for testing network response to repeated failed attempts and monitoring for anomaly detection mechanisms.

Scenario Summary

- Launches a specially prepared UE (ID=100) which is expected to fail authentication.
- Repeats the process a random number of times (3 to 6 retries).
- Between each attempt, the UE is stopped and a random interval (5–30 seconds) is observed.
- Total scenario duration is also bounded by a global timeout (120–300 seconds).
- All events are logged to the console and scenario type is saved to *data/current_uc.txt*.

Args

None

Returns

None

LSTM Model with Attention

LSTM model with custom attention mechanism for multi-class classification of time-series data in the context of 5G network behavior prediction.

This module defines a deep learning model using TensorFlow and Keras, integrates a custom attention mechanism, and trains the model on preprocessed input data with categorical labels.

Expected data format: - Input: X_train.npy, X_test.npy (shape: [samples, 60, features]) - Labels: y_train.npy, y_test.npy (categorical class indices) - Class weights: class_weights.json

The trained model is saved as HDF5 and Keras formats.

`lstm_attention_model.`**`train_attention_model`**()

LSTM Base Model

LSTM Base Model for classification of network use case scenarios.

This script loads preprocessed training and testing data, defines and trains a baseline LSTM model, evaluates its performance, and saves the final model in HDF5 and Keras formats. Class balancing is handled using precomputed class weights.

Usage:

This script is designed to be executed as a module. Use the function *build_base_model()* to construct and optionally train the model.

`lstm_base_model.`**`build_base_model`**(`X_train`, `y_train`, `X_test`, `y_test`, `class_weight_dict`)

Builds, trains, evaluates and saves a baseline LSTM model.

LSTM BathNorm Model

lstm_robust_model.py

Trénovanie hlbokého LSTM modelu s viacerými vrstvami a dropoutmi pre klasifikáciu typov používate■ského správania v 5G sieti.

Modul obsahuje: - *build_robust_model()*: hlavná funkcia na tréning modelu, vizualizáciu a uloženie

`lstm_bathnorm_model.`**`build_robust_model`**()

Vytrénuje robustný LSTM model a uloží ho do prie■inka *trained_models/*.

Tento model obsahuje 3 vrstvy LSTM s rôznymi dropoutmi a 2 Dense vrstvy. Používa sa na klasifikáciu UC tried na základe predspracovaných sekven■ných vstupov.

Classify the real data using LSTM

evaluate_and_finetune_models.py

Modul na vyhodnotenie viacerých LSTM modelov (base, robust, batchnorm, attention) na reálnych dátach a možnos■ dodato■ného finetuningu attention modelu.

Obsahuje: - definíciu attention vrstvy, - výpo■et váh tried z reálnych dát, - generovanie sekven■ných dát, - vyhodnotenie klasifika■nej presnosti, - možnos■ jemného dotrénovania attention modelu na reálnych dátach.

class `lstm_results_real_data.`**`AttentionLayer`**(`*args`, `**kwargs`)

Bases: **Layer**
Custom Attention Layer pre LSTM architektúru.

build(input_shape)

call(x)

compute_output_shape(input_shape)

lstm_results_real_data.**create_sequences**(X, y, seq_len=60)
Transformácia plochých vstupov na sekven■né okná pre LSTM modely.

lstm_results_real_data.**evaluate_model**(model, X_seq, y_seq, name)
Vyhodnotenie modelu pomocou klasifika■nej správy.

lstm_results_real_data.**load_and_preprocess_data**()
Na■íta a spracuje reálne dáta + zmapuje kategórie.

lstm_results_real_data.**run_evaluation_and_finetuning**()
Hlavná funkcia – vyhodnotenie 4 modelov + volite■ný finetuning attention modelu.

Exploratory Data Analysis

EDA module for exploratory analysis of synthetic and real 5G network datasets.

This module contains functions to load data, preprocess it, visualize it, and perform feature selection using multiple strategies including RF, RFE, RFECV, SFS and permutation importance.

Functions in this module should be called explicitly from a main script or notebook.

eda.**compute_class_weights**(y)
Compute class weights for imbalanced classes.

eda.**load_dataset**(path)
Load dataset from CSV file.

Return type: **DataFrame**

eda.**load_maps**(log_map_path='log_map.json', app_map_path='app_map.json', uc_map_path='uc_map.json')
Load mapping dictionaries from JSON files.

eda.**permutation_importance_stable**(X, y, selected_features, n_runs=10)
Calculate stable permutation importances over multiple runs.

eda.**preprocess_data**(df, log_map, app_map, uc_map)
Preprocess dataset: fill NA, map strings to ints, scale numeric columns.

eda.**random_forest_importance**(X_scaled, X, y)
Train Random Forest and return feature importances.

eda.**rfe_selection**(X_scaled, y, X, rf)
Recursive Feature Elimination.

eda.**rfecv_selection**(X_scaled, y, X, rf)
RFECV - RFE with cross-validation.

eda.**sfs_selection**(X_scaled, y, X, rf)
Sequential Feature Selector.

Label Real Dataset

Module for labeling real network CSV data with Use Case (UC) intervals.

This module loads the original dataset, compares each timestamp to predefined intervals, assigns a UC label, and saves the labeled dataset to disk.

add_current_uc.**apply_uc**(row)

Assign a Use Case (UC) label to a row based on its timestamp.

Parameters: **row** (*pd.Series*) – A row with a ‘timestamp’ field.

Returns: The UC label (‘uc1’ to ‘uc6’).

Return type: str

`add_current_uc.`**`compare_intervals`**`(row, interval)`

Compare a timestamp from a row with a given time interval.

Parameters:

- **row** (*pd.Series*) – Row from the DataFrame with a ‘timestamp’ column.
- **interval** (*dict*) – Dictionary with “from” and “to” datetime strings.

Returns: True if timestamp is within interval, else False.

Return type: bool

`add_current_uc.`**`label_realnetwork_csv`**`(input_path='../backup/realnetwork.csv', output_path='../real_data.csv')`

Load the CSV, assign UC labels to each row, and save the updated file.

Parameters:

- **input_path** (*str*) – Path to the input CSV file.
- **output_path** (*str*) – Path to save the labeled CSV file.

Preprocess data for LSTM

Module for preparing LSTM input data from preprocessed features. Includes functionality for loading feature arrays, creating sequences, splitting the dataset, and saving the output for training and testing.

This module is intended for use with the Digital Twin of 5G Network project.

`lstm_preprocessing.`**`create_sequences`**`(X, y, seq_len)`

Vytvorí sekvencie vstupných dát pre LSTM z k■zavého okna.

Parameters:

- **X** (*np.ndarray*) – Vstupné dáta (features)
- **y** (*np.ndarray*) – Cie■ové hodnoty (triedy)
- **seq_len** (*int*) – D■žka sekvencie pre LSTM

Returns: (X_seq, y_seq) ako ndarray

Return type: tuple

`lstm_preprocessing.`**`load_data`**`(X_path, Y_path, scaler_path, features_path, uc_map_path)`

Na■íta vstupné dáta, škálova■, vybrané príznaky a mapu UC tried.

Parameters:

- **X_path** (*str*) – Cesta k súboru s X vstupmi (.npy)
- **Y_path** (*str*) – Cesta k súboru s y triedami (.npy)
- **scaler_path** (*str*) – Cesta k uloženému škálova■u
- **features_path** (*str*) – Cesta k JSON súboru s vybranými príznakmi
- **uc_map_path** (*str*) – Cesta k JSON súboru s mapovaním UC

Returns: (X, y, scaler, selected_features, uc_map)

Return type: tuple

`lstm_preprocessing.`**`split_and_save_data`**`(X_seq, y_seq, output_dir='preprocessed_data')`

Rozdelí dáta na trénovaciu a testovaciu množinu a uloží ich.

Parameters:

- **X_seq** (*np.ndarray*) – Vstupné sekvencie
- **y_seq** (*np.ndarray*) – Výstupné triedy
- **output_dir** (*str*) – Adresár pre uloženie súborov

Simulate a running network

running_network.**init_log** ()
Initializes the log file and directory if needed.

Args

None

Returns

None

running_network.**run_simulation** (script_name)
Runs the selected UC simulation and logs its outcome.

Args

- script_name (str): Name of the UC script to run.

Returns

None

Main orchestrator

class log_watcher.**AttentionLayer** (*args, **kwargs)
Bases: **Layer**
Custom attention layer for LSTM model. This layer computes the attention weights and applies them to the input sequence.

Args

- Layer (tf.keras.layers.Layer): Base class for all layers in Keras.

Returns

None

build (input_shape)
Create the attention weights and bias.

Args

- input_shape (tuple): Shape of the input tensor.

Returns

None

call (x)
Calculate the attention weights and apply them to the input sequence.

Args

- x (tensor): Input tensor of shape (batch_size, sequence_length, features).

Returns

- tensor: Output tensor of shape (batch_size, features).

compute_output_shape (input_shape)
Compute the output shape of the layer.

Args

- input_shape (tuple): Shape of the input tensor.

Returns

- tuple: Shape of the output tensor.

log_watcher.**clean_old_models** (directory='/app/data/Model', keep_last_n=7, pattern='Model_bn_*.keras')
Keep only the last *keep_last_n* saved model files and delete older ones.

Args

- directory (str): Directory containing the model files.
- keep_last_n (int): Number of recent models to keep.
- pattern (str): Pattern to match model files.

Returns

None

log_watcher.**load_last_sequence** (csv_path, selected_features, sequence_length=60)
Load the last sequence of records from CSV, ensuring all required features and labels are present.

Args

- csv_path (str): Path to the CSV file.
- selected_features (list): List of features to select from the DataFrame.
- sequence_length (int): Length of the sequence to load.

Returns

- tuple: DataFrame with selected features and the correct labels.

Raises

- ValueError: If any of the selected features are missing in the DataFrame.

log_watcher.**main_loop** (interval=1, prometheus_port=9000)
Main loop that monitors UE activity, parses logs, updates Prometheus metrics, and fine-tunes the model in real-time.

Args

- interval (int): Time interval for monitoring and updating metrics.
- prometheus_port (int): Port for Prometheus metrics.

Returns

None

log_watcher.**parse_amf** (lines, previous_state)
Parse AMF log lines to extract UE registration and deregistration events, update UE states, and compute registration/session durations.

Args

- lines (list): List of log lines to parse.
- previous_state (dict): Previous state of UE details and durations.

Returns

- tuple: Updated UE details, new registration durations, and new session durations.

log_watcher.**predict_current_uc** (latest_window_df)
Predict the current use case (UC) using the loaded LSTM model based on the latest data window.

Args

- latest_window_df (pd.DataFrame): DataFrame containing the latest data window.

Returns

- tuple: Predicted UC class and confidence score.

log_watcher.**remove_offset** ()
Remove the offset file used by Pygtail to start reading the log file from the beginning.

Args

None

Returns

None

log_watcher.**run_main_notebook_with_backup** ()
Execute the main.ipynb notebook, log its execution, truncate CSV, and periodically create CSV backups.

Args

None

Returns

None

log_watcher.**run_notebook_in_thread** ()
Run the main.ipynb notebook in a separate thread to avoid blocking the main loop.

Args

None

Returns

None

log_watcher.**save_model_with_date** (model, path_prefix='/app/data/Model/Model_bn_')
Save the current model to disk with the current date as part of the filename.

Args

- model (tf.keras.Model): The model to save.
- path_prefix (str): Prefix for the filename.

Returns

None

log_watcher.**truncate_running_data** (csv_path, keep_last_n=60)
Truncate the CSV file to keep only the last *keep_last_n* records.

Args

- csv_path (str): Path to the CSV file.
- keep_last_n (int): Number of records to keep.

Returns

None

Index

S

T

U

Python Module Index

a

add_current_uc

e

eda

l

log_watcher

lstm_attention_model

lstm_base_model

lstm_bathnorm_model

lstm_preprocessing

lstm_results_real_data

r

running_network

u

uc1

uc2

uc3

uc4

uc5

uc6