Project Διάχυτου Υπολογισμού

Εφαρμογή Federated Learning συστήματος σε τοπικά φανάρια εξομοιωτή κυκλοφορίας οχημάτων

Νικόλαος Ανδριανόπουλος
Μηχ. Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
Πάτρα, Ελλάδα
up1084637@ac.webmail.gr

Δημήτριος Στασινός
Μηχ. Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
Πάτρα, Ελλάδα
up1084643@ac.webmail.gr

Βασίλειος Μάριος Κουρτάκης
Μηχ. Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών
Πάτρα, Ελλάδα
up1090061@ac.webmail.gr

ABSTRACT

Το πρόβλημα της αυξημένης κίνησης στα μεγάλα αστικά κέντρα είναι ένα σύγχρονο πρόβλημα. Οι παραδοσιακοί μέθοδοι διαχείρισης της κυκλοφορίας είναι ανεπαρκείς και αναζητούνται νέοι μέθοδοι διαχείρισής της. Η τεχνητή νοημοσύνη είναι ένα εργαλείο που θα μπορούσε να χρησιμοποιηθεί και να δώσει λύση σε αυτό το πρόβλημα, κάτι που χρησιμοποιούμε και στην δικιά μας πρόταση. Προτείνουμε την χρήση της τεχνητής νοημοσύνης για την βελτιστοποίηση των χρονισμών των φαναριών ώστε να αυξήσουμε την μέση ταχύτητα της διερχόμενης κίνησης.

KEYWORDS

Federated Learning, Deep Learning, Traffic Control, Traffic Simulator, Kafka server, DeepNN, GRU, Pytorch, UXSIM

Επιλογή Εφαρμογής

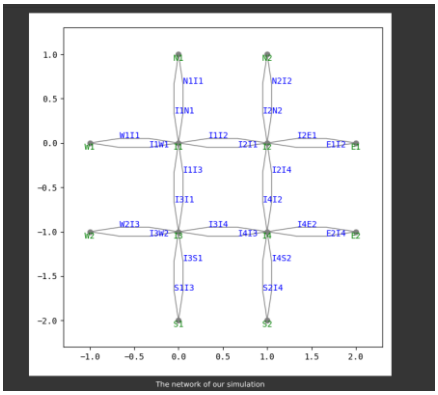
Για να καταφέρουμε να υλοποιήσουμε την εφαρμογή η οποία μας ζητείται (η 4η επιλογή του Project), χρειαζόμαστε τον εξομοιωτή που μας παρέχεται (UXSIM), τη προγραμματιστική γλώσσα Python, τη βιβλιοθήκη PyTorch για να μπορέσουμε να εφαρμόσουμε Deep Neural Network αλγόριθμους καθώς και άλλες βιβλιοθήκες της python.

1. Περιγραφή και στόχοι προβλήματος

Το πρόβλημα που παρουσιάζεται είναι οι σταθεροί χρονισμοί που παρέχονται σε κάθε διασταύρωση, οι οποίοι μπορεί να μην είναι αποτελεσματικοί για την αποτελεσματική διαχείριση της κυκλοφορίας.

Στόχοι:

1. Τοπική εκπαίδευση σε κάθε φανάρι και συλλογή δεδομένων με χρήση federated learning.
2. Δημιουργία συστήματος επικοινωνίας των τοπικών φαναριών με ένα απομακρυσμένο server.
3. Κατανόηση του εξομοιωτή.
4. Σωστή χρήση των δεδομένων που παρέχονται από το νευρωνικό δίκτυο και σωστή εκπαίδευση του.
5. Δημιουργία ενός ολικού μοντέλου, αξιολόγηση και χρήση του

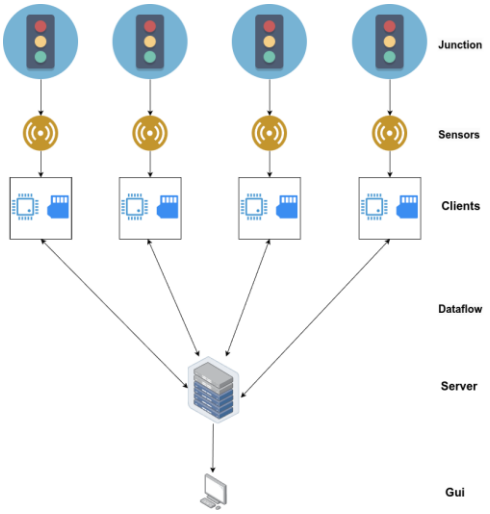


Σχήμα 1.1: Το δίκτυο στο οποίο υλοποιούμε το federated learning

2. Τελικός σχεδιασμός εφαρμογής

Για να βελτιώσουμε τους χρονισμούς στα τοπικά φανάρια, χρησιμοποιούμε federated learning με χρήση Kafka και νευρωνικών δικτύων.

2.1. Αρχιτεκτονικά διαγράμματα



Σχήμα 2.1: Αρχιτεκτονικό διάγραμμα της υλοποίησής μας

1. Αρχικά, λαμβάνουμε τα δεδομένα του εξομοιωτή (δεδομένα από κάθε τοπικό φανάρι) για να εκπαιδεύουμε τα νευρωνικά δίκτυα. Αλλάζουμε δυναμικά τα σήματα των φαναριών ανάλογα τις πυκνότητες των δρόμων (μεγαλύτερη πυκνότητα: μεγαλύτερος χρονισμός πράσινου χρώματος και αντίστροφα). Έτσι, το dataset μας προσαρμόζεται ανάλογα με τις πυκνότητες να περιέχει σωστά έναν δυναμικό χρονισμό με στόχο να μειώσει τη κίνηση των δρόμων

2. Έπειτα, εκπαιδεύουμε τα νευρωνικά δίκτυα που προαναφέρθηκαν με την χρήση των παραπάνω dataset. Στόχος της εκπαίδευσης είναι η βελτίωση της μέσης ταχύτητας. Προσπαθούμε να το επιτύχουμε αυτό τοπικά σε κάθε φανάρι προβλέποντας τους χρονισμούς του, με βάση τις πυκνότητες των δρόμων που είναι συνδεδεμένοι στο φανάρι.

Για να μπορεί να δημιουργηθεί το ολικό μοντέλο 4 εισόδων και 2 εξόδων που χρησιμοποιείται σε όλα τα φανάρια με σκοπό τη σωστή πρόβλεψη, πρέπει:

- Στο τοπικό επίπεδο των φαναριών, να εκπαιδεύσουμε τα τοπικά μοντέλα χρησιμοποιώντας το σωστό μέγεθος δεδομένων και επεισοδίων, ώστε το κάθε τοπικό μοντέλο να μπορεί να προβλέψει αποδοτικά τους σωστούς χρονισμούς για το φανάρι του.
- Συνολικά, όλα τα βάρη των μοντέλων να στέλνονται σε ένα server, που με ένα federated average συνυπολογίζει τα βάρη δημιουργώντας μια πρώτη ολική εικόνα και για τα 4 φανάρια.
- Τέλος, η επικοινωνία αυτή πρέπει να εκτελεστεί για ορισμένους γύρους. Η ανατροφοδότηση αυτή έχει ως αποτέλεσμα τη πιο ολοκληρωμένη μάθηση του τρόπου αποσυμφόρησης της συνολικής πυκνότητας κυκλοφορίας.

Πρακτική υλοποίηση με κομμάτια κώδικα:

Αρχικά, για να προσομοιώσουμε τα 4 ξεχωριστά φανάρια, χρησιμοποιούμε 4 threads:

```
threads = [  # Each thread represents the parallel computation in real
time of each traffic light
        threading.Thread(target=start_client_process, args=(
            'I1', 'datasets/data_for_I1.pkl', 0, option)),
        threading.Thread(target=start_client_process, args=(
            'I2', 'datasets/data_for_I2.pkl', 1, option)),
        threading.Thread(target=start_client_process, args=(
            'I3', 'datasets/data_for_I3.pkl', 2, option)),
        threading.Thread(target=start_client_process, args=(
            'I4', 'datasets/data_for_I4.pkl', 3, option))
    ]
```

Τα threads αυτά διαβάζουν δεδομένα από .pkl αρχεία που εκπροσωπούν τα δεδομένα που έχουν συλλεχθεί κατά την απλή λειτουργία τους από τους αισθητήρες.

```
# Load and preprocess the data for each traffic light
        self.data = []
        with open(training_file, 'rb') as f:
            while True:
                try:
                    self.data.append(pickle.load(f))
                except EOFError:
                    break

        self.data_inputs = []
        self.data_targets = []

        for row in self.data:
            densities, timings = row
            self.data_inputs.append(densities)
            self.data_targets.append(timings)
```

Έπειτα, το κάθε φανάρι/thread αρχίζει την εκπαίδευσή του. Τα δεδομένα χωρίζονται 80-20 σε δεδομένα εκπαίδευσης και δεδομένα αξιολόγησης αντίστοιχα.

```
# Train the nn model
    def training(self, epochs):
        train_losses = []
        val_losses = []
        for epoch in range(epochs):
            self.neural.train()
            total_loss = 0
            for densities_batch, timings_batch in
self.train_dataloader:
                densities_batch = densities_batch.to(self.device)
                timings_batch = timings_batch.to(self.device)

                self.optimizer.zero_grad()
                outputs = self.neural(densities_batch)
                loss = self.criterion(outputs, timings_batch)
                loss.backward()
                self.optimizer.step()
                total_loss += loss.item()

            avg_loss = total_loss / len(self.train_dataloader)
            train_losses.append(avg_loss)

            # Validation phase
            self.neural.eval()
            val_loss = 0
            with torch.no_grad():
                for densities_batch, timings_batch in
self.val_dataloader:
                    densities_batch = densities_batch.to(self.device)
                    timings_batch = timings_batch.to(self.device)
                    outputs = self.neural(densities_batch)
                    loss = self.criterion(outputs, timings_batch)
                    val_loss += loss.item()

            avg_val_loss = val_loss / len(self.val_dataloader)
            val_losses.append(avg_val_loss)

        return train_losses, val_losses  # Return the losses for
plotting
```

```
train loss:0.6576960006107887,evaluation loss:0.4547342816988627
train loss:0.28998184961577256,evaluation loss:0.16310700650016466
train loss:0.12171100537913541,evaluation loss:0.08655184805393219
train loss:0.07179254195652902,evaluation loss:0.06230604164302349
train loss:0.051728354472046094,evaluation loss:0.047712421262015 9
train loss:0.03996241127606481,evaluation loss:0.03820171523839235
train loss:0.03174821234618624,evaluation loss:0.032639152463525535
train loss:0.026960308919660746,evaluation loss:0.026793084169427554
train loss:0.022828264355969925,evaluation loss:0.02361381654627621
train loss:0.019931178027763962,evaluation loss:0.022134721387798588
train loss:0.017888278529668847,evaluation loss:0.019243125167364875
train loss:0.016240947224044552,evaluation loss:0.01795788478727142
train loss:0.014882032856500397,evaluation loss:0.01681258054450154
train loss:0.013950983035222938,evaluation loss:0.015587668019967775
train loss:0.012807486922247336,evaluation loss:0.014877579446571568
train loss:0.012117910839151591,evaluation loss:0.01416607719535629
train loss:0.011553454501942421,evaluation loss:0.013605537381954491
train loss:0.010777434012076507,evaluation loss:0.012737690246043105
train loss:0.010247331229038537,evaluation loss:0.012648872612044216
train loss:0.009802841253501053,evaluation loss:0.011578479665331542
```

Screenshot 2.2: Κομμάτι από το training που δείχνει τα training loss, evaluation loss ανά 10 epochs.

Αξίζει να σημειωθεί πως τα δεδομένα κανονικοποιούνται με scalers που δημιουργούνται επάνω σε όλα τα δεδομένα.

```
# For each client, load their data
for client_id in ['I1', 'I2', 'I3', 'I4']:

    client_data = []
    with open(f'datasets/data_for_{client_id}.pkl', 'rb') as f:
        while True:
            try:
                client_data.append(pickle.load(f))
            except EOFError:
                break
     #append all the values in order to make scalers based on all the
traffic lights
    for row in client_data:
        densities, timings = row
        all_data_inputs.append(densities)
        all_data_targets.append(timings)


all_data_inputs = np.array(all_data_inputs)
all_data_targets = np.array(all_data_targets)


densities_scaler = StandardScaler()
timings_scaler = StandardScaler()

densities_scaler.fit(all_data_inputs)
timings_scaler.fit(all_data_targets)


def normalize_and_split_data(self):
        # Use global scalers to transform data
        self.densities_normalized = densities_scaler.transform(
            self.data_inputs)
        self.timings_normalized =
timings_scaler.transform(self.data_targets)

        # Split the data into training and validation sets
        self.train_densities, self.val_densities, self.train_timings,
self.val_timings = train_test_split(
            self.densities_normalized, self.timings_normalized,
test_size=0.2, random_state=42
        )
```

Μόλις τελειώσει η εκπαίδευση, αρχίζει η επικοινωνία με τον server. Για να στείλουμε και να πάρουμε δεδομένα η επικοινωνία του server με το client υλοποιείται με kafka. Για τη λογική της επικοινωνίας χρησιμοποιούνται 3 topics: Το “local-weights”, “global-weights”, και “handshake”.

- **Local weights:** Είναι topic που στέλνονται τα τοπικά βάρη ώστε να τα διαβάσει ο server. Το κάθε thread έχει δικό του partition στο server.

- **Global weights:** Παρόμοια για τα ολικά βάρη.

- **Handshake:** Είναι το topic που ανταλλάζεται η πληροφορία που δείχνει ποιο νευρωνικό δίκτυο θα χρησιμοποιήσει το σύστημα και αποτελεί την αρχή (handshake) της επικοινωνίας client με server.

Σχόλιο: Όταν ο client ή ο server διαβάζει από τα topics, τοποθετούνται σελιδοδείκτες ώστε να ξέρουν ποια είναι τα τελευταία διαβασμένα μηνύματα.

Name	Message Count	Partitions	Replication Factor	In Sync Replicas
__consumer_offsets	158	50	1	50
global-weights	72	4	1	4
handshake	2	1	1	1
local-weights	80	4	1	4

Σχήμα 2.3: Τα topics του client

Το send weights του client φαίνεται παρακάτω:

```
# Send the weights to the server
    def send_weights(self, partition, topic='local-weights'):

        weights = self.get_weights()

        metadata = {'client_info': f'partition:{partition}',
                    'weights': weights}
        formatted_message = pickle.dumps(metadata)
        self.kafka_producer.send(topic, value=formatted_message,
                                 partition=partition)
        self.kafka_producer.flush()
        print(f"Weights sent successfully to partition {partition}")
```

Ομοίως, το receive weights του server:

```
# read the weights on the local topic,read from the last position
    def recieve_weights(self, topic='local-weights'):
        local_weights = []
        for partition in range(self.n_clients):

            assigned_partition = [TopicPartition(topic, partition)]
            self.consumer.assign(assigned_partition)

            # you read all the messages that are directed for you and
you store them in a list
            for message in self.consumer:

                data = pickle.loads(message.value)
                partition_id = data['client_info']
                local_weight = data['weights']
                local_weights.append(local_weight)
                print(f"got the message from {partition_id} on
{partition}")
                self.consumer.commit()
                break

        return local_weights
```

Για να ολοκληρωθεί η επικοινωνία και η εκπαίδευση, το «ping-pong» κρατάει ορισμένους γύρους που ορίζονται ανάλογα τη πολυπλοκότητα του μοντέλου και των δεδομένων.

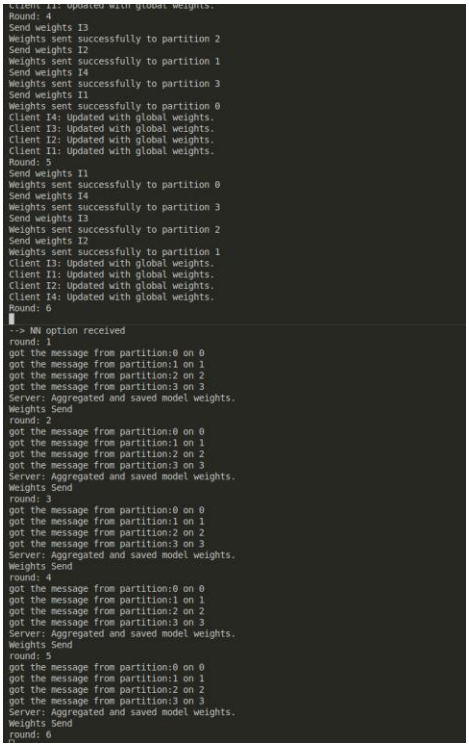
Server initialization:

```
def initialize_server(self):
        # initialzie the communication form the server side(start the
"ping pong")
        rounds = 10
        for round in range(rounds):
            print(f"round: {round+1}")
            local_weights_list = self.recieve_weights()
            self.aggregate_weights(local_weights_list)

            # in the last round we just need the global model dont need
to update the local ones
            if round < (rounds-1):
                self.send_weights_back()
            else:
                self.consumer.close()

                self.producer.flush()
                self.producer.close()
```

Client initialization:

```
def initialize_nn(self, partition):
        # Initialize the communication from the client side
        rounds = 10
        starting_epochs = 100

        if partition == 0:
            print(f"Running the program for {
                  rounds} rounds and {starting_epochs} epochs")

        for round in range(rounds):

            if partition == 0:
                print(f"Round: {round + 1}")

            train_losses, val_losses = self.training(
                starting_epochs-10*round)  # Get losses from training
            self.save_loss_plot(train_losses, val_losses,
                                round + 1)  # Save the plot

            print(f"Send weights {self.light_id}")
            self.send_weights(partition=partition)

            # Last round doesn't need to receive any messages, just
need the global model
            if round < (rounds - 1):
                self.receive_global_weights(partition=partition)
            else:
                self.kafka_consumer.close()

                self.kafka_producer.flush()
                self.kafka_producer.close()
```

Σχόλιο: Στο τελευταίο γύρο, ο client θέλει να στείλει τα βάρη και να μη πάρει τίποτα πίσω και ο server θέλει να διαβάσει τα τελικά βάρη ώστε να δημιουργήσει το τελικό ολικό μοντέλο.



Screenshot 2.4: Snippet από την επικοινωνία

Τέλος, μετά την εκπαίδευση και τη δημιουργία ολικού μοντέλου με τα αρχεία model_eval.py, gui.py, neural_tester.py ο χρήστης μπορεί να:

model_eval.py: Αρχικός τρόπος αξιολόγησης αποτελεσμάτων του αρχείου. Υπολογίζει τα MSE και MAE για κάθε φανάρι και εκτυπώνει γραφικές παραστάσεις των προβλέψεων σε σύγκριση των υπαρχουσών τιμών στα dataset.

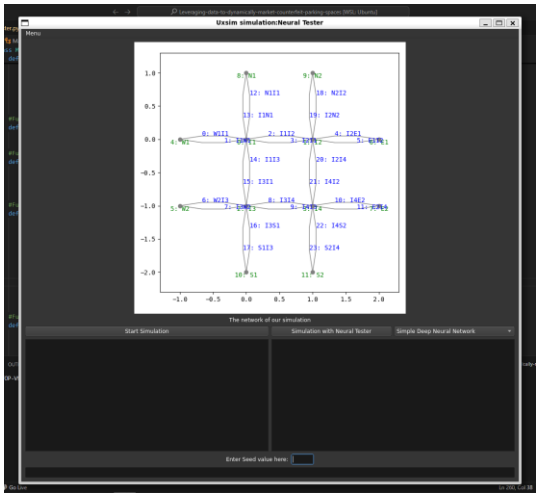
neural_tester.py: Το περιβάλλον εξομοίωσης της λειτουργίας των φαναριών. Βασικός του σκοπός είναι να τρέξει το πρόγραμμα και να παράγει δεδομένα για τις δύο εξομοιώσεις, η μία η οποία είναι η κανονική εξομοίωση και η δεύτερη είναι η εξομοίωση που οι τιμές των χρονισμών των φαναριών προβλέπονται από το μοντέλο.

gui.py: Το γραφικό περιβάλλον της εφαρμογής που χρησιμοποιεί το neural_tester για να παρουσιάσει τα αποτελέσματα. Μέσα από αυτό μπορεί να τρέξει μια απλή προσομοίωση με σταθερούς χρονισμούς είτε με δυναμικούς χρονισμούς επιλέγοντας ένα από τα δύο διαθέσιμα Neural Networks. Επιπλέον εμφανίζεται κάτοψη του δικτύου δρόμου δείχνοντας την πυκνότητα της κίνησης κάθε δρόμου μαζί με τα αποτελέσματα του simulation. Στο γραφικό περιβάλλον έχουμε προσθέσει και επιπλέον λειτουργείες που υπάρχουν στο Menu. Εκεί ο χρήστης μπορεί να με μεγαλύτερη λεπτομέρεια (animation) τα αποτελέσματα των simulations. Τέλος υπάρχει η επιλογή να γίνει το simulation για 100 seeds τόσο για

τον σταθερό χρονισμό όσο και για το δυναμικό μέσο του επιλεγμένου νευρωνικού δικτύου.

Για την έναρξη του kafka και την δημιουργία των Topics έχουμε το αρχείο run_traffic_system.sh.

Σημείωση: Επειδή το Run for many seeds στο gui εκτελείται για 100 seeds που τρέχουν ταυτόχρονα, μπορεί ορισμένοι υπολογιστές (λόγω έλλειψης cuda cores και πολλών thread του cpu) (καθώς και το dataset_creation.py) να δυσκολεύονται να τα εκτελέσουν, ή και να αποτύχουν εντελώς, επειδή είναι από τη φύση τους υπολογιστικά βαριές εργασίες.



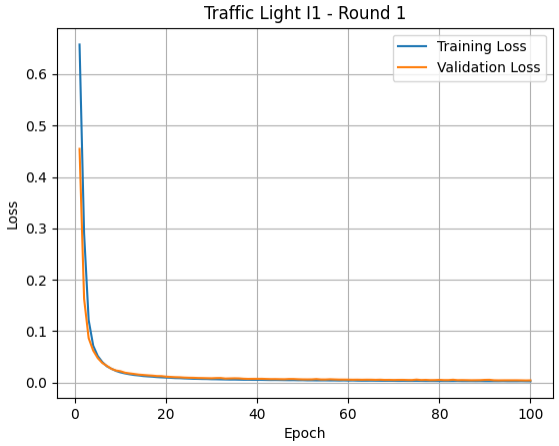
Screenshot 2.5: Graphical User Interface

Τα screenshots για τους υπόλοιπους κώδικες βρίσκονται στο παρακάτω μέρος της αναφοράς («Αποτελέσματα δοκιμών λειτουργίας») καθώς αποτελούν μέρος της αξιολόγησης του προγράμματος.

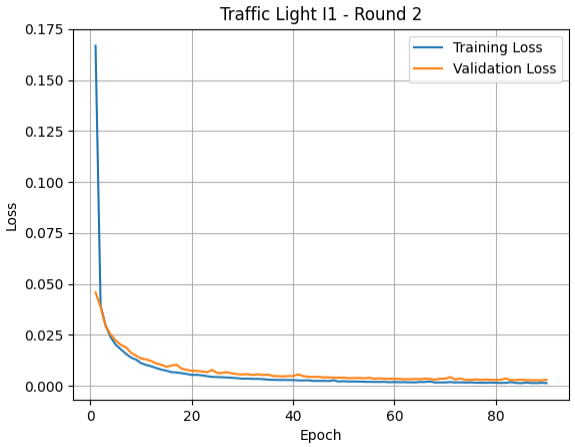
3. Ευρήματα υλοποίησης τελικής εφαρμογής

3.1. Αποτελέσματα δοκιμών λειτουργίας:

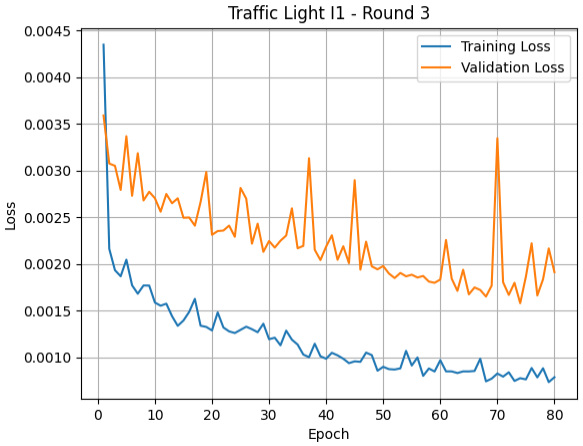
Αρχικά, για την εκπαίδευση των νευρωνικών δικτύων, όλα τα φανάρια εκπαιδεύονται σωστά (δηλαδή δεν υπάρχει underfitting/overfitting) και χρησιμοποιώντας το model_eval.py υπολογίζουμε τα MSE και MAE , που είναι πολύ μικρά (της τάξης 10 και 10-1)



3.1



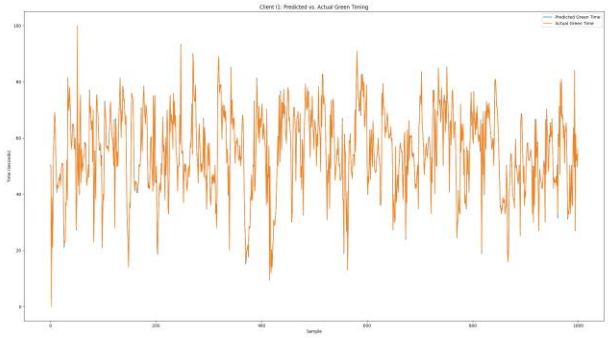
3.2



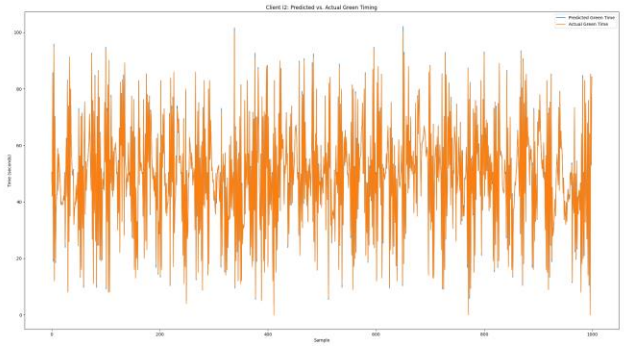
3.3

Σχήματα 3.1, 3.2, 3.3: Πρώτος, δεύτερος και τρίτος γύρος αντίστοιχα

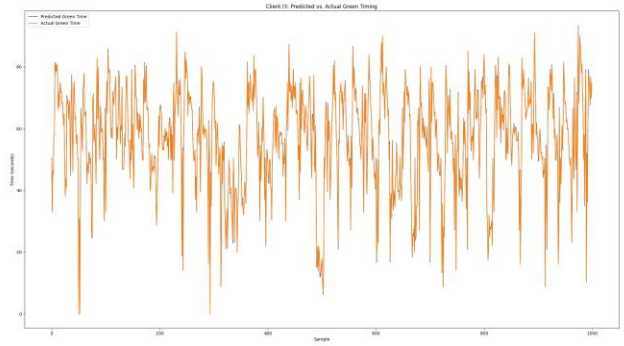
Λόγω μεγάλου πλήθους screenshot, περισσότερα μπορούν να βρεθούν στον φάκελο “plots”



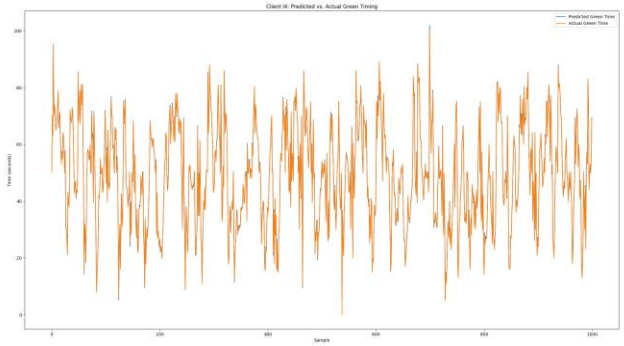
3.4



3.5



3.6



3.7

```
1) Simple Deep Neural Network
2) Gated Recurrent unit (GRU)
Select an option: 1

Evaluating model for I1...
Client I1 - MSE: 0.2186, MAE: 0.3429

Evaluating model for I2...
Client I2 - MSE: 0.2776, MAE: 0.3845

Evaluating model for I3...
Client I3 - MSE: 0.3222, MAE: 0.3658

Evaluating model for I4...
Client I4 - MSE: 0.2074, MAE: 0.3463
```

3.8: MSE και MAE για το Simple Deep Neural Network

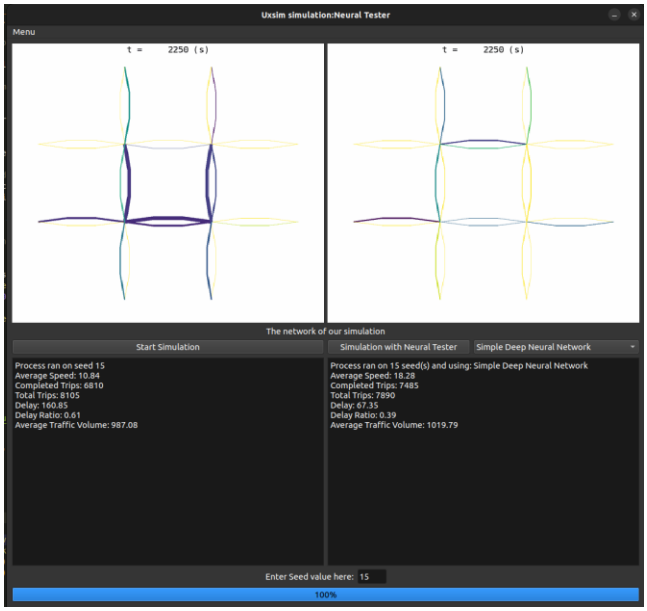
```
Select a neural network to evaluate:
1) Simple Deep Neural Network
2) Gated Recurrent unit (GRU)
Select an option: 2

Evaluating model for I1...
Client I1 - MSE: 0.1115, MAE: 0.2671

Evaluating model for I2...
Client I2 - MSE: 0.1062, MAE: 0.2707

Evaluating model for I3...
Client I3 - MSE: 0.1318, MAE: 0.2692

Evaluating model for I4...
Client I4 - MSE: 0.1049, MAE: 0.2677
```

3.9: MSE και MAE για το Gated Recurrent Unit

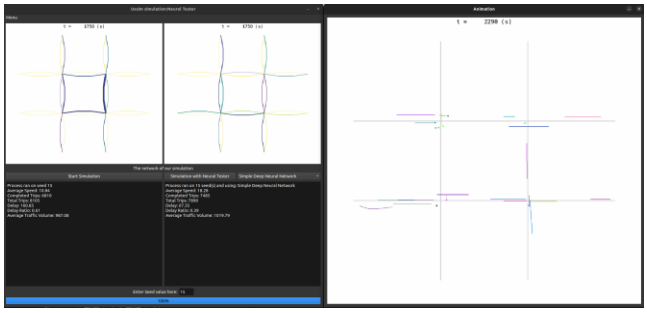
Σχήματα 3.4, 3.5, 3.6, 3.7: Οι προβλέψεις των χρονισμών για τα I1,I2,I3 και I4 αντίστοιχα σε σύγκριση με τις τιμές των dataset.

Φαίνεται από όλα τα παραδείγματα πως και για τα 2 νευρωνικά δίκτυα οι προβλέψεις είναι αρκετά ακριβείς.

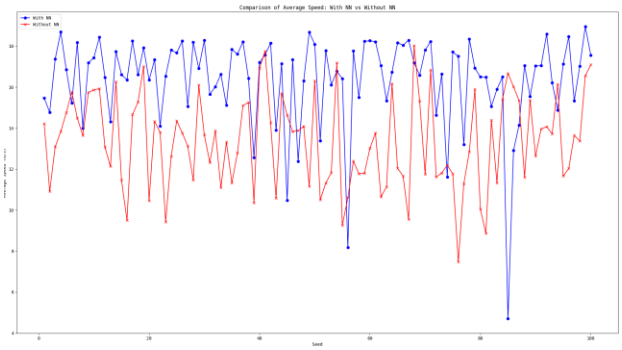
Σχόλιο: Οι εικόνες από 3.4 έως 3.7 είναι μόνο για το DeepNN.



Σχήμα 3.10: Από τη παραπάνω εικόνα, φαίνονται οι διαφορές στο νευρωνικό δίκτυο. Οι δρόμοι έχουν ισοκατανεμημένη κίνηση κυκλοφορίας (αριστερά: απλό simulation, δεξιά: simulation με νευρωνικό δίκτυο)



Σχήμα 3.11: Μια πιο πλήρης εικόνα του gui και των γραφικών παραστάσεων του δρόμου. Στα δεξιά φαίνεται το δίκτυο με χρήση νευρωνικού δικτύου.



Σχήμα 3.12: Σύγκριση μέσων ταχυτήτων για 100 seeds όπως φαίνεται στη πληθώρα των παραδειγμάτων, το νευρωνικό δίκτυο παράγει καλύτερα αποτελέσματα(παρόμοια για το GRU).

3.2. Συζήτηση και σχολιασμός αποτελεσμάτων:

Το γενικό αποτέλεσμα είναι πως ο διαμοιρασμός της εργασίας μας κερδίζει χρόνο σε σύγκριση με ένα γενικό μοντέλο που καλύπτει όλα τα φανάρια ενώ παράλληλα δεν έχει μεγάλη απώλεια στην ακρίβεια των προβλέψεων. Όμως, η διαδικασία σύνδεσης εκτίμησης και επικοινωνίας των φαναριών προσθέτει μια πιο σύνθετη πολυπλοκότητα στην εφαρμογή που δεν υπάρχει στο απλό μοντέλο.

4. Συμπεράσματα

Συμπεράνουμε πως σε μεγάλα συστήματα όπως το δικό μας, η υλοποίηση federated learning έχει περισσότερα προνόμια παρά αδυναμίες καθώς ο κερδισμένος χρόνος μαζί με τη μικρή διαφορά στην απώλεια ακρίβειας των προβλέψεων υπερτερεί της πολυπλοκότητας της διασύνδεσης. Παράλληλα, ένα τέτοιο μοντέλο βελτιστοποιεί τις συνθήκες κάθε τοπικού κόμβου, σε σύγκριση με ένα μοντέλο που εκπαιδεύεται μόνο στο ολικό δίκτυο.

5. Προτάσεις για βελτίωση και μελλοντικές επεκτάσεις

Αρχικά, μπορεί να βελτιωθεί ο τρόπος με των οποίο συλλέγονται τα δεδομένα, να είναι πιο ακριβής και να συλλέγει περισσότερα δεδομένα. Από την άποψη του μοντέλου του νευρωνικού δικτύου, να χρησιμοποιηθεί ένα που είναι σχεδιασμένο για αυτού του τύπου των προβλέψεων (όταν οι μεταβλητές είναι τοπικά εξαρτημένες, υπάρχουν πολλές εξαρτήσεις μεταξύ των οχημάτων κ.τ.λ.) Βελτιστοποίηση του συστήματος διαχείρισης φαναριών με τον server (buffer, network optimization) και κάποιο καλύτερο πρωτόκολλο επικοινωνίας. Έπειτα, από την άποψη του server, να έχει καλύτερο αλγόριθμο τύπου federated averaging. Καλύτεροι τρόποι evaluation και testing για την απόδοση του νευρωνικού δικτύου. Τέλος πιο user optimized GUI με περισσότερες δυνατότητες.

6. Τεχνικά χαρακτηριστικά περιβάλλοντος λειτουργίας

Χαρακτηριστικό	Τιμή
CPU model	AMD Ryzen™ 7 7700X Desktop
CPU clock speed	5.4 GHz
Physical CPU cores	8
Logical CPU cores	16
RAM	32 GB
Secondary Storage Type	SSD
Operation System	Linux Ubuntu 22.04

7. Εκπόνηση

Για την εκπόνηση της εργασίας όλα τα μέλη εργάστηκαν πάνω σε όλα τα κομμάτια της, δημιουργία dataset, νευρωνικά δίκτυα, εκπαίδευση νευρωνικών δικτύων, επικοινωνία μέσω kafka, gui και αξιολόγηση.

Βιβλιογραφία

- https://pytorch.org/
- https://en.wikipedia.org/wiki/Federated_learning
- https://en.wikipedia.org/wiki/Deep_learning
- https://en.wikipedia.org/wiki/Gated_recurrent_unit
- https://pytorch.org/docs/stable/generated/torch.nn.GRU.html
- https://pytorch.org/tutorials/beginner/basics/buildmodel_tutorial.html
- Yesha Shastri,2023, A Step-by-Step Guide to Federated Learning in Computer Vision, https://www.v7labs.com/blog/federated-learning-guide#:~:text=Federated%20learning%20(often%20referred%20to,model%20locally%2C%20increasing%20data%20privacy