T.C.
IŞIK UNIVERSITY
SCHOOL OF GRADUATE STUDIES

MASTER THESIS
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COMPUTER ENGINEERING PROGRAM

Musa ŞİMŞEK

FEDERATED HYBRID PRIVACY-PRESERVING MOVIE RECOMMENDATION SYSTEM FOR INTERNET-OF-VEHICLES

SUPERVISOR
Asst. Prof. Dr. Ayşegül ERMAN TÜYSÜZ

İSTANBUL, August 2024

T.C.
IŞIK UNIVERSITY
SCHOOL OF GRADUATE STUDIES

MASTER THESIS
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COMPUTER ENGINEERING PROGRAM

Musa ŞİMŞEK

(22COMP5002)

FEDERATED HYBRID PRIVACY-PRESERVING MOVIE RECOMMENDATION SYSTEM FOR INTERNET-OF-VEHICLES

SUPERVISOR

Asst. Prof. Dr. Ayşegül ERMAN TÜYSÜZ

İSTANBUL, August 2024

T.C.
IŞIK UNIVERSITY
SCHOOL OF GRADUATE STUDIES

MASTER THESIS
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
COMPUTER ENGINEERING PROGRAM

Musa ŞİMŞEK

(22COMP5002)

FEDERATED HYBRID PRIVACY-PRESERVING MOVIE RECOMMENDATION SYSTEM FOR INTERNET-OF-VEHICLES

Date: 02/08/2024

Thesis Supervisor:
Asst. Prof. Dr. Ayşegül ERMAN TÜYSÜZ / Istanbul University

Jury Members:
Asst. Prof. Dr. Emine EKIN / Işık University
Asst. Prof. Dr. Rahim DEHKHARGHANI / Kadir Has University

İSTANBUL, August 2024

ÖZET

ARAÇLARIN İNTERNETİNDE FEDERE HİBRİT GİZLİLİK KORUMALI FİLM ÖNERİ SİSTEMİ

Bu araştırmada, araç içi film öneri sistemleriyle ilgili gizlilik endişelerinin üstesinden gelmek için yenilikçi bir strateji tanıttık. Günümüzde, kullanıcıların kişisel verilerinin korunması, özellikle araç içi eğlence sistemleri gibi hassas alanlarda büyük bir önem taşımaktadır. Bu bağlamda, PyTorch çerçevesi kullanılarak sıfırdan bir temel oluşturulmuştur. Eğitim sürecinde, Laplace gürültü eklenmesi yöntemi, diferansiyel gizliliği sağlarken aynı zamanda model performansını optimize etmektedir. Bu yöntem, kullanıcı verilerini koruma amacını güderken, öneri sisteminin doğruluğunu da artırmaktadır. Ayrıca, Optuna hiperparametre optimizasyon çerçevesi, modelin performansını daha da geliştirmekte ve sistemin genel verimliliğini artırmaktadır.

Movielens-1M referans film veri kümesini kullanarak gerçekleştirdiğimiz kapsamlı deneyler, yaklaşımımızın öneri doğruluğundan ödün vermeden kullanıcı gizliliğini korumadaki etkinliğini ortaya koymuştur. Elde ettiğimiz sonuçlar, temel modellere göre kayda değer bir iyileşme göstermekte ve gizliliği koruyan araç film öneri sistemimizin etkinliğini doğrulamaktadır. Ayrıca, merkezi film öneri sistemimizi FedAvg, FedProx ve FedMedian gibi pratik federe çerçevelerle kapsamlı bir şekilde karşılaştırdık. Bulgularımız, tüm federe modellerin daha kısa çalışma süreleriyle merkezi modellere göre en az %2 daha iyi performans gösterdiğini ve öneri kalitesinden ödün vermeden sistem verimliliğini artırdığını ortaya koymaktadır. Bu sonuçlar, gelecekteki araştırmalar için önemli bir temel oluşturarak, kullanıcı gizliliği ile sistem performansı arasında bir denge kurmanın mümkün olduğunu göstermektedir.

Anahtar Kelimeler: Federe Öğrenme, Hiperparametre Optimizasyonu, Araçların İnterneti, Gizlilik Güçlendirici Teknolojiler, Öneri Sistemleri**.**

ABSTRACT

FEDERATED HYBRID PRIVACY-PRESERVING MOVIE RECOMMENDATION SYSTEM FOR INTERNET-OF-VEHICLES

In this research, we introduced a pioneering strategy to address the pressing privacy concerns associated with vehicular movie recommendation systems. As the demand for personalized entertainment options in vehicles increases, so does the need to protect user data. To tackle this challenge, we utilized the PyTorch framework to create a robust foundation from scratch. A key component of our approach was the addition of Laplace noise during the training process, which ensured differential privacy. This technique effectively safeguarded user data while simultaneously optimizing model performance, allowing us to maintain high levels of recommendation accuracy.

Furthermore, we employed the Optuna hyperparameter optimization framework, which played a crucial role in enhancing the model's performance. By fine-tuning various parameters, we were able to elevate the overall efficiency of the system beyond the capabilities of the base model. Our extensive experimentation utilized the Movielens-1M benchmark movie dataset, which provided a solid basis for evaluating our approach. The results demonstrated a significant improvement over baseline models, validating the effectiveness of our privacy-preserving vehicular movie recommendation system.

In addition to our centralised model, we conducted a comprehensive comparison with practical federated frameworks, including FedAvg, FedProx, and FedMedian. Our findings revealed that all federated models outperformed the centralised models by at least 2%, while also exhibiting shorter runtimes.

Keywords: Federated Learning, Hyperparameter Optimisation, Internet-of-Vehicles, Privacy Enhancing Technologies, Recommender Systems.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to my thesis advisor Asst. Prof. Ayşegül Erman Tüysüz, for her invaluable guidance and support throughout the process of completing my thesis. Her expertise, encouragement, and perseverance have been greatly affected in shaping the outcome of this thesis. I am truly thankful for the knowledge and inspiration we have shared, which have been valuable in my academic journey.

Musa ŞİMŞEK

TABLE OF CONTENTS

PAGE NO

APPROVAL PAGE i

ÖZET ii

ABSTRACT iii

ACKNOWLEDGEMENT iv

TABLE OF CONTENTS v

LIST OF FIGURES vi

LIST OF TABLES vii

ABBREVIATIONS LIST viii

CHAPTER 1 1

1. THE PURPOSE OF THE RESEARCH 1

CHAPTER 2 3

2. INTRODUCTION 3

2.1 INTERNET OF THINGS 3

2.2 INTERNET OF VEHICLES 4

2.3 VEHICULAR INFOTAINMENT SYSTEMS 7

2.4 DESIGNING AN ETHICAL MACHINE LEARNING MODEL 8

2.5 DIFFERENTIAL PRIVACY 9

2.6 HYPERPARAMETER OPTIMISATION 10

2.7 CENTRALISED LEARNING 10

2.8 FEDERATED LEARNING 10

2.9 VEHICULAR FEDERATED LEARNING 12

2.10 HYBRID RECOMMENDATION SYSTEMS 12

2.11 CENTRALISED HYBRID MOVIE RECOMMENDATION SYSTEMS 13

2.12 FEDERATED HYBRID MOVIE RECOMMENDATION SYSTEMS 14

2.13 COMPARING CENTRALISED AND FEDERATED HYBRID MOVIE RECOMMENDATION SYSTEMS 14

CHAPTER 3 16

3. LITERATURE 16

3.1 RELATED WORK 16

CHAPTER 4 20

4. RESEARCH METHOD 20

4.1 SYSTEM OVERVIEW 20

4.1.1 Mutual Core Features 20

4.1.2 Mutual Quality Features 24

4.1.3 Adaptive Moment Estimation (ADAM) Optimiser 25

4.1.4 L2 Regularisation 25

4.1.5 Mean Squared Error (MSE) Loss Function 26

4.2 SYSTEM ARCHITECTURE 26

4.2.1 Data Pre-Processing 28

4.2.2 Hyperparameters 28

4.2.2.1 Mutual Hyperparameters 28

4.2.2.2 Unique Hyperparameters – Federated Learning 31

4.2.2.3 Mutual Hyperparameters for Early Stopping Mechanism 33

4.2.2.4 Mutual Hyperparameters for Differential Privacy 34

4.2.2.5 Mutual Hyperparameters for Hyperparameter Optimisation 35

4.2.3 Hybrid Recommendation Model 36

4.2.3.1 User-Item Embeddings (Matrix Factorisation).....37
4.2.3.2 User-Item Embeddings (Multi-Layer Perceptron).....38
4.2.3.3 LSTM (Long-Short Term Memory).....38
4.2.3.4 Kaiming (He) Initialisation.....39
4.2.3.5 Fully Connected (Dense) Layers.....39
4.2.3.6 Forward Pass Computation.....41
4.2.3.7 Backward Pass Computation.....42
4.2.3.8 L2 Regularisation.....42
4.2.3.9 Adam Optimiser.....43
4.2.3.10 MSE Loss Function.....43
4.2.3.11 Calculating Error.....44
4.2.3.12 Centralised Training/Validation/Testing Phases.....45
4.2.3.13 Federated Training/Validation/Testing Phases.....47

4.3 DIFFERENTIAL PRIVACY.....49

4.4 HYPERPARAMETER OPTIMISATION.....50

CHAPTER 5.....52

5. RESEARCH FINDINGS.....52

5.1 EXPERIMENTATION ENVIRONMENT.....52

5.2 PERFORMANCE METRICS.....53

5.3 EVALUATION.....54

5.3.1 Case-by-Case Analysis.....55

5.3.2 Comparative Analysis.....65

CONCLUSION AND RECOMMENDATIONS.....71

REFERENCES.....74

APPENDICES.....77

CURRICULUM VITAE.....81

LIST OF FIGURES

Figure 2.1 Internet-of-Things Architecture.……4

Figure 2.2 Internet-of-Vehicles Architecture.……5

Figure 4.1 Matrix Factorisation Interaction.……..….21

Figure 4.2 Multi-Layer Perceptron.……...23

Figure 4.3 Comparing Collaborative Filtering with Content-Based Filtering...24

Figure 4.4 Data Pre-processing.……27

Figure 4.5 Hybrid Recommendation Neural Network Model.……..….35

Figure 4.6 Defining Kaiming (He) Initialisation.……..….39

Figure 4.7 Defining Fully Connected (Dense) Layers.……..….40

Figure 4.8 Enhancing Adam Optimiser with L2 Regularisation.……...43

Figure 4.9 Early Stopping Mechanism in Centralised Model.……...47

Figure 4.10 Early Stopping Mechanism in Federated Model.……...48

Figure 4.11 Implementing Differential Privacy with Laplace Noise.……49

Figure 4.12 Optuna Hyperparameter Optimisation Implementation.……50

Figure 5.1 Centralised Core.……55

Figure 5.2 Centralised Core with Differential Privacy in Lower and Higher Privacy Settings.…….…….…….……56

Figure 5.3 Centralised Core with Hyperparameter Optimisation.……56

Figure 5.4 Centralised Core with Hyperparameter Optimisation and Differential Privacy in Lower and Higher Privacy Settings.……..….57

Figure 5.5 Federated Core (S2: AFO, S3: FedAvgTrimmedMean)..……58

Figure 5.6 Federated Core (S4: FedAvg, S5: FedAvgMomentum).……..….58

Figure 5.7 Federated Core (S6: FedMedian, S7: FedProx)……58

Figure 5.8 Federated Core with Differential Privacy in Lower Privacy Settings (S10: AFO, S11: FedAvgTrimmedMean)..……59

Figure 5.9 Federated Core with Differential Privacy in Lower Privacy Settings (S12: FedAvg, S13: FedAvgMomentum).……59

Figure 5.10 Federated Core with Differential Privacy in Lower Privacy Settings (S14: FedMedian, S15: FedProx).……..….59

Figure 5.11 Federated Core with Differential Privacy in Higher Privacy Settings (S16: AFO, S17: FedAvgTrimmedMean)..……60

Figure 5.12 Federated Core with Differential Privacy in Higher Privacy Settings (S18: FedAvg, S19: FedAvgMomentum)..……60

Figure 5.13 Federated Core with Differential Privacy in Higher Privacy Settings (S20: FedMedian, S21: FedProx).……..….61

Figure 5.14 Federated Core with Hyperparameter Optimisation (S23: AFO, S24: FedAvgTrimmedMean).……61

Figure 5.15 Federated Core with Hyperparameter Optimisation (S25: FedAvg, S26: FedAvgMomentum).…….62

Figure 5.16 Federated Core with Hyperparameter Optimisation (S27: FedMedian, S28: FedProx).……...62

Figure 5.17 Federated Core with Hyperparameter Optimisation with Differential Privacy in Lower Privacy Settings (S31: AFO, S32: FedAvgTrimmedMean).……63

Figure 5.18 Federated Core with Hyperparameter Optimisation with Differential Privacy in Lower Privacy Settings (S33: FedAvg, S34: FedAvgMomentum).…….63

Figure 5.19 Federated Core with Hyperparameter Optimisation with Differential Privacy in Lower Privacy Settings (S35: FedMedian, S36: FedProx).……...63

Figure 5.20 Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings (S37: AFO, S38: FedAvgTrimmedMean, S39: FedAvg).……64

Figure 5.21 Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings (S39: FedAvg, S40: FedAvgMomentum)...64

Figure 5.22 Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings (S41: FedMedian, S42: FedProx).……...65

Figure 5.23 Centralised Core vs. Federated Core……..….66

Figure 5.24 Centralised Core with Differential Privacy in Lower Privacy Settings vs. Federated Core with Differential Privacy in Lower Privacy Settings ……...66

Figure 5.25 Centralised Core with Differential Privacy in Higher Privacy Settings vs. Federated Core with Differential Privacy in Higher Privacy Settings ……...67

Figure 5.26 Centralised Core with Hyperparameter Optimisation vs. Federated Core with Hyperparameter Optimisation……...68

Figure 5.27 Centralised Core with HPO and DP in Lower Privacy Settings vs. Federated Core with HPO and DP in Lower Privacy Settings……....69

Figure 5.28 Centralised Core with HPO and DP in Higher Privacy Settings vs. Federated Core with HPO and DP in Higher Privacy Settings……...70

LIST OF TABLES

Table 2.1 Comparison between Centralised and Federated Architecture of Hybrid Recommendation Systems……15

Table 4.1 Hyperparameter Optimisation Search Space…….35

Table 4.2 Comparison between Group Normalisation and Batch Normalisation ……...41

Table 4.3 Comparison between MAE and MSE……...45

Table 5.1 Average Computing Runtimes……..….53

ABBREVIATIONS LIST

AFO: Adaptive Federated Optimisation

DP: Differential Privacy

FedAvg: Federated Averaging

FedAvgMean: Federated Averaging with Trimmed Mean

FedAvgMomentum: Federated Averaging with Momentum

FedMedian: Federated Median

FedProx: Federated Proximal

HPO: Hyperparameter Optimisation

IoT: Internet-of-Things

IoV: Internet-of-Vehicles

MF: Matrix Factorisation

MLP: Multi-layer Perceptron

MAE: Mean Absolute Error

MSE: Mean Squared Error

NDCG: Normalized Discounted Cumulative Gain

LSTM: Long Short-Term Memory

CHAPTER 1

1. THE PURPOSE OF THE RESEARCH

In the expanding landscape of recommendation systems, privacy considerations have become crucial. As users engage with these systems to discover content tailored to their tastes, safeguarding personal information has emerged as a significant concern. This concern goes beyond personalised recommendations; it requires balancing user experience with the ethical responsibility of protecting privacy. Furthermore, the relevant efforts become more nuanced in movie recommendation systems within the Internet of Vehicles (IoV) domain. As vehicles become interconnected, the challenge of providing personalised movie suggestions while safeguarding data privacy intensifies. Consequently, IoV infotainment systems must navigate user preferences while ensuring confidentiality, integrity, and availability of personal information. In this context, recommending personalised movie content generally involves processing and analysing user behaviour and preferences. However, due to the potential risks of revealing user preferences in a connected vehicular environment, implementing a robust hybrid movie recommendation system is imperative to address both ethical and practical concerns.

Considering all the essential criteria outlined above in the context of the hybrid movie recommendation systems for vehicles, the main contributions of our research are as follows:

- We have proposed a hybrid neural network architecture-based movie recommendation system that integrates matrix factorisation (MF), Multi-Layer Perceptron (MLP), and Long Short-Term Memory (LSTM) to comprehensively capture diverse aspects of the input data and learn intricate patterns.
- To gain a thorough understanding of the overall differences, we established and compared our centralised and federated architectures

under identical settings. The primary objective is to create a federated environment that outperforms, or at least matches, the centralised architecture. The comparison criteria include:

  - By architecture: Centralised / Federated
  - By fine-tuning hyperparameters of the model
  - By aggregation strategies: Federated Averaging (FedAvg), Federated Averaging with Trimmed Mean, Federated Averaging with Momentum, Federated Proximal (FedProx), Federated Median (FedMedian), Adaptive Federated Optimisation (AFO).
  - By metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), Normalised Discounted Cumulative Gain (NDCG)

- We also apply the following frameworks to both set-ups to demonstrate their effect under differing conditions:
  - Differential privacy
  - Hyperparameter optimisation

This comprehensive evaluation research allows us to assess the performance of both architectures across different dimensions, providing insights into their strengths and weaknesses, and to answer the question of possibility of the implementation a hybrid movie recommendation system as a component of a complex federated or centralised infotainment system.

CHAPTER 2

2. INTRODUCTION

The convergence of emerging technologies, such as the Internet of Things (IoT) and smart vehicular networks, has created unprecedented opportunities for innovation. In this interconnected era, integrating entertainment services with intelligent recommendation systems promises to enhance user experiences within the Internet-of-Vehicles (IoV) ecosystem. As that ecosystem evolves, the need for intelligent systems that respect user privacy while delivering relevant content becomes increasingly paramount. This research focuses on harmonising the challenges and opportunities of federated learning, hybrid recommendation techniques, and privacy-preserving mechanisms to create a seamless, personalised movie recommendation experience for vehicle occupants. This section explores the design and implementation of such a system, highlighting the methodologies, algorithms, considerations needed to balance recommendation accuracy and user privacy within the dynamic context of IoV.

2.1 INTERNET OF THINGS

The number of Internet of Things (IoT) devices is expected to almost double, from 15.1 billion in 2020 to over 29 billion by 2030 (Zhang, 2022). These devices, which are low-power and embedded, mainly collect data. IoT creates a network where everyday objects with sensors, software, and connectivity gather and share information. This network spans areas like healthcare, smart homes, agriculture, and industry, enhancing our ability to get real-time information and insights.

As IoT devices increase, advancements like 5G technology change how data is collected, processed, and managed. These technologies allow faster and more reliable communication, enabling new real-time applications and

improving the IoT system's efficiency. However, they also bring challenges such as data security, privacy concerns, and the need for standardised protocols. Balancing innovation with user privacy protection is essential for responsible and sustainable IoT development.

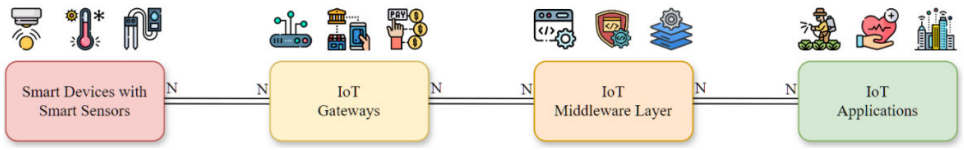

Figure 2.1 Internet-of-Things Architecture

As shown in Figure 2.1, the architecture plays a crucial role in generating insights from raw data. Smart devices with intelligent sensors capture and transmit data, forming the foundation of a network connected through IoT gateways. These gateways act as intermediaries, securely and efficiently transferring data to the next layer—the IoT middleware.

The IoT middleware serves as the system's central nervous system, managing data flow, communication protocols, and device interoperability. It harmonises data streams from various sources, creating a unified dataset. At the top of this architecture are IoT applications that use this organised data to deliver real value, whether in healthcare, smart homes, or industrial workflows. This layered architecture of devices, gateways, middleware, and applications ensures connectivity, intelligence, and functionality.

2.2 INTERNET OF VEHICLES

Within massive and expansive nature of IoT field, the IoT concept extends beyond personal devices to interconnected systems, finding various application areas. The IoV encompasses a network of connected vehicles that communicate with each other and with the infrastructure set-up to enhance safety, efficiency, and overall driving experience. To elucidate certain application domains of IoV

ecosystem, they encompass various sectors: In the domain of traffic management, IoV is instrumental in optimising traffic flow and mitigating congestion. In the domain of smart cities, IoV contributes to the enhancement of urban mobility. In the automotive industry, IoV facilitates predictive maintenance strategies. In the healthcare sector, IoV plays a vital role in ensuring timely medical assistance. Moreover, within commercial fleet management, IoV applications extend to the refinement of routes, monitoring, and fuel efficiency. Additionally, IoV introduces transformative features to in-car experiences, revolutionising the landscape of infotainment. As vehicles and users take on fundamental roles within the various IoV ecosystem, they significantly contribute to the surge of data utilised for machine learning (ML) applications developed by both individual and corporate ML engineers.

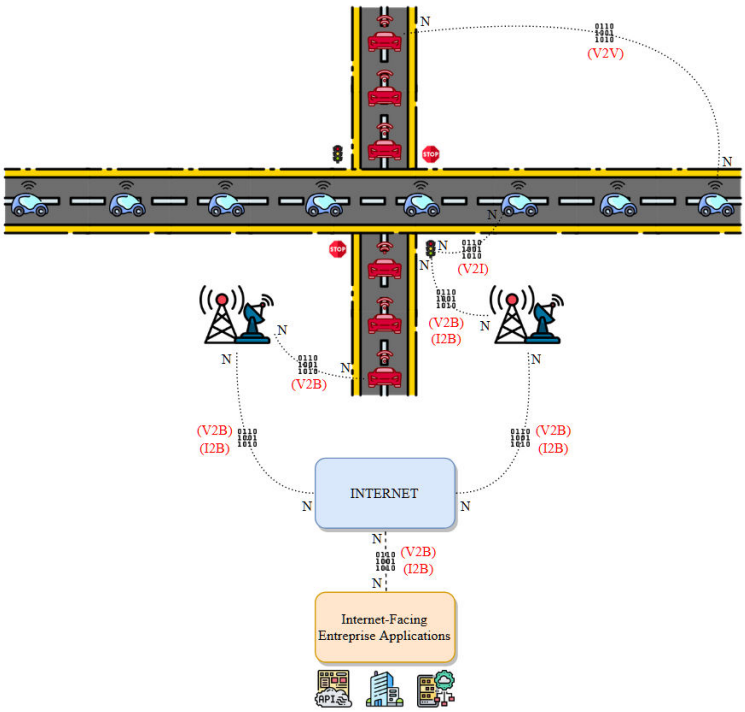

Figure 2.2 Internet-of-Vehicles Architecture

As shown in Figure 2.2, the journey of data within the IoV ecosystem commences at the sensor level, embedded within individual vehicles. These sensors, equipped on each vehicle, capture a diverse array of real-time information ranging from traffic conditions and vehicle diagnostics to environmental factors. The first layer of communication, known as Vehicle-to-Vehicle (V2V), facilitates the direct exchange of this information among vehicles within proximity. In each situation, when a vehicle identifies a sudden braking event, it has the ability to communicate this information to nearby vehicles, enabling them to take pre-emptive action.

Simultaneously, Vehicle-to-Infrastructure (V2I) communication comes into play, enabling vehicles to interact with the surrounding infrastructure. This involves communication with infrastructure components such as traffic lights, road signs, and smart intersections. Information exchanged through V2I communication can cover traffic signal timings, road conditions, and real-time updates on infrastructure changes.

Moreover, Infrastructure-to-Vehicle (I2V) communication facilitates the transmission of critical information from infrastructure to vehicles, offering insights into traffic patterns and road conditions.

As vehicles and infrastructure communicate internally, the next phase involves transmitting this data externally. This is where Vehicle-to-Base (V2B) communication becomes essential. The data collected by vehicles and infrastructure is aggregated and transmitted to base stations, often cell towers, which act as intermediaries. These base stations play a crucial role in relaying the information from the local IoV network to the broader Internet.

Once the data reaches the Internet, it enters the cloud-based applications and enterprise systems. These applications can range from centralised traffic management systems and predictive maintenance platforms to urban planning tools. In the example of a swarm of cars communicating with each other and the infrastructure, the collective data is transmitted via cell towers to the Internet, where cloud applications process and analyse the information. This aggregated data is then utilised by enterprise-level applications for numerous purposes, such

as optimising traffic flow, enhancing safety measures, and refining urban planning strategies.

2.3 VEHICULAR INFOTAINMENT SYSTEMS

The emphasis goes beyond conventional vehicle functionalities, embracing a range of interactive and entertainment features within the vehicular infotainment systems. These systems seamlessly integrate with the IoV ecosystem, providing users with enhanced in-car experiences that go beyond navigation and basic vehicle-related features. Infotainment systems offer an involved approach, incorporating features such as real-time updates, personalised content delivery, and advanced connectivity options. In the context of the IoV, these systems contribute to a holistic driving experience, blending entertainment, information, and connectivity.

From that vantage point, Augmented Reality (AR) integration stands as an attractive feature, overlaying digital information onto the real-world driving environment. This not only enriches navigation by providing intuitive visual cues but also introduces safety enhancements, such as augmented windshield displays offering real-time hazard warnings.

Voice recognition and Natural Language Processing (NLP) further amplify the user interface, enabling hands-free interactions and minimising distractions. Users can seamlessly control various infotainment functions, make calls, or even dictate messages, fostering a safer and more convenient driving experience. The integration of an extensive app ecosystem and third-party functionalities ensures versatility in content and services. From music streaming and location-based services to smart home integration, users can tailor their in-car environment to suit personal preferences, expanding the scope of the infotainment system beyond the vehicular context.

AI-powered personalisation is key, using machine learning algorithms to tailor content recommendations, optimise predictive maintenance alerts, and suggest fuel-efficient routes based on user habits and preferences. Over-the-Air

(OTA) updates ensure the infotainment system remains updated with new features and improvements, without requiring physical visits to service centres.

Collectively, these advancements make infotainment systems integral to the IoV ecosystem, offering opportunities for innovation in user-centric functionalities and enriching the data reservoir for ML applications.

2.4 DESIGNING AN ETHICAL MACHINE LEARNING MODEL

For ML applications, ensuring fairness, robustness, explainability, transparency, and privacy is integral. Fairness is vital to prevent biased outcomes and ensure equitable treatment for all users. Robust models maintain performance in unforeseen conditions, enhancing reliability. Explainability fosters trust by making the decision-making process transparent. Transparency in the ML process enhances accountability and comprehension. Preserving privacy, through techniques like data anonymisation, is crucial for ethical AI practices, preventing unauthorised access to personal information.

Alongside all up-to-date technological strides and the ethical ML principles to build a ML model with best practices applied, as IoV becomes more seamlessly integrated into the daily lives, conventional centralised approaches to data processing in the IoV raise concerns among the public regarding the privacy of the sensitive nature of vehicular data and potential exposure of their sensitive information to unauthorised governmental and private entities. Addressing these concerns becomes imperative, not only to uphold user privacy rights but also to foster user trust and promote the responsible development of ML applications on top of IoV technologies. Adopting a federated model, wherein the training and inference processes occur locally on user vehicles rather than a centralised server, guarantees that sensitive user data stays confined and protected. This approach aligns with the principles of privacy by design and mitigates the risks associated with central repositories that may become targets for unauthorised access.

2.5 DIFFERENTIAL PRIVACY

Differential privacy is a privacy-enhancing technology that aims to protect sensitive information from individual data contributors. Its main goal is to add controlled noise or randomness to the source data before creating the final model. This ensures that the impact of any single data point on the overall results is statistically insignificant, making it difficult for an outsider to determine specific contributions. Differential privacy is widely employed in various domains, including statistics, machine learning, and data analysis, and it plays a crucial role in balancing the need for accurate insights with the preservation of user privacy in an increasingly data-driven world.

More specifically, in pursuit of safeguarding user privacy, the field of differential privacy employs various methods to introduce randomness. Gaussian noise and Laplace noise, derived from the Gaussian distribution and Laplace distribution, are two common probabilistic methods frequently employed in differential privacy. Gaussian noise exhibits lighter tails, resulting in a more centralised and predictable spread of randomness. This quality is beneficial in scenarios where a balanced and moderate level of randomness is needed to protect privacy while maintaining a smoother, less disruptive impact on the overall dataset. In contrast, Laplace noise exhibits heavier tails, indicating a greater willingness to introduce extreme or unexpected values. This property adds a controlled layer of uncertainty and variability, making Laplace noise particularly suitable for scenarios where a robust level of privacy is imperative. Yet, the exponential mechanism is another privacy technique that picks outcomes based on their usefulness, using math to control how likely each outcome is to be chosen. It doesn't directly add noise to the data but introduces randomness in the decision-making process. This randomness protects privacy by making it harder to pinpoint specific outcomes, ensuring a careful balance between usefulness and privacy. Widely used in survey scenarios, randomised response is a different approach by introducing noise to individual responses through a randomised process. Although distinct from traditional noise addition,

this technique achieves privacy by obscuring individual responses while maintaining accuracy at the aggregate level. Moreover, subsampling involves randomly selecting a subset of the data. This technique enables privacy without directly perturbing every data point, proving particularly useful in scenarios where reducing the sample size has minimal impact on the outcome.

2.6 HYPERPARAMETER OPTIMISATION

Hyperparameter optimisation is essential for maximising the performance of machine learning models. It involves systematically exploring different combinations of hyperparameter values to identify the ones that optimise the model's performance on a chosen metric or objective function. Common methods include grid search, which evaluates all possible combinations within a defined array, and random search, which randomly samples hyperparameter values. More advanced techniques, such as Bayesian optimisation, use probabilistic models to guide the search toward favourable regions of the hyperparameter space. The goal is to find the best set of hyperparameters that allows the model to generalise well to new data and perform optimally for a given task, such as classification or regression.

2.7 CENTRALISED LEARNING

Centralised learning is a traditional machine learning approach where models are trained within a centralised infrastructure. Unlike federated learning, centralised learning requires transferring data to a central location. While this method simplifies the training process, concentrating sensitive information in one location may present significant privacy challenges.

2.8 FEDERATED LEARNING

Federated learning is revolutionising machine learning by offering solutions to challenges associated with centralised data processing. Unlike

traditional methods where user data is stored centrally, federated learning decentralises the process. Each user's device becomes a node for local model training, allowing the algorithm to learn without transmitting raw user data to a central server. This architecture is particularly advantageous for applications like movie recommendations, where user preferences are diverse and dynamic.

By keeping user data on local devices, federated learning addresses privacy concerns associated with centralised storage. Each user's sensitive information remains localised, complying with regulatory requirements and enhancing trust in recommendation systems. Additionally, federated learning enables adaptability crucial for movie preferences. Learning occurs locally, allowing the model to adapt to changes in user preferences over time without compromising performance.

In the same perspective, handling the federated workflow in a more performative way has given rise to various sophisticated algorithms and methodologies while addressing the limitations of centralised machine learning. Among these, McMahan et al. (2017) proposed a communication-efficient learning model from decentralised data which created the foundation for the federated learning concept. In FedAvg, clients autonomously train the global model using their locally stored data. Periodically, they synchronise with the central server by averaging the global model with weights determined during this synchronisation process. In the present era, FedAvg remains deeply rooted in tradition within its context. FedMedian, an abbreviation for Federated Median, extends the principles of FedAvg to overcome its challenges associated with data heterogeneity and non-IID (Non-Independently and Identically Distributed) data distributions across devices. FedMedian adopts a median aggregation approach, diverging from the conventional averaging of all updates. This method considers the median, offering increased robustness to outliers or devices with inconsistent data. This enhanced aggregation technique proves benefit in situations where specific devices may possess noisy or unreliable data, effectively preventing these outliers from exerting disproportionate influence on the global model and introducing a more stable and resilient approach to

aggregating updates within federated learning systems. Similarly, FedProx, an abbreviation for Federated Proximal, extends the principles of FedAvg while explicitly considering non-IID (Non-Independently and Identically Distributed) data across clients, potentially preventing suboptimal model convergence problem. FedProx incorporates a regularisation term into the training process to foster greater consistency in updates across clients. This optimisation term penalises deviations from the local model during training, with the objective of alleviating the influence of non-IID data distributions. The goal is to encourage more uniform and meaningful contributions from all clients, enhancing the overall effectiveness of the federated learning process.

2.9 VEHICULAR FEDERATED LEARNING

In vehicular federated learning, client devices collaborate to train a model which is specifically crafted for an IoV network without directly sharing the local data of clients, with a primary focus on maintaining privacy. A vehicular federated learning setup typically includes one server and multiple clients. The server chooses segments of clients from a pool of thousands of participants to train the model using their local data, this is a necessity due to communication resource constraints. Subsequently, the trained models are transmitted back to the server and combined to establish the global model.

This forward-thinking strategy ushers in an era where advancements in vehicular technologies harmonise with a strong commitment to user privacy, fostering a safer, more efficient, and privacy-aware future on the roads.

2.10 HYBRID RECOMMENDATION SYSTEMS

The introduction of hybrid recommendation algorithms marks a significant advancement in personalised content recommendations. These algorithms overcome the limitations of singular techniques by integrating various approaches. In a hybrid system, collaborative filtering and content-based

filtering are combined.

Collaborative filtering is effective in capturing user preferences based on past interactions but may struggle with sparse data or new users/items. Content-based filtering excels in recommending new items/users by analysing their features but may struggle to capture nuanced preferences. By combining these techniques, the recommendation system aims to improve accuracy and adaptability, catering to a wider range of scenarios while addressing the limitations of individual methods.

2.11 CENTRALISED HYBRID MOVIE RECOMMENDATION SYSTEMS

Traditionally, a centralised hybrid movie recommendation system combines collaborative filtering and content-based filtering techniques within a central server to provide personalised movie suggestions. User preferences and movie characteristics are gathered and processed in a centralised database, allowing the system to generate recommendations based on both user behaviour patterns and content features. While this approach has benefits, it also has drawbacks. Privacy is a major concern as user data is concentrated in one location. Users may hesitate to share preferences due to privacy concerns, leading to biased datasets. Additionally, scalability is challenging as the central server may struggle to handle increased load with a growing user base, resulting in reduced system efficiency. Handling diverse content is another challenge as the system needs frequent updates to stay relevant.

Furthermore, if the central server experiences downtime, the entire recommendation system becomes inaccessible, disrupting user experience.

2.12 FEDERATED HYBRID MOVIE RECOMMENDATION SYSTEMS

On the contrary, a federated hybrid movie recommendation system represents an innovative approach that combines federated learning with hybrid

recommendation techniques.

In addition to integrating traditional collaborative and content-based filtering methods across distributed networks, an advanced hybrid movie recommendation model may include various algorithms like matrix factorisation, LSTM, or fully connected layers. This intentional fusion aims to overcome limitations of singular approaches and leverage their combined strengths, enriching the recommendation process and providing users with a diverse range of suggestions that align with their preferences.

The primary goal of both vehicular federated learning and hybrid movie recommendation algorithms is to improve the accuracy of movie recommendations while safeguarding user data confidentiality and testing the limits of system architecture. Developing a recommendation system involves not only ensuring algorithmic efficiency but also aligning with ethical standards regarding privacy, demonstrating a strong commitment to user trust, data confidentiality, and contributing to ongoing research in the field.

2.13 COMPARING CENTRALISED AND FEDERATED HYBRID MOVIE RECOMMENDATION SYSTEMS

The choice of recommendation system architecture significantly impacts the effectiveness and efficiency of personalised content suggestions. In the realm of hybrid movie recommendation systems, selecting between centralised and federated architectures presents unique advantages and challenges. Table 2.1 illustrates this analysis, which encompasses critical factors such as data storage, privacy considerations, scalability, and collaborative learning. Understanding these distinctions is crucial for making informed decisions when designing recommendation systems that meet user preferences and adhere to current technological standards.

Table 2.1 Comparison between Centralised and Federated Architecture of Hybrid Recommendation Systems

Comparison Criterion	Centralised Hybrid Recommendation System	Federated Hybrid Recommendation System
Data Storage and Data Processing	Centralised server stores and processes user data	User data is distributed across local devices or servers, and processing occurs locally
Privacy	Higher risk of privacy concerns due to centralised data storage	Enhanced privacy as user data remains on local devices
Scalability	May face scalability issues with a growing user base, as the central server must handle increased load	Generally, more scalable as the load is distributed among local devices or servers
Data Diversity	May struggle to handle diverse content efficiently	Well-suited for handling diverse content, as data can be diverse across local nodes
Content Updates	Requires regular updates to the central server to stay relevant with evolving content	Can adapt to evolving content without a centralised update, allowing for more dynamic changes
Fault Tolerance	Vulnerable to a single point of failure; system downtime affects the entire recommendation process	Improved fault tolerance as the system can continue functioning even if some local nodes fail
Collaborative Learning	Limited collaboration among local nodes; recommendations are generated centrally	Promotes collaboration among local nodes without sharing sensitive user data, improving collaborative learning

CHAPTER 3

3. LITERATURE

3.1 RELATED WORK

In this section, we inspect the existing body of literature relevant to our research. Understanding the landscape of related work is crucial to contextualise the current state of knowledge, key findings, and methodologies.

Unlike traditional centralised methods, federated learning offers an encouraging method to generate distributed machine learning systems while ensuring privacy protection. In this process, participating clients compute local model updates based on their data, which are then aggregated to enhance a global model without centrally sharing raw data. Various aggregation algorithms have been proposed for updating global models based on these local updates. Among these, Federated Averaging (FedAvg) stands out as a widely adopted approach due to its simplicity and effectiveness. FedAvg aggregates local updates by computing their simple average, enabling collaborative model training across distributed datasets. McMahan et al. (2016) suggested a useful technique for federated learning of deep networks that is resilient to naturally occurring imbalanced and non-IID data distributions. This technique allowed high-quality models trained in comparatively few communication rounds and created a practical foundation for federated learning. McMahan et al. (2017) introduced the FedAvg algorithm based on iterative model averaging while evaluating communication efficiency of the model, since its emergence, federated learning took a lot of improvements. As a solid example, Hard et al. (2018) represented a federated recurrent neural network language model for a mobile keyboard application employing the FedAvg algorithm. This work demonstrated that training language models on client devices is feasible, advantageous and private. This federated method adapted to different writing styles and language trends,

continually improving predictive accuracy without transmitting personal information to central servers. On top of their work, Xu et al. (2023) presented a word prediction neural network language model with differential privacy for the same mobile application. Naturally, differential privacy accomplishes data privacy by arbitrarily making minor adjustments to each individual data point that have no bearing on the relevant statistics. As a result, little may be inferred about any one person from the data. Yang et al. (2022) presented two schemes to enhance privacy protection in federated learning. They introduced Kalman filtering, a statistical algorithm, on top of differential privacy mechanism. They succeeded to increase model accuracy by decreasing the noise caused the traditional differential privacy process. For a federated environment, communication overhead, convergence speed and accuracy are some of the essential evaluation criteria. Chen et al. (2020) studied an adaptive gradient method which may assure the convergence and the communication effectiveness. Particularly, they demonstrated that, in contrast to SGD, a naive synthesis of intermittent model averaging, and adaptive gradient methods may not lead to convergence. Wang et al. (2020) proposed FAVOR, a framework to accelerate convergence by balancing the bias caused by non-IID data by intelligently selecting the client devices to participate in each round of federated learning. Mills et al. (2020) introduced a communication-efficient framework, CE-FedAvg. Their study demonstrated a convergence to the target accuracy in less training rounds in shorter training time while having less communication overhead compared to similarly compressed FedAvg. Lui et al. (2019) introduced HierFAVG, a hierarchical federated averaging algorithm. Their study showed that their proposed schema reaches a desired model accuracy with less communication overhead for non-IID user data as clients first pushes their model weights to interim servers before they are sent to the main parameter server in two predefined periods as global model converges faster as aggregation frequency increases. Duan et al. (2020) introduced Astraea, a self-balancing federated learning framework, to tackle the problem related to accuracy degradation and biases in model training caused by imbalanced dataset.

Compared with the state-of-the-art FedAvg algorithm, Astraea increased model accuracy around 5% on the imbalanced EMNIST and CINIC-10 datasets while their trial results showing that the global imbalanced training data leads to about 8% accuracy loss for FedAvg. Jiang et al. (2020) presented a bandwidth-aware federated learning model. In their research, a segmented gossip aggregation mechanism is proposed, and training time is decreased by up to 18 times with complete use of node-to-node bandwidth to improve the communication time.

Because the client selection and networking architecture to enable federated learning in dynamic vehicular environment are largely important, Bao et al. (2020) presented a study as edge computing-based fuzzy logic client selection for vehicular IoT. The proposed schema creates a network schema where selected vehicles are assigned as edge node devices. In this distributed approach, selected nodes are learning optimal behaviours from environment interactions and clients are placed as information forwarder nodes in the vehicle network.

Alongside of the FedAvg, FedProx is also another popular federated learning framework. Li et al. (2020) studied federated optimisation in heterogenous networks. They introduced FedProx algorithm to eliminate heterogeneity problem in federated networks. The results of their work showed a drastic improvement in test accuracy by 22% on average even though FedProx is only a minor modified version of FedAvg algorithm. In another study, Zeng et al. (2023) studied a FedProx-based federated learning algorithm based on homomorphic encryption for Internet-of-Vehicles. They aimed to improve data privacy by adding another protection mechanism to the data aggregation process. The proposed image-based federated model is designed to identify emergency vehicles while comparing the results by accuracy and loss.

Moreover, FedAvgMomentum functions similarly to FedAvg but includes momentum factor in the model aggregation phase. In this approach, each client computes a model update based on its local data and applies momentum to this update before transmitting it to the server. The server then aggregates these momentum-adjusted updates to update the global model. Specifically,

momentum is an optimisation technique utilised to accelerate convergence during the training of machine learning models. It's particularly supportive in scenarios where the optimisation landscape exhibits noisy gradients. By accumulating past gradients, momentum helps smooth out oscillations in the optimisation process, allowing the optimiser to persist in the same direction if gradients have been consistently pointing in that direction.

Another notable extension to FedAvg is FedAvg with Trimmed Mean which introduces robustness to outliers and improves the convergence of federated learning. This approach extends the basic FedAvg algorithm by incorporating the concept of trimmed mean aggregation, where extreme values are removed before computing the average. By leveraging the trimmed mean, FedAvg with Trimmed Mean mitigates the influence of outliers on the global model updates, resulting in more stable and robust convergence properties.

On the other hand, the heterogeneity among client devices in terms of data distributions, computational resource constraints and communication capabilities present significant challenges. In response, Reddi et al. (2021) have proposed Adaptive Federated Optimisation (AFO) algorithm to address these challenges and augment the efficiency of federated learning systems. By integrating adaptive client selection, dynamic learning rate adjustments, personalised model updates and privacy-preserving mechanisms under resource-constrained settings, AFO enhances the scalability, efficiency, and effectiveness of federated learning systems.

Although, federated learning has progressed in various domains since its emergence, research on infotainment systems within the Internet-of-Vehicles domain, particularly in the context of federated hybrid recommendation systems, remains notably very limited.

CHAPTER 4

4. RESEARCH METHOD

This section methodically presents our centralised and federated hybrid system architectures while outlining the governing mechanisms.

In our study, we implement a centralised hybrid recommendation system that naturally combines collaborative filtering and content-based filtering techniques within a unified framework. This system leverages the strengths of both paradigms, employing Matrix Factorisation (MF) to capture latent user-item interactions and a Multi-Layer Perceptron (MLP) to identify non-linear patterns and complex relationships. Additionally, we introduce a Long Short-Term Memory (LSTM) layer to process temporal features, enhancing our centralised recommendation system with the ability to understand evolving user preferences over time.

Under the same experimental settings, our federated recommendation system extends beyond our centralised recommendation system. Ultimately, our federated recommendation system shares all features with its centralised counterpart while accommodating the advantages of federated setup as well.

4.1 SYSTEM OVERVIEW

4.1.1 Mutual Core Features

Collaborative Filtering: The collaborative filtering aspect of our hybrid recommendation systems is realized through Matrix Factorisation (MF) components. These components embed users and items into low-dimensional vectors, capturing latent factors that influence user-item interactions. Specifically, the component comprises:

- Typically, as in our study, the interaction between users and items is

computed through the dot product of user and item embeddings as shown in Figure 4.1, providing a measure of similarity in the collaborative filtering process. Additionally, it's worth noting that the learned embeddings condense high-dimensional user and item information into lower-dimensional representations, facilitating efficient computation and revealing meaningful relationships between users and items in the hidden space.

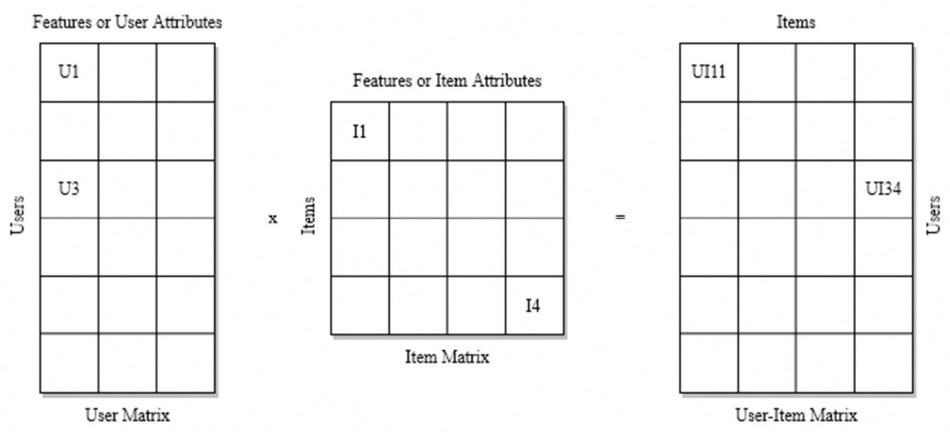

Figure 4.1 Matrix Factorisation Interaction

Content-Based Filtering: The content-based filtering aspect of our hybrid recommendation systems is implemented by integrating temporal features and leveraging a Multi-Layer Perceptron (MLP) into the recommendation logic. To capture temporal patterns in user-item interactions, we include a Long Short-Term Memory (LSTM) layer. Additionally, the MLP processes user and item embeddings along with the LSTM temporal representation, allowing the model to capture content-based information. Specifically, the component comprises:

- The LSTM is a dedicated neural network architecture prominent for its effectiveness in handling sequential data. Unlike traditional neural networks, fundamentally, LSTMs are designed with memory cells and gating mechanisms, enabling them to capture and remember long-term dependencies in sequences. This unique structure makes LSTMs

particularly well-suited for assignments including temporal patterns, such as time series analysis and sequential data processing. In our study, an additional LSTM layer takes the sequence of normalised timestamps as input, generating a temporal representation that reflects the evolving preferences of users over time.

- As shown in Figure 4.2, a Multi-Layer Perceptron (MLP) is a neural network comprising multiple layers of nodes (neurons) and is often used for various machine learning tasks, including recommendation systems. Particularly, the configuration of input and output layers in neural networks varies based on the specific task and architecture. In single input, single output (SISO) setups like regression, multiple input nodes represent features, while one output node predicts a continuous value. Conversely, single input, multiple output (SIMO) networks produce multiple outputs from a single input, beneficial for multitask learning. Multiple inputs, single output (MISO) networks, such as in reinforcement learning, process various inputs to generate a single output like a reward. Finally, multiple inputs, multiple outputs (MIMO) architectures handle complex tasks like sequence-to-sequence learning, where sequences of data produce sequences of outputs. These configurations cater to diverse neural network requirements and application domains. In the context of collaborative filtering, an MLP can be employed as a component for learning complex patterns and representations from user-item interactions. Specifically, in our study, user and item embeddings are processed through the MLP, along with the temporal representation obtained from the Long Short-Term Memory (LSTM) layer. The resulting concatenated tensor serves as input for subsequent layers, enabling the model to learn complex patterns that may be challenging to capture through separate components.

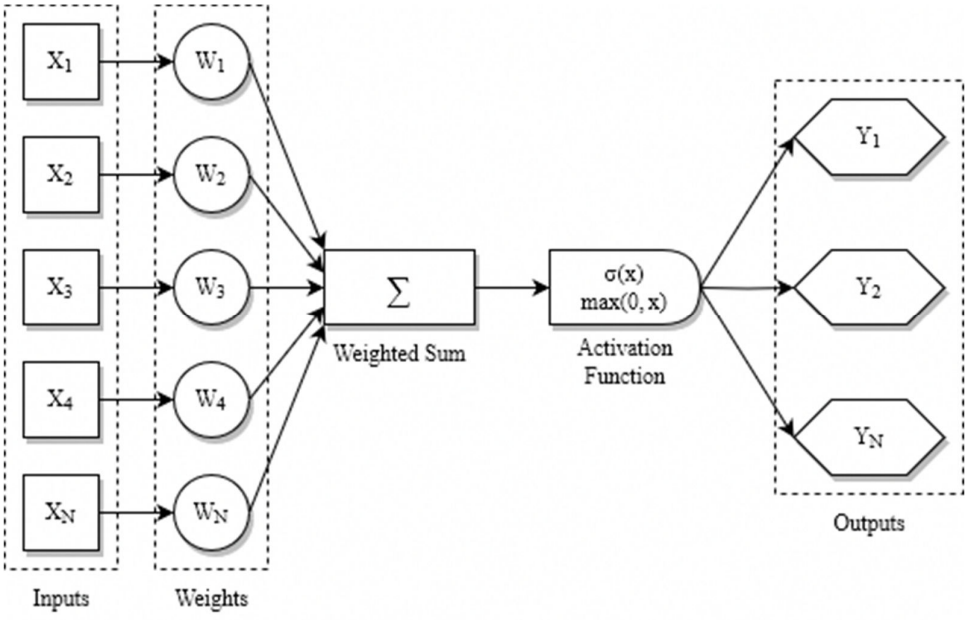

Figure 4.2 Multi-Layer Perceptron

User-Item Embeddings: User embeddings are compact numerical representations of user characteristics derived from patterns and interactions in the data, often used in recommendation systems to capture user preferences and behaviours for personalised content recommendations. In our study, we transform user identifiers into dense vectors utilising a user embedding layer, capturing hidden user features.

Similarly, item embeddings are compact vector representations in a recommendation system that capture latent features and relationships of items, facilitating efficient similarity calculations and personalised content recommendations. In our study, we transform item identifiers into dense vectors utilising an item embedding layer, capturing hidden item features.

Integrating Collaborative Filtering and Content-Based Filtering Components: As metaphorically shown in Figure 4.3, the final recommendations in a hybrid recommendation system are generated with the combination of outputs from both collaborative filtering (Matrix Factorisation) and content-based filtering (Multi-Layer Perceptron).

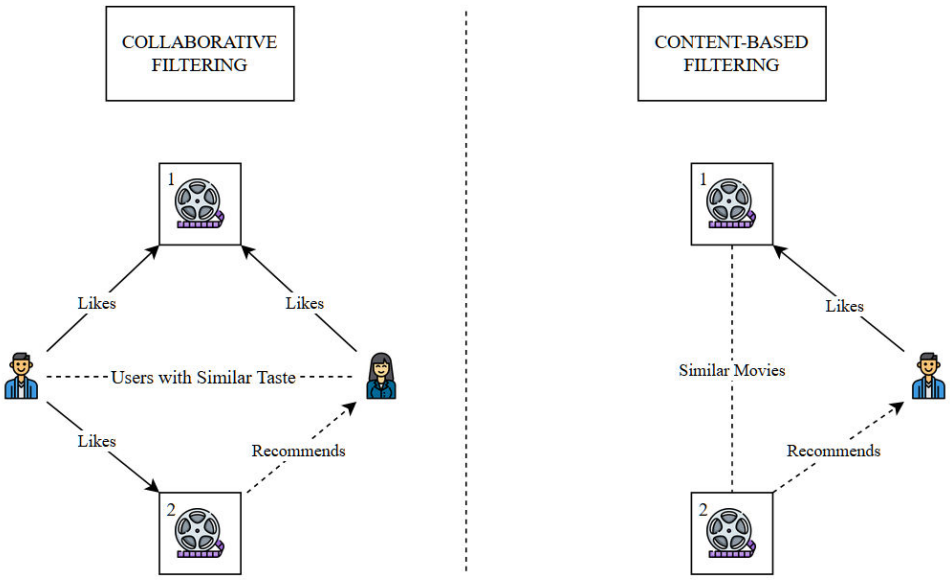

Figure 4.3 Comparing Collaborative Filtering with Content-Based Filtering

4.1.2 Mutual Quality Features

Differential Privacy: In our study, we prioritise privacy by employing differential privacy. In that respect, we generate meaningful Laplace noise and then add it to the gradients of each model parameter.

Hyperparameter Optimisation: In our study, we prioritise the enhancement of system performance by implementing hyperparameter optimisation via Optuna framework. Leveraging its efficient and automated search algorithms, we systematically explore and refine the hyperparameter space. Optuna stands out as a hyperparameter optimisation framework that employs a more intelligent approach compared to other techniques. Instead of exhaustively trying all possible combinations of hyperparameter values, Optuna utilises a sequential optimisation strategy to efficiently search the hyperparameter space. This refined approach allows us to fine-tune model hyperparameters, significantly enhancing the overall performance and effectiveness of the recommendation system.

4.1.3 Adaptive Moment Estimation (Adam) Optimiser

Adam (Adaptive Moment Estimation) is an optimisation algorithm frequently employed in training deep neural networks. It belongs to the family of stochastic gradient descent (SGD) optimisation algorithms and is known for its efficiency and effectiveness in practice. Adam maintains adaptive learning rates for each parameter by combining ideas from two other popular optimisation algorithms: RMSprop and Momentum.

In our study, we utilised the Adam optimiser since it proves to be particularly beneficial for a hybrid movie recommendation system due to its adaptive learning rates and efficient handling of sparse gradients. In the context of recommendation systems, especially those that leverage a combination of collaborative filtering and content-based filtering, Adam's characteristics contribute to the overall effectiveness of the optimisation process. Some key reasons why Adam is well-suited for a hybrid movie recommendation system are: Adaptive Learning Rates, Efficient Handling of Sparse Gradients, Seamless Integration of Collaborative and Content-Based Components, Fast Convergence and Robustness to Hyperparameter Choices.

The update rule of Adam involves computing moving averages of the gradient and squared gradient, which are then used to adaptively adjust the learning rates for each parameter:

The adaptive nature and momentum-like behaviour make Adam well-suited for optimising neural networks in a variety of scenarios, including hybrid movie recommendation systems.

4.1.4 L2 Regularisation

L2 regularisation is implemented by incorporating an additional component into the loss function, aiming to penalise excessive weights within a neural network. This component involves a scaled summation of the squares of all network weights. The objective of L2 regularisation is to curb model complexity by discouraging the proliferation of large weights, thus mitigating

the risk of overfitting.

As can be seen in our study, in the context of a neural network's optimisation process, the L2 regularisation term is added to MSE loss function.

4.1.5 Mean Squared Error (MSE) Loss Function

Mean Squared Error (MSE) is a common loss function used in regression tasks, including those related to recommendation systems. The MSE loss measures the average squared difference between the predicted values and the actual (ground truth) values. Mathematically, it is expressed as:

The MSE loss function penalises greater mistakes considerably due to the squaring operation, making it particularly subtle to outliers. In the context of a hybrid recommendation system, MSE is often used to quantify the dissimilarity between predicted ratings and actual user ratings for items.

4.2 SYSTEM ARCHITECTURE

This sub-section provides a deep-dive explanation of our hybrid recommendation systems by delving into the detailed system architecture across both centralised and federated systems. These systems comprise a set of interconnected components. Below, we outline all key elements that constitute our hybrid recommendation systems.

4.2.1 Data Pre-Processing

The raw Movielens-1M (Harper et al., 2015) benchmark dataset includes files pertaining to both movie ratings and tags. Within these files are 1,000,209 anonymous ratings covering approximately 3,900 movies. These ratings were provided by 6,040 MovieLens users who became members in the year 2000. The movie tags within the dataset provide additional details about the movies such as movie genres and movie titles. The data pre-processing phase involves merging these two files and shuffling the merged dataset to have a homogenous and fair distribution for each environment. For centralised setup, we only split

our dataset into three parts as training (60%), testing (20%) and evaluation (20%) subsets and directly use each chunk. For federated learning setup, we first split the dataset into three parts as training (60%), testing (20%) and evaluation (20%) subsets and after that distribute each chunk to the clients because it is required for federated learning that each client must store and process its own local data. All data pre-processing steps are demonstrated as below in Figure 4.4.

```
# Step 1: Load ratings data from 'ml-1m/ratings.dat'
# Step 2: Normalise timestamp in 'ml-1m/ratings.dat'
# Step 3: Load movies data from 'ml-1m/movies.dat'
# Step 4: Merge ratings and movies data
# Step 5: Shuffle merged data
# Step 6: Split data into training, validation, and testing sets
# Step 7: Distribute the split data among workers (If federated)
```

Figure 4.4 Data Pre-processing

4.2.2 Hyperparameters

Hyperparameters play an essential role in determining the performance and behaviour of our proposed centralised and federated hybrid recommendation systems. Some hyperparameters are fine-tuned during the training process to optimise the system performance, while others are fine-tuned during phases specific to differential privacy or hyperparameter optimisation, if applicable. As presented in Appendix-B, we divide the hyperparameters into two categories: those specific to the centralised hybrid recommendation system and those specific to the federated hybrid recommendation system. These hyperparameters have been carefully selected and fine-tuned through extensive experimentations to optimise the performance, robustness, convergence, and privacy preservation capabilities of our hybrid recommendation systems.

Due to the replication of both centralised and federated systems within our framework, it is notable that some hyperparameters overlap across both paradigms. This overlap occurs as several key hyperparameters, such as learning rate, batch size, and number of epochs, are fundamental to the underlying machine learning algorithms and model architectures employed in both

centralised and federated settings. Consequently, the hyperparameters that govern the core aspects of model training and optimisation exhibit consistency regardless of the centralised or federated nature of the recommendation system.

4.2.2.1 Mutual Hyperparameters

Learning Rate (η): The learning rate controls the step size during the optimisation process. A higher learning rate may lead to faster convergence but jeopardies exceeding the optimum outcome, while a lower learning rate may result in slower convergence but potentially more precise optimisation. We set the learning rate to a value of 1e-3 based on empirical experimentation and prior literature.

Regularisation Parameter (λ): Regularisation is employed to mitigate overfitting by penalising overly complex models. The regularisation parameter, denoted as λ, governs the strength of this penalty. Higher values of λ impose greater penalties on complex models, encouraging simplicity and improving generalisation. Conversely, lower values of λ permit more complexity, potentially leading to overfitting. We carefully select the regularisation parameter to balance model complexity and generalisation performance.

Weight decay, often referred to as L2 regularisation, is a common technique utilised to control model complexity and prevent overfitting. It achieves this by penalising large weights in the model. We employ weight decay with an hyperparameter value of 1e-3 to regulate model complexity and enhance generalisation capabilities.

Hidden Dimension (d): Hidden dimension refers to the size of the hidden layers within the neural network architecture. In the context of hybrid recommendation systems, the hidden dimension determines the size of the latent space in which items and users are represented. A higher hidden dimension allows for more expressive representations but increases the computational complexity of the model. Considering factors such as the size of the dataset and computational constraints, we set hidden dimension to 50 and LSTM hidden dimension to 32. Both hidden dimensions are crucial in shaping the model's

capacity to capture complex patterns within the dataset.

Batch Size: The batch size hyperparameter defines the number of samples processed by the neural network model in a single forward and backward pass during each iteration of training.

Setting the batch size is a critical decision in training neural network models. It directly impacts the speed, quality, training dynamics, and overall performance. A larger batch size concurrently processes more samples per iteration, potentially leading to faster convergence and efficient utilisation of computational resources. However, larger batch sizes may also require more memory, limiting the size of models that can be trained on available hardware. Conversely, a smaller batch size processes fewer samples per iteration, which may lead to slower convergence but may result in more stable training process and better generalisation performance. Smaller batch sizes are particularly useful when working with limited memory resources or when training on datasets with varied sample characteristics.

We conducted initial experiments using batch sizes of 16, 32 and 64 and determined that a batch size of 32 exhibits slower convergence but offers more stable training. In that respect, we set the batch size to 16 to find the optimal value that balances computational efficiency with model effectiveness, considering the specific requirements and constraints of the learning task and available hardware resources.

Epochs: Depending on the specific characteristics of the dataset, the complexity of the model architecture, and the computational resources available for model training, setting the number of epochs hyperparameter is crucial aspect of neural network architecture as it directly influences the convergence speed and general performance of the model. Each epoch consists of multiple iterations, where the model updates its parameters using batches of training data. Insufficient epochs may lead to underfitting, where the model struggles to grasp intricate patterns within the data, whereas excessive epochs may cause overfitting, where the model excessively memorises the training data and performs inadequately on new data. We set the number of epochs

hyperparameter to 25 to balance model convergence and computational resources. Yet, considering the implemented early stopping mechanism in our neural network architecture, the number of epochs holds less significance. Regardless of the number of epochs, training will halt prematurely if consecutive training rounds fail to demonstrate improvement in validation loss. With that additional measure in place, our aim is to enable the model to sufficiently learn from the training data without underfitting or overfitting, while also mitigating the potential negative effect of one of the important hyperparameter in our hybrid recommendation system, number of epochs.

Dropout Rate: Dropout is another widely adopted regularisation technique in neural network architectures to prevent overfitting. During training, dropout randomly deactivates a fraction of input units with a specified probability, effectively excluding them from the network for that iteration. This introduces noise, encouraging the network to learn more robust features by preventing reliance on specific neurons and complex co-adaptations. Consequently, dropout enhances the model's generalisation capabilities, reducing the risk of overfitting to the training data.

However, the dropout hyperparameter requires careful tuning because while higher dropout rates can effectively prevent overfitting, excessively high dropout rates may hinder the learning ability of the network. Moreover, extremely lower dropout rates may be impractical. Therefore, selecting an appropriate dropout rate involves experimentation and validation to achieve optimal model performance.

In our model architecture, we conducted initial experiments using dropout rates of 0.3 and 0.5. We concluded that excessive noise harms the network. In that respect, we incorporate dropout regularisation with a dropout rate of 0.3. During each training iteration, 30% of the units in the input layer of the specified dropout layers are randomly set to zero, with the aim of not harming disproportionately the prior knowledge in the network.

Number of Groups: In the context of neural network architecture, the number of groups hyperparameter pertains to group normalisation denoted as

'nn.GroupNorm', a technique used to normalise the activations of a neural network layer.

Group normalisation divides the channels or features of the input tensor into smaller groups where each group consists of a subset of channels. Statistics (e.g., mean and variance) are then computed independently for each group which allows the model to capture features more effectively across different channels and encourages stable training dynamics. In that respect, the number of groups hyperparameter determines how many groups the channels will be divided into for normalisation. A higher number of groups results in finer granularity of normalisation, potentially enhancing the model's ability to capture complex patterns in the data. However, too many groups may increase computational overhead. Group normalisation is particularly useful in scenarios where batch sizes are small or when the data distribution significantly varies across batches.

In our model architecture, we utilise group normalisation with a specified number of groups set to 32, carefully considering the balance between computational efficiency and the model's capacity to learn diverse features.

4.2.2.2 Unique Hyperparameters – Federated Learning

Number of Clients (K): The number of clients hyperparameter has a profound impact on the federated learning architecture, influencing various aspects of system design and overall robustness. With an increased number of clients, the federated system becomes more distributed, encompassing a larger network of devices. This distributed nature enhances data privacy and security by minimising data centralisation and reducing the risk of data exposure. However, accommodating a larger number of clients presents several systematic challenges. Communication overhead becomes more significant as model updates need to be aggregated from a larger pool of devices. Efficient communication protocols, resource utilisation algorithms and model aggregation mechanisms become essential to ensure coordinated and optimised synchronisation across the federated network.

During our initial experiments, we noted that having a small number of

clients, each with a little portion of the partitioned dataset, impeded the effectiveness of the network. However, it expedited the learning process by decreasing the computational overhead and shortening the time required. On the contrary, having a large number of clients, each with a substantial portion of the partitioned dataset, expedited the effectiveness of the network. However, it impeded the learning process by increasing the computational overhead and extending the time required. Consequently, in our federated learning architecture, we set the number of clients hyperparameter to 25 by carefully balancing the benefits of data diversity and privacy with the challenges of communication overhead and computational resource management.

Aggregation Method: In federated learning architectures, aggregation method does not pertain to a specific hyperparameter but rather a governing factor how client updates are aggregated to form a global model update. Various aggregation algorithms exist, each offering unique advantages depending on the specific requirements of the federated learning scenario. The choice of aggregation method impacts factors such as model convergence and communication overhead. Below, we outline aggregation methods selected for implementation and their distinctive characteristics:

- FedAvg (Federated Averaging): FedAvg calculates the average of the client model updates to produce the global model update. It is widely used due to its simplicity and effectiveness in aggregating client contributions, resulting in a consensus global model.
- FedMedian (Federated Median): FedMedian computes the median of the client model updates instead of the average. This method is robust to outliers and can enhance model robustness, particularly in scenarios with non-IID data distributions.
- FedProx (Federated Proximal): FedProx integrates a proximal term into the aggregation process, imposing a penalty on deviations from a central model. It enables fine-tuning of the global model while mitigating client heterogeneity.
- FedAvg with Trimmed Mean: FedAvg with trimmed mean combines the

benefits of FedAvg with robust statistical techniques like trimmed mean, which excludes extreme values from the aggregation process. This approach may improve model convergence and stability.

- FedAvg Momentum: FedAvg Momentum extends FedAvg by incorporating momentum into the aggregation process. Momentum accumulates past gradients to stabilise updates, potentially accelerating convergence and enhancing model performance.
- Adaptive Fed Optimisation (AFO): Adaptive Fed Optimisation dynamically adjusts aggregation parameters based on the characteristics of client updates and network conditions. It adapts the aggregation process to optimise model convergence and communication efficiency.

In our federated architecture, we evaluate and compare each of these aggregation methods individually to assess their performance and suitability based on empirical results for our hybrid recommendation system.

4.2.2.3 Mutual Hyperparameters for Early Stopping Mechanism

Patience: In our neural network architecture, patience hyperparameter refers to the number of epochs the training process will wait for an improvement in validation loss before getting terminated. When training a neural network model, it's common for the validation loss to fluctuate due to factors such as noise in the data or the stochastic nature of the optimisation process. We allow the model some flexibility to navigate these fluctuations by setting a patience hyperparameter.

Choosing an appropriate patience value requires balancing between allowing sufficient time for the model to converge and preventing it from training for too long, potentially overfitting the data. Conversely, terminating training too early due to a low patience value may result in underfitting, where the model fails to capture the complexity of the data and performs poorly on both training and validation sets. Therefore, it's crucial to carefully fine-tune the patience hyperparameter to strike a balance between preventing overfitting and avoiding underfitting. In that respect, we set the patience hyperparameter to 5 to

facilitate timely convergence. At the end of each training epoch, the system calculates the average validation loss over the last 10 epochs and compares it with the best validation loss obtained so far. If the last average validation loss consecutively increases for more than specified patience hyperparameter, training is halted, assuming that the model has reached an optimal point.

4.2.2.4 Mutual Hyperparameters for Differential Privacy

Privacy Budget (ε): Differential privacy is a framework designed to enhance the user privacy. It introduces controlled randomness into the learning process to protect information while still allowing useful insights to be extracted. In that respect, the privacy budget is one of the fundamental hyperparameters in differential privacy that quantifies the level of privacy protection.

In our neural network architecture, alongside with noise multiplier hyperparameter, it controls the amount of Laplace noise added to the gradients of the model parameters during the training process. A smaller privacy budget corresponds to stronger privacy guarantees, as it limits the amount of information that can be inferred about individual data points from the model's output. However, reducing the privacy budget too much may lead to a significant degradation in the utility or accuracy of the model's predictions. Therefore, we set the privacy budget to 1 | 5 to achieve a meaningful compromise between privacy requirements and model quality.

Noise Multiplier: The noise multiplier is another fundamental hyperparameter in determining the amount of noise added to the model parameters to ensure differential privacy. It directly influences the scale of the Laplace noise injected into the gradients during the training process. A higher noise multiplier results in more substantial noise addition, thereby enhancing privacy guarantees but potentially impacting model accuracy.

In our neural network architecture, we set the noise multiplier hyperparameter to a value of 0.1 | 0.01. This choice aims to mitigate privacy risks while maintaining a balance between privacy protection and model effectiveness.

4.2.2.5 Mutual Hyperparameters for Hyperparameter Optimisation

Search Space: The search space refers to the range of possible values assigned to key model parameters that are explored during the hyperparameter optimisation process for improved system performance.

In our study, the search space encompasses strategically chosen hyperparameters crucial to execute an effective hybrid recommendation system. As shown in Table 4.1, we specifically explore intervals for hyperparameters such as hidden dimension, LSTM hidden dimension, learning rate, and weight decay. For instance, the hidden dimension determines the size of the latent space in which features are represented, while the LSTM hidden dimension controls the complexity of the temporal features captured by the LSTM layer, the learning rate governs the step size of parameter updates during model optimisation, and the weight decay parameter regulates the degree of regularisation applied to prevent overfitting. Once the optimisation process is complete, the best-found hyperparameters are then used to ensure optimal performance and effectiveness.

Table 4.1 Hyperparameter Optimisation Search Space

Hyperparameter	**Search Space**	**Justification for Importance**
hidden_dim	20 - 100	Determines the size of the latent space for feature representation.
lstm_hidden_dim	16 - 64	Controls the complexity of temporal features captured by the LSTM layer.
learning_rate	1E-4 - 1E-2	Governs the step size of parameter updates during model optimisation.
weight_decay	1E-6 - 1E-4	Regulates the degree of regularisation to prevent overfitting.

We also would like to highlight that it is not advisable to optimise certain hyperparameters, particularly those related to differential privacy.

Hyperparameters, such as privacy budget and noise multiplier, are typically determined based on specific privacy requirements and considerations, rather than through conventional optimisation techniques.

Sampling Strategy: Hyperparameter optimisation process involves efficiently exploring the hyperparameter search space to discover high-performing configurations. In that regard, various methods, such as grid search, random search, or Bayesian optimisation, can be employed.

In our hyperparameter optimisation process, we utilise the Tree-structured Parzen Estimator (TPE) algorithm as sampling strategy, implemented through Optuna TPESampler class. TPE is a sequential model-based optimisation algorithm that effectively balances exploration and exploitation of the hyperparameter search space. It leverages the observed performance of previous hyperparameter configurations to guide the search towards promising regions, gradually converging towards optimal configurations. During the optimisation, the direction hyperparameter is explicitly set to minimize. We aim to find the hyperparameter configurations that lead to the lowest possible value of the objective of progressively minimising our validation loss, thereby optimising the overall performance.

Number of Trials: The number of trials hyperparameter corresponds to the number of iterations executed during hyperparameter optimisation. We set the number of trials hyperparameter to 5 to balance computational resources while efficiently exploring the definite hyperparameter space.

4.2.3 Hybrid Recommendation Neural Network Model

The neural network model within our hybrid recommendation system is the result of a synergy among various components. In Figure 4.5, we outline all key elements that constitute our hybrid recommendation neural network model.

```
# Step 0: Define hyperparameters
# Step 1: Define Kaiming (HE) initialisation
# Step 2: Define hybrid recommendation neural network model
(via 'nn.Module')
      # Step 2.1: Initialise matrix factorisation components
(via 'nn.Embedding')
      # Step 2.2: Initialise multi-layer perceptron components
(via 'nn.Embedding')
      # Step 2.3: Initialise LSTM layer for temporal features
(via 'nn.LSTM')
      # Step 2.4: Initialise Kaiming (HE) initialisation to
embedding layers (via 'init.kaiming_uniform_')
      # Step 2.5: Initialise a container for the neural network
model (via 'nn.Sequential')
# Step 3: Define forward pass computation in neural network
model
      # Step 3.1: Compute matrix factorisation embeddings in
forward pass
      # Step 3.2: Compute multi-layer perceptron embeddings in
forward pass
      # Step 3.3: Utilise LSTM for temporal representation in
forward pass
      # Step 3.4: Combine MF and MLP predictions for final
output
# Step 4: Instantiate neural network model
      # Step 4.0: Switch to CUDA-compatible GPU device
      # Step 4.1: Create & move neural network model to CUDA-
compatible GPU device
      # Step 4.2: Apply Kaiming (HE) initialisation to neural
network model
# Step 5: Attach L2 regularisation into the model parameters
using Adam optimiser (via 'optim.Adam')
# Step 6: Define loss function (via 'nn.MSELoss')
# Step 7: Distribute partitioned datasets to clients (If
federated)
# Step 8: Perform hyperparameter optimisation (If applicable)
      # Step 8.1: Sample hyperparameters for the trial
      # Step 8.2: Train and validate the neural network model
for the trial
# Step 9: Push initial global model from central server to all
participated clients (If federated)
# Step 10: Train, validate and test the neural network model
      # Step 10.1: Apply differential privacy during model
training (If applicable)
# Step 11: Aggregate and pull all local models from all
participated clients to central server (If federated)
# Step 12: Push up-to-date global model from central server
to all participated clients (If federated)
# Step 13: Aggregate and analyse metrics
```

Figure 4.5 Hybrid Recommendation Neural Network Model

4.2.3.1 User-Item Embeddings (Matrix Factorisation)

The user-item embeddings are utilised in the matrix factorisation component, where the interaction between users and items is computed as the element-wise dot product of user and item embeddings.

In our study, as shown in Figure 4.5, we define MF user-item embeddings as nn.Embedding within the NeuMF class where self.user_embedding_mf and self.item_embedding_mf represent the embeddings for users and items.

4.2.3.2 User-Item Embeddings (Multi-Layer Perceptron)

It is a common and effective practice to use the user-item embeddings for both collaborative filtering and content-based filtering components in a hybrid recommendation model. The collective embeddings allow the model to capture the characteristic relationships and patterns in user-item interactions, providing a unified representation. Afterwards, the final prediction generated by the neural network model is typically used to activate both approaches to enhance the overall recommendation accuracy in a more performative manner.

In our study, as shown in Figure 4.5, we define MLP user-item embeddings as nn.Embedding within the NeuMF class where self.user_embedding_mlp and self.item_embedding_mlp represent the embeddings for users and items, respectively. The output of the LSTM, specifically the representation from the last time step (time_embed), is concatenated with the user and item embeddings along the last dimension. The concatenated tensor serves as the input to the MLP, and the result is obtained through the fully connected (dense) layers (self.fc_layers). The final output of the MLP is stored in mlp_output. The predictions from the matrix factorisation (mf_interaction) and the MLP (mlp_output) are combined to form the final prediction (prediction).

4.2.3.3 LSTM (Long-Short Term Memory)

The LSTM is employed to capture temporal dependencies in the dataset, enabling the model to learn and adapt to evolving user preferences over time.

The final temporal representation is used alongside user and item embeddings in the later stages of the model.

In our study, we again introduce the LSTM layer within the NeuMF class using nn.LSTM. As shown in Figure 4.5, the layer takes time sequences as input and produces temporal representations.

4.2.3.4 Kaiming (He) Initialisation

Weight initialisation methods play a crucial role in the training step of a neural network model, helping to set neutral weights before the optimisation process begins. For instance, Kaiming initialisation is especially beneficial with activation functions such as ReLU, preventing issues like vanishing or exploding gradients because such problems may hinder the convergence and effectiveness of the neural network model.

As shown in Figure 4.5 and Figure 4.6, we apply Kaiming initialisation explicitly to the embedding layers via weights_init_kaiming function, ensuring appropriate initialisation for ReLU activation. Moreover, during the initialisation of the neural network model weights, we check m.bias if linear layers (e.g. nn.Linear) have bias terms. If m.bias is not None, meaning the module has a bias term, m.bias.data.zero_() sets all the elements in the bias tensor to zero.

```
def weights_init_kaiming(m):
            if isinstance(m, (nn.Linear)):
                      init.kaiming_uniform_(m.weight.data)
                      if m.bias is not None:
                               m.bias.data.zero_()
```

Figure 4.6 Defining Kaiming (He) Initialisation

4.2.3.5 Fully Connected (Dense) Layers

Fully connected (dense) layers in a neural network model serve as a crucial component for capturing complex patterns in input data. In these layers, each neuron is connected to every neuron in the preceding and succeeding layers, allowing for comprehensive information exchange in between. The weights and

biases associated with these connections are learned during model training step, enabling the network to transform input features into a task-specific representation.

As shown in Figure 4.5 and Figure 4.7, we define the fully connected (dense) layers (fc_layers) using nn.Linear in the NeuMF class within the self.fc_layers sequential block. The model architecture includes fully connected layers with ReLU activation, dropout, and group normalisation. In the context of a neural network model, self.fc_layers would typically be a set of fully connected (dense) layers defined using PyTorch's nn.Linear. It is common to use the sequential container to organize these layers in a sequential manner, where the output of one layer becomes the input to the next one. These layers introduce non-linearity to the model through ReLU activation. Additionally, the sequential structure of fully connected (dense) layers captures complex relationships and hierarchical features in the data, facilitating the learning during backpropagation.

```
self.fc_layers = nn.Sequential(
      nn.Linear(hidden_dim * 2 + lstm_hidden_dim, 512),
      nn.ReLU(),
      nn.Dropout(0.3),
      nn.GroupNorm(num_groups, 512),
      nn.Linear(512, 256),
      nn.ReLU(),
      nn.Dropout(0.3),
      nn.GroupNorm(num_groups, 256),
      nn.Linear(256, 1)
)
```

Figure 4.7 Defining Fully Connected (Dense) Layers

It is noteworthy that, in our fully connected layer setup, initially, we employed BatchNorm1d for normalisation. However, after extensive testing and evaluation, we made a strategic decision to transition to the utilisation of GroupNorm instead of BatchNorm1d. This decision was primarily driven during our experimentation which revealed that GroupNorm offered more effective feature normalisation and regularisation, particularly in scenarios where the input data exhibited non-uniform distributions and complex interdependencies.

Table 4.2 Comparison between Group Normalisation and Batch Normalisation

Key Characteristic	GroupNorm	BatchNorm1d
Grouping	Channels are divided into groups, and normalisation is performed independently within each group.	Normalisation is performed across the entire batch dimension.
Scalability	GroupNorm tends to be more scalable compared to Batch Normalisation, particularly in scenarios with small batch sizes or when the batch size varies.	BatchNorm1d may suffer from reduced effectiveness with small batch sizes or varying batch sizes due to the reliance on statistics computed across the entire batch.
Robustness	GroupNorm is less sensitive to batch size variations and can provide stable performance across different batch sizes.	BatchNorm1d may exhibit sensitivity to batch size variations, which could affect its performance, especially with small or varying batch sizes.

Additionally, GroupNorm demonstrated superior scalability and efficiency with better loss values, enabling more streamlined training processes and facilitating the optimisation of computational resources. Table 4.2 provides a comparative analysis in that respect.

4.2.3.6 Forward Pass Computation

The forward pass computation in a neural network model, specifically in architectures involving fully connected dense layers, involves the sequential flow of input data through layers of interconnected neurons. Starting with the input layer, each neuron's input is multiplied by weights, and an activation function is applied elementwise to introduce non-linearity into the network. This process is repeated through hidden (dense) layers until the output layer generates predictions. The output is then compared to the actual values using a defined loss function. The forward pass is a crucial step in training a neural network, with subsequent adjustments to weights performed during the backpropagation phase

to minimise the computed loss and improve the model's accuracy.

In our study, as shown in Figure 4.5, we calculate the recommendation prediction by aggregating user-item embeddings along with LSTM layer for temporal features within the forward method. The embeddings and temporal features are concatenated and fed into an MLP, and the final prediction is obtained by combining the MF interaction and MLP output.

4.2.3.7 Backward Pass Computation

Backpropagation is exclusively conducted during the training phase of a neural network. This procedure entails calculating the gradients of the loss function relative to the model's parameters and then adjusting those parameters to minimise the loss. Through iterative adjustments of parameters based on observed errors in the training data, the neural network learns to enhance the accuracy of its predictions. Backpropagation serves as a pivotal component in optimisation algorithms like Adam, leveraging these gradients to update parameters in a manner that minimises the loss function. On the other hand, during evaluation or testing phases, backpropagation is not performed. Instead, the parameters of the trained model remain unchanged, and only the forward pass is executed instead to calculate the ability of the model to generalise to unseen data based on the learned parameters. This separation ensures consistent and reliable predictions.

In our study, we implement backward pass computation (backpropagation) by computing gradients of the loss function with respect to the model parameters using ‘loss.backward ()’, allowing the Adam optimisation algorithm to update the model's parameters during training phase.

4.2.3.8 L2 Regularisation

L2 regularisation penalises large weights, preventing overfitting by discouraging excessively complex representations. It enhances the model ability to generalise to unseen data.

As shown in Figure 4.5, we implicitly incorporate L2 regularisation during

the optimiser setup. The weight_decay parameter in optim.Adam introduces L2 regularisation to the model's parameters during training.

```
optimiser = optim.Adam(model.parameters(), lr=learning_rate,
weight_decay=weight_decay)
```

Figure 4.8 Enhancing Adam Optimiser with L2 Regularisation

As shown in Figure 4.8, model.parameters () provides the weights of the model, and weight_decay controls the strength of L2 regularisation. Adjusting the weight_decay parameter allows us to control the regularisation strength during training.

4.2.3.9 Adam Optimiser

The Adam optimiser is employed for efficient training of the hybrid recommendation neural network model. The combination of adaptive learning rates, momentum-like behaviour, and L2 regularisation makes Adam well-suited for training a hybrid recommendation neural network.

As shown in Figure 4.5 and Figure 4.8, model.parameters supplies the weights of the neural network to the optimiser. The learning rate (lr) controls the step size during optimisation, determining the magnitude of parameter updates. Additionally, the weight_decay parameter introduces L2 regularisation to prevent overfitting by penalising large weights in the model.

4.2.3.10 MSE Loss Function

The essential objective is to minimise the MSE loss, resulting in a model that provides predictions that are as close as possible to the actual ratings.

As shown in Figure 4.5, we define the MSE loss function as criterion = nn.MSELoss(). Afterwards, during model training, validation, and testing steps, we utilise this loss function to measure the dissimilarity between the predicted values and the actual (ground truth) values. The optimisation process aims to minimise this loss, adjusting the hyperparameters to improve the accuracy.

4.1.3.11 Calculating Error

Evaluation of hybrid recommendation systems is fundamental in assessing their effectiveness in delivering personalised recommendations and comprehending the overall system performance. From that perspective, common metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE) offer valuable insights into different aspects of these systems. By leveraging these metrics, we can commentate on the strengths and weaknesses of our proposed hybrid recommendation systems and guide the refinement and optimisation areas to enhance user experience and develop further better systems.

The preferred loss function holds a key place in evaluating a hybrid recommendation system. It refers to the objective function used to quantify the difference between the predicted outputs of the model and the actual target values during training, validation, and testing phases. Thus, loss values indicate how well a model performs. Ideally, the main objective is to minimise the value of the loss function. Moreover, the choice of loss function depends on the specific task and the nature of the dataset. For regression tasks, such as our proposed hybrid movie recommendation system, where the goal is to predict continuous values such as user movie ratings, one common loss function is Mean Squared Error (MSE). MSE calculates the average squared difference between the predicted and actual values.

Additionally, Mean Absolute Error (MAE), provides further insights into model performance. Alongside MSE, MAE is another metric used in regression tasks to assess model accuracy. MAE calculates the average absolute difference between predicted and actual values, providing a straightforward interpretation and being less sensitive to outliers compared to MSE. As shown in Table 4.3, Both MSE and MAE offer valuable insights into the accuracy of models.

Table 4.3 Comparison between MAE and MSE

Criterion	Mean Absolute Error (MAE)	Mean Squared Error (MSE)
Formula	Average of absolute differences between predicted and actual values	Average of squared differences between predicted and actual values
Sensitivity	Less sensitive to outliers as it considers absolute differences	More sensitive to outliers as it squares the differences
Interpretation	Represents the average magnitude of errors in the predictions	Represents the average magnitude of squared errors in the predictions
Penalty	No penalty for larger errors beyond their absolute value	Larger errors are penalised more heavily due to squaring
Optimisation	Not differentiable at actual observed values and predicted values are equal, which may complicate optimisation	Differentiable at all points, making optimisation smoother

In our neural network model, for our centralised setup, we compute all metrics after each epoch. Similarly, for our federated setup, we aggregate and average all metrics from each client after each epoch before printing out the final metrics.

4.1.3.12 Centralised Training/Validation/Testing Phases

In our centralised hybrid recommendation system, the process starts after data pre-processing. It is worth noting that each step of the training, validation, and testing phases is conducted on separate subsets from raw dataset. Key hyperparameters such as the number of epochs, batch size, and hidden

dimensions are defined to configure the model architecture. The centralised hybrid neural network model with temporal features is then constructed using PyTorch, incorporating MF components, MLP components, and an LSTM layer for temporal features. Kaiming (HE) initialisation is applied to embedding layers to ensure effective learning. The Adam optimiser and MSE loss criterion are specified to train the model.

The training phase includes the initialisation of the model, optimiser, and other essential components. The model is set to training mode (via 'model.train()'). During each iteration, the following steps are repeated for multiple epochs until the performance of the model converges when the loss stops decreasing considerably:

- Forward Pass: During the forward pass, the model takes input data (e.g., user IDs, movie IDs, time, and rating) to make predictions. These predictions are compared with the actual ratings provided by users.
- Loss Computation: The MSE loss function computes the difference between the predicted ratings and actual ratings. This loss represents how well the model is performing compared to the ground truth.
- Backward Pass (Backpropagation): During the backward pass, the model computes the gradients of the loss function related to the model parameters using backpropagation. These gradients indicate how much each parameter needs to be adjusted to reduce the loss.
- Parameter Update: The Adam optimiser utilises the computed gradients to update the model parameters in the direction that minimises the loss.

After the training phase is completed, the model performance is evaluated on the validation set. The model is set to evaluation mode (via 'model.eval()'), and the validation loss is calculated within the same training loop. The reason PyTorch sets the model to evaluation mode is it disables operations like dropout and batch normalisation layers, ensuring consistent and deterministic behaviour during inference which is crucial for obtaining reliable and reproducible results. As shown in Figure 4.9, if the validation loss shows improvement compared to the best validation loss observed so far, the current model parameters are retained

as the best ones as part of early stopping mechanism. Otherwise, the model continues training with the goal of further improvement. The process repeats for the specified number of epochs or until early stopping criteria are met.

```
# Check if we have at least 10 validation losses for averaging
   if len(valid_losses) >= 10:
      print ('Checking last ten validation losses...')
      # Calculate the average of the last 10 validation losses
      avg_last_10_valid_loss = np.mean(valid_losses[-10:])
      # Check if the current average validation loss is greater
than the best valid loss
      if avg_last_10_valid_loss > best_valid_loss:
         epochs_without_improvement += 1
      else:
         best_valid_loss = avg_last_10_valid_loss
         epochs_without_improvement = 0
      # Check if the training should stop early
      if epochs_without_improvement >= patience:
         print (f'Early stopping after {epoch + 1} epochs without
improvement in average validation loss.')
         break
```

Figure 4.9 Early Stopping Mechanism in Centralised Model

Once training phase is completed, the model performance is evaluated on the testing set. The testing phase includes testing of model performance on unseen data to assess its generalisation capabilities. The model is again set to evaluation mode (via 'model.eval()'), and the testing loss is calculated. The testing loss serves as a crucial metric to gauge the model's effectiveness in making accurate predictions on new, unseen data. Subsequently, Mean Squared Error (MSE) and Mean Absolute Error (MAE) between the model predictions and the ground truth ratings is computed to assess the model's predictive accuracy as these metrics measure how well the model's predictions align with the actual ratings provided by users for items.

4.1.3.13 Federated Training/Validation/Testing Phases

In our federated hybrid recommendation system, all core components are replicated from the centralised system to enable fair comparison in between while preserving privacy. The process again begins after data pre-processing,

where the raw dataset is split into training, validation, and testing subsets. Each of these subsets is distributed among workers, with each worker assigned a unique user ID randomly. The distribution ensures that each chunk of the split subsets is stored on each client device, maintaining data locality and privacy. In that respect, data loaders are created for each worker to handle their own training, validation, and testing datasets independently. These data loaders facilitate data access and data manipulation during model training, validation, and testing. Additionally, a distribution function initialises the global model and global optimiser to each worker before starting the training phase to ensure that all workers start with the same initial model parameters before conducting local training.

Accordingly, after the initialisation of the global model, global optimiser, and other essential components, local training is performed by each worker on its own local data. Afterwards, local validation is conducted to assess model performance. As shown in Figure 4.10, an early stopping mechanism is again implemented by aggregating validation loss from all workers to prevent overfitting and promote model generalisation.

```
# For each epoch, calculate and store each worker's validation
loss for early stopping
   validation_losses.append(epoch_valid_loss.item())
   if len(validation_losses) >= 10:
      print ('Checking last ten validation losses...')
      # Calculate the average of the last 10 validation losses
      avg_last_10_valid_loss = np.mean(validation_losses[-10:])
      # Check if the current average validation loss is greater
than the best valid loss
      if avg_last_10_valid_loss > global_best_valid_loss:
         global_epochs_without_improvement += 1
      else:
         global_best_valid_loss = avg_last_10_valid_loss
         global_epochs_without_improvement = 0
      # Check if the training should stop early
      if global_epochs_without_improvement >= patience:
         print (f'Early stopping after {epoch + 1} epochs without
improvement in average validation loss.')
         break
```

Figure 4.10 Early Stopping Mechanism in Federated Model

After local validation, an aggregation algorithm is applied to update the global model using the gradients from each worker. Before the start of each new training epoch, the up-to-date global model is distributed to all workers to ensure consistency.

Upon completing local training and validation, local testing is executed, followed by the calculation of metrics same as those in the centralised setup. Local metrics are aggregated to monitor overall performance. Finally, global metrics are calculated by aggregating and averaging metrics from all workers across epochs.

4.2 DIFFERENTIAL PRIVACY

In our study, we tactically implement differential privacy to preserve the confidentiality of user data. The primary purpose is to prevent the model from memorising individual details, ensuring that the learning process remains privacy-preserving. During each training iteration, the implementation involves a series of steps. After computing the loss and performing the backward pass to calculate model's parameter gradients, we introduce privacy safeguards by adding Laplace-distributed noise to these gradients. This is achieved by iterating through the model parameters and augmenting their gradients with noise, effectively preventing the model from memorising individual data points. Subsequently, the model parameters are updated using the optimiser. This particular inclusion of differential privacy in the training loop ensures a robust approach to user data protection without compromising the model effectiveness.

```
# Add noise to gradients for differential privacy
for param in final_model.parameters ():
   if param.grad is not None:
      noise     =     torch.tensor     (np.random.laplace     (0,
noise_multiplier     /     epsilon,     param.grad.size     ()),
device=param.grad.device)
      param.grad += noise
```

Figure 4.11 Implementing Differential Privacy with Laplace Noise

As shown in Figure 4.11, the Laplace noise addition technique aids in achieving differential privacy by introducing a layer of controlled randomness to the model parameter updates during the training. However, despite the technical feasibility of extending differential privacy to validation and testing, a deliberate strategic choice has been made to conduct these phases with unperturbed data. Fundamentally, the primary objective of differential privacy is to prevent the model from memorising sensitive information during the training. Extending the implementation to the validation and testing phases could introduce unnecessary perturbations; potentially hindering a more accurate and reliable evaluation of the model's real-world performance under the influence of privacy-preserving measures.

4.3 HYPERPARAMETER OPTIMISATION

In our study, we strategically implement the Optuna hyperparameter optimisation framework with the key objective of refining and optimising the model effectiveness through a systematic exploration of predefined intervals for critical hyperparameters, a pivotal step in achieving superior results. Notably, the implementation of the hyperparameter optimisation also serves to reduce the manual efforts traditionally associated with fine-tuning model configurations.

```
# Step 1: Define Hyperparameter Function (via 'def
objective(trial)')
      # Step 1.1: Define Hyperparameters for Optimisation with
Tolerance Internals
      # Step 1.2: Execute the Training and Validation
# Step 2: Set Up the Study Logging for Optimisation Monitoring
(via 'optuna.logging.set_verbosity')
# Step 3: Create the Hyperparameter Optimisation Study (via
'optuna.create_study')
      # Step 3.1: Specify the Optimisation Direction (Minimise
or Maximise) (via 'study.optimise')
      # Step 3.2: Set the Number of Trials
# Step 4: Get the Best Hyperparameters from the Study (via
'study.best_params')
# Step 5: Utilise the Best Hyperparameters to train, validate and
test the final model
```

Figure 4.12 Optuna Hyperparameter Optimisation Implementation

As shown in Figure 4.12, the optimisation process unfolds through a carefully orchestrated series of steps during each trial. Initially, we establish a logging mechanism to monitor and record the progress of the optimisation process. Following the logging setup, a study is created to manage and orchestrate the optimisation trials. The number of trials hyperparameter is adjustable to satisfy the computational resources available. It is noteworthy that the objective function is optimised with the explicit aim of minimising the validation loss metric. The choice of validation loss as the optimisation target is a strategic decision because it serves as a crucial indicator of the model generalisation to unseen data, playing a pivotal role in preventing overfitting. The extraction of the best hyperparameters ensues upon the completion of the optimisation trials. Finally, set with the final refined hyperparameters, the subsequent phases involve the commencement of training, validation and testing for comprehensive evaluation of the model performance with the anticipation of achieving superior results.

CHAPTER 5

5. RESEARCH FINDINGS

In this section, we present the findings obtained from the simulations conducted as part of our research study on hybrid recommendation systems. Through rigorous experimentations, we analyse the advantages and disadvantages of our proposed models. By examining the outcomes of the analysis, we seek to elucidate the strengths and limitations of each approach, shedding light on their suitability for different scenarios.

By advancing our understanding of both centralised and federated learning paradigms within this context, we endeavour to contribute to the existing body of knowledge while informing subsequent research and development efforts in the field of recommender systems.

5.1 EXPERIMENTATION ENVIRONMENT

We conducted our experimentations on a computing environment with 2GB of RAM and 1GB of GPU RAM. The GPU used for the experimentation was the Nvidia T4, while the CPU was a 2-Core Intel(R) Xeon(R) CPU running at 2.20GHz. Average computing runtimes are shown as below in Table 5.1.

Table 5.1 Average Computing Runtimes

Context	TimeElapsed	AdditionalNotes
CentCore	48min	-
CentCore with DP	6h3m	-
CentCore with HPO	55min	-
CentCore with HPO and DP	3h21m	-
FedCore	19min	61% less than CentCore
FedCore with DP	45min	87,6% less than CentCore with DP
FedCore with HPO	33min	40% less than CentCore with HPO
FedCore with HPO and DP	37min	81,5% less than CentCore with HPO and DP

In our study, we observed that due to large volume of Movielens-1M source dataset, training a centralised model always takes a longer time compared to training a federated model while, additionally, requiring more computing overhead. Especially, for IoT units with limited powers, training a centralised model on such systems overuse existing computing resources.

5.2 PERFORMANCE METRICS

In our study, we utilised 3 different performance metrics as MSE, MAE, NDCG. These metrics are known as ideal performance metrics for recommender systems. We present our performance metrics results in Appendix-A.

For MSE and MAE, lower values are better. A lower error value indicates that the model predictions are closer to the actual values, which is better accuracy in the model predictions. In contrast, for NDCG, higher values are better. It indicates that the recommendation system is ranking the items more accurately, with the most relevant items appearing higher in the list. As we cover in our research, our goal is to create an equal or better performing federated model

compared to centralised model. In terms of restricting the evaluation scope only down to MSE, MAE and NDCG performance metrics, our study concludes the following comments:

- Centralised core model performs slightly better than federated core model.
- When differential privacy considerations are added to the core model, privacy budget significantly affects each model performance. Under lower privacy settings, some federated setups perform closer to centralised models. Under higher privacy settings, all centralised models perform better than federated models.
- Hyperparameter optimisation increases the capabilities of federated models where it performs as equal as centralised models.
- After combining hyperparameter optimisation capabilities with differential privacy settings, federated models perform significantly better than centralised models.

5.3 EVALUATION

In our study, we tested 42 different scenarios.[1] In centralised approach, we test the models once due to resource and time constraints; in contrast, in federated approach, we test the models twice and average the outputs. Below, we present our evaluation results in case-by-case and comparative format based on relative training, validation and testing losses.

[1] Kindly refer to the following link to access all evaluation results of our research: https://tinyurl.com/22comp5002

5.3.1 Case-by-Case Analysis

In the first part of the analysis, we publish the results of each tested scenario to obtain a transparent view.

Centralised Core: In our study, the centralised core model is the first and primitive model we create. As shown in Figure 5.1, this base model doesn't show any overfitting and we succeeded to steadily train our model. We took loss values of this model as reference for all other further developed models to have a fair comparison in between.

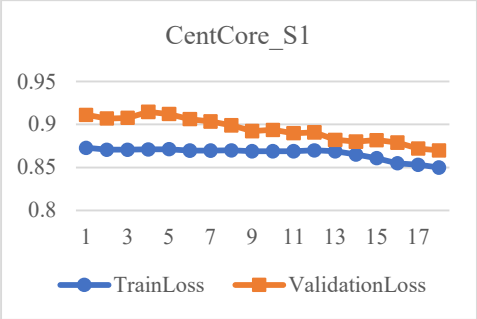

Figure 5.1 Centralised Core

Centralised Core with Differential Privacy: We integrate differential privacy capabilities into the centralised core model as two parts: lower and higher privacy. As expected, lower privacy settings affect the core model less compared to higher privacy settings, but also providing less privacy guarantee. As shown in Figure 5.2, we observe that the model with lower privacy settings performs very similar to core model. On the other hand, with almost 40% performance degradation, the model with higher privacy settings performs worse but provides more privacy guarantee. Additionally, we note that if a model is trained on large-scale datasets, differential privacy efforts may have a more negative impact on model performance, making training a federated model more advantageous.

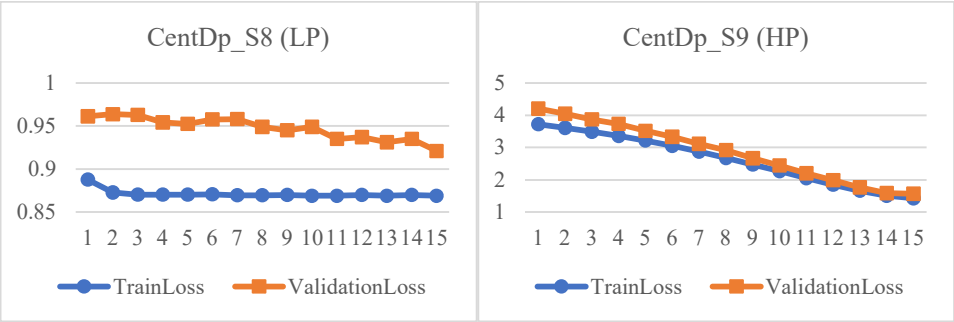

Figure 5.2 Centralised Core with Differential Privacy in Lower and Higher Privacy Settings

Centralised Core with Hyperparameter Optimisation: We add hyperparameter optimisation capabilities into the centralised core model to increase the model performance. As shown in Figure 5.3, the core model performance increases 27% by effectively searching the hyperparameter space with carefully selected hyperparameter intervals.

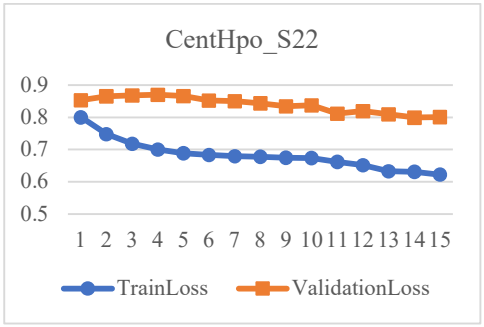

Figure 5.3 Centralised Core with Hyperparameter Optimisation

Centralised Core with Hyperparameter Optimisation and Differential Privacy: By further combining hyperparameter optimisation and differential privacy capabilities, we target to amortise the negativity of differential privacy with positivity of hyperparameter optimisation. As shown in Figure 5.4, we increase the previous model performance with lower privacy considerations

15%, 17% with higher privacy considerations. Furthermore, we observe that when a model is perturbed to a greater extent than another model, it is possible that the more perturbed model may have greater potential for improvement.

Figure 5.4 Centralised Core with Hyperparameter Optimisation and Differential Privacy in Lower and Higher Privacy Settings

Federated Core: In our study, the federated core model is the base model in distributed architecture. One of the most important aspects of such systems is choosing the most appropriate aggregation algorithm because that choice affects the whole communication network of the model and, thus, its performance. As shown in Figure 5.5, Figure 5.6 and Figure 5.7, we try 6 different aggregation algorithms. Among them, we note all models except the model with AFO converge well without showing any overfitting and we succeeded to steadily train our model, because AFO aggregation algorithm requires very sensitive model adjustments to be practical. In terms of model performance, the federated core models with FedAvgTrimmedMean, FedAvg, FedMedian and FedProx aggregation algorithms performs very similar compared to centralised core model but fail to outperform it.

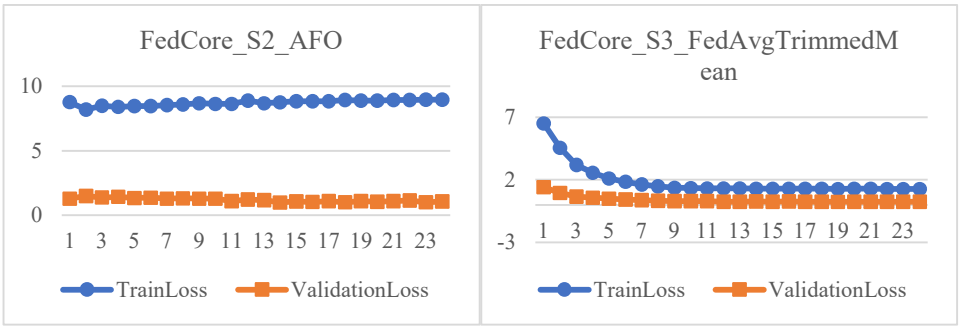

Figure 5.5 Federated Core (S2: AFO, S3: FedAvgTrimmedMean)

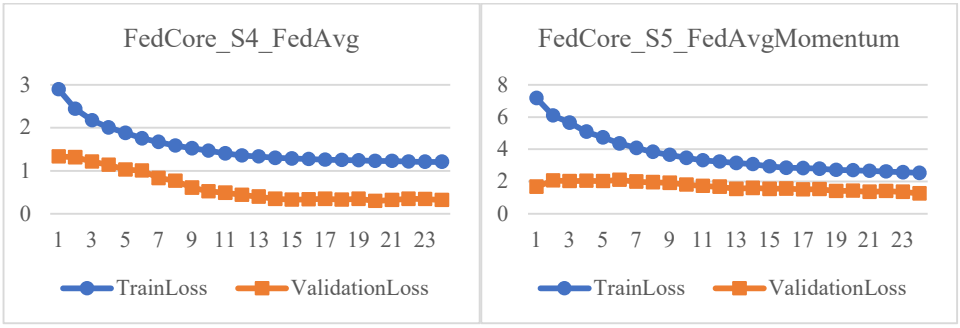

Figure 5.6 Federated Core (S4: FedAvg, S5: FedAvgMomentum)

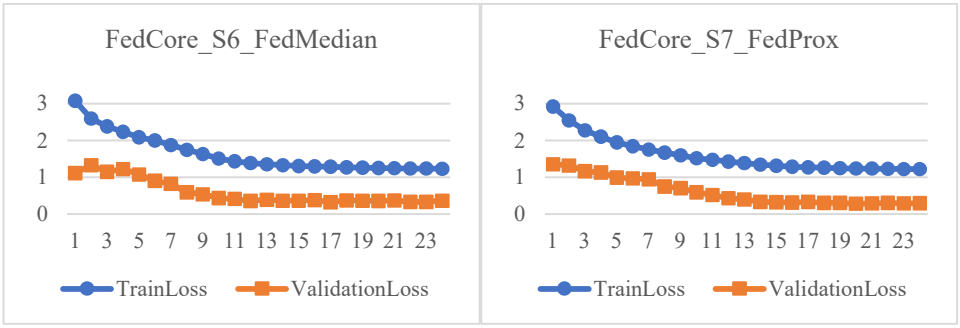

Figure 5.7 Federated Core (S6: FedMedian, S7: FedProx)

Federated Core with Differential Privacy: Similarly, we integrate differential privacy into the federated core model as two parts. Lower privacy settings affect the core model less compared to higher privacy settings, but also providing less privacy guarantee. As shown in Figure 5.8, Figure 5.9 and Figure

5.10, we observe that the models with lower privacy settings performs very similar to core model with only 1% performance degradation in average.

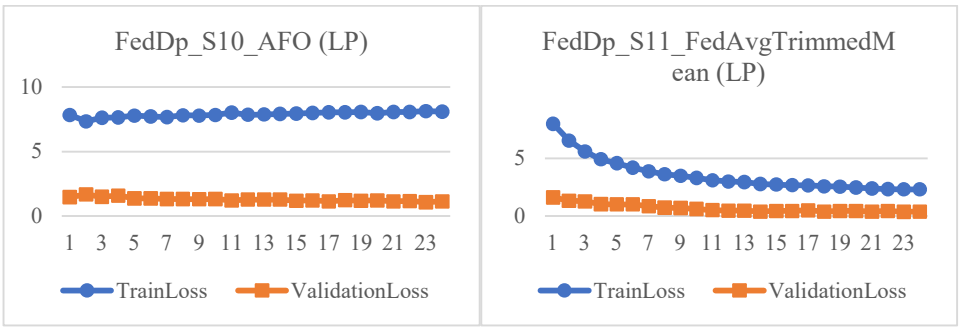

Figure 5.8 Federated Core with Differential Privacy in Lower Privacy Settings (S10: AFO, S11: FedAvgTrimmedMean)

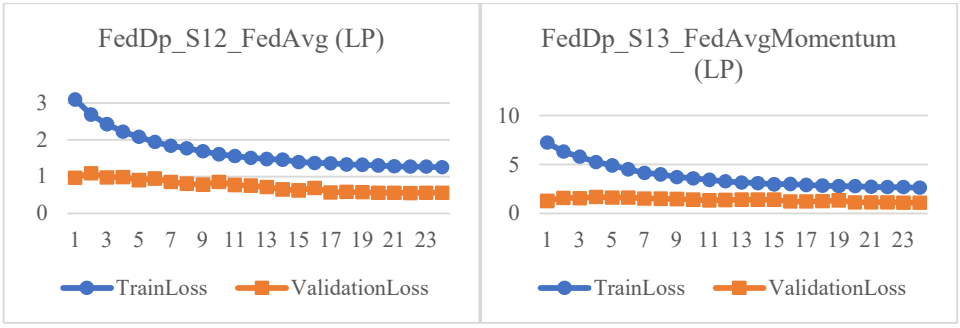

Figure 5.9 Federated Core with Differential Privacy in Lower Privacy Settings (S12: FedAvg, S13: FedAvgMomentum)

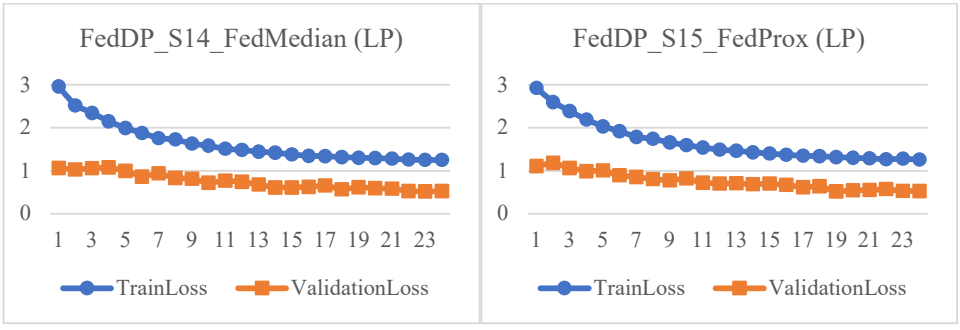

Figure 5.10 Federated Core with Differential Privacy in Lower Privacy Settings (S14: FedMedian, S15: FedProx)

Moreover, as shown in Figure 5.11, Figure 5.12 and Figure 5.13 the model with higher privacy settings falls behind with 20% performance degradation in average. The choice between lower and higher privacy settings really depends on the needs, an optimal performance degradation should not be problematic if the main concern is data privacy, especially when the trade-off can be patched with other important adjustments.

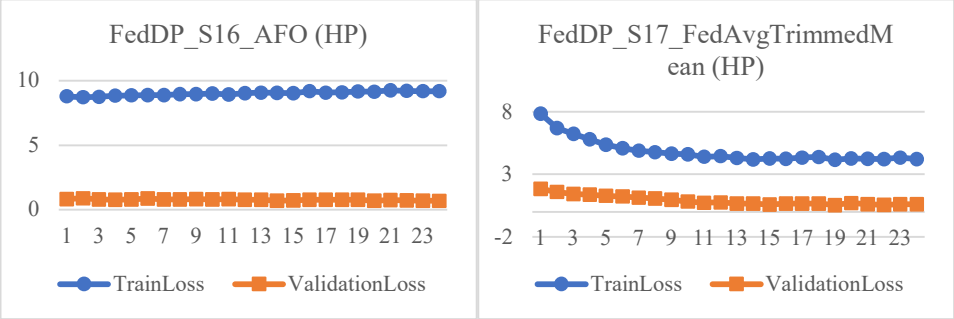

Figure 5.11 Federated Core with Differential Privacy in Higher Privacy Settings (S16: AFO, S17: FedAvgTrimmedMean)

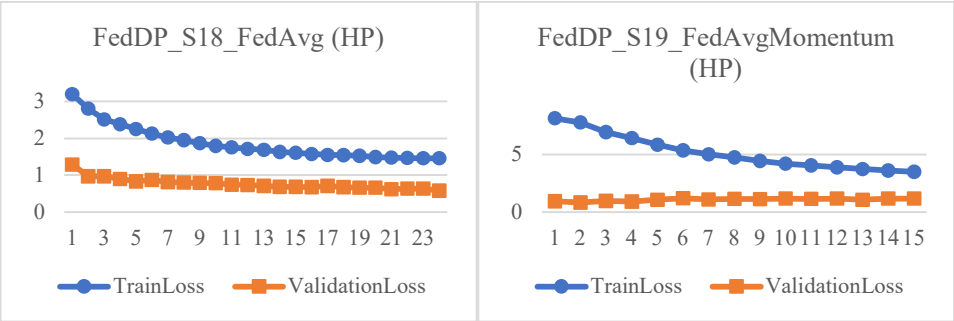

Figure 5.12 Federated Core with Differential Privacy in Higher Privacy Settings (S18: FedAvg, S19: FedAvgMomentum)

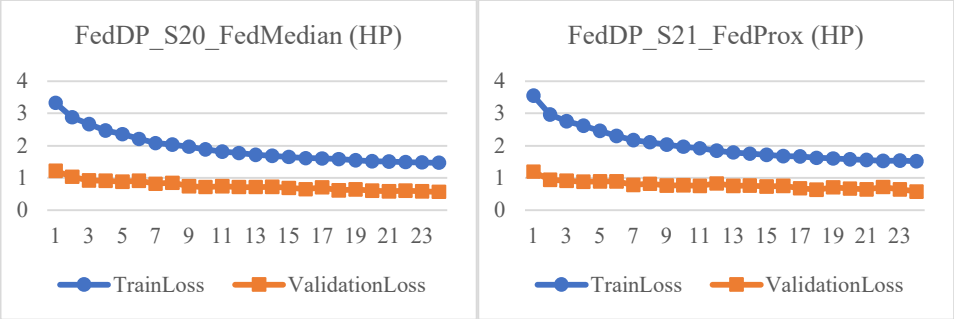

Figure 5.13 Federated Core with Differential Privacy in Higher Privacy Settings (S20: FedMedian, S21: FedProx)

Federated Core with Hyperparameter Optimisation: We add hyperparameter optimisation capabilities into the federated core model to increase the model performance. As shown in Figure 5.14, Figure 5.15 and Figure 5.16, the core model performance increases 50% in average by effectively searching the hyperparameter space with carefully selected hyperparameter intervals. During our study, we note that adjusting hyperparameters gets more crucial to increase the model performance in federated learning; especially, when the dataset is limited, the model may be more prune to overfitting. Consequently, with these results, we also succeed to outperform the centralised core model in federated models with FedAvg, FedMedian and FedProx aggregation algorithms.

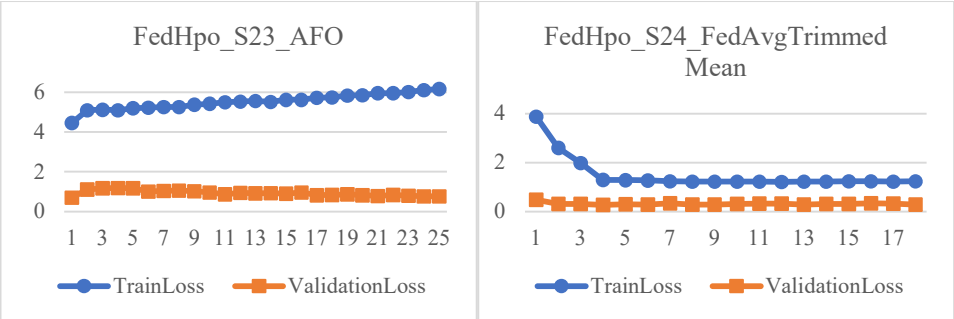

Figure 5.14 Federated Core with Hyperparameter Optimisation (S23: AFO, S24: FedAvgTrimmedMean)

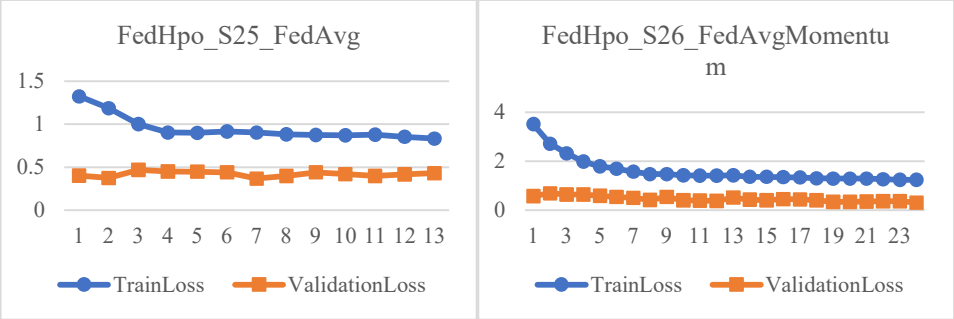

Figure 5.15 Federated Core with Hyperparameter Optimisation (S25: FedAvg, S26: FedAvgMomentum)

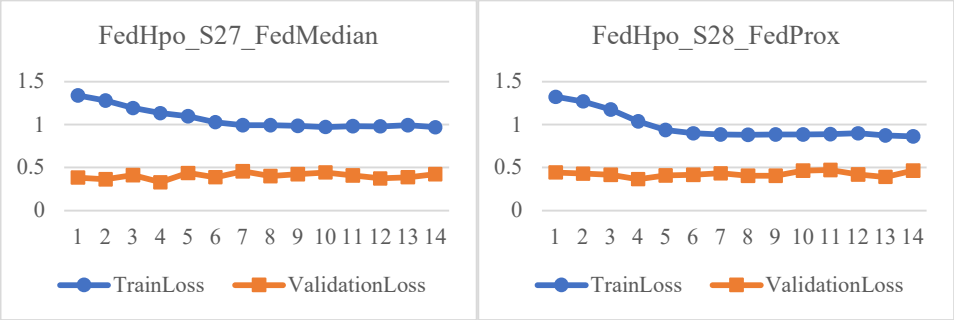

Figure 5.16 Federated Core with Hyperparameter Optimisation (S27: FedMedian, S28: FedProx)

Federated Core with Hyperparameter Optimisation and Differential Privacy: Again, by further combining hyperparameter optimisation and differential privacy capabilities, we target to amortise the negativity of differential privacy with positivity of hyperparameter optimisation. As shown in Figure 5.17, Figure 5.18 and Figure 5.19 we increase the previous model performance 5% in average.

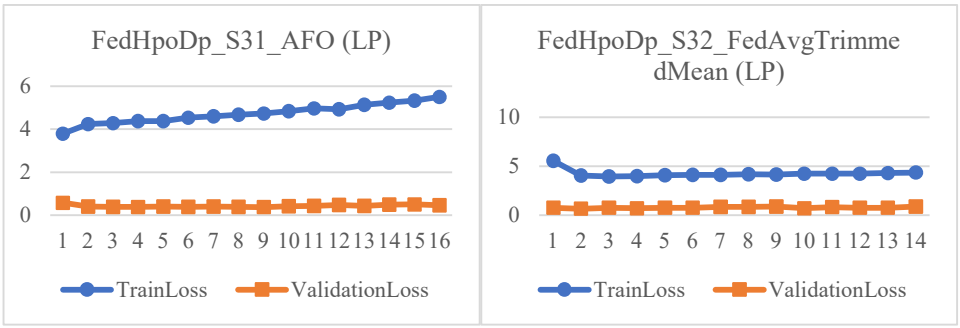

Figure 5.17 Federated Core with Hyperparameter Optimisation with Differential Privacy in Lower Privacy Settings (S31: AFO, S32: FedAvgTrimmedMean)

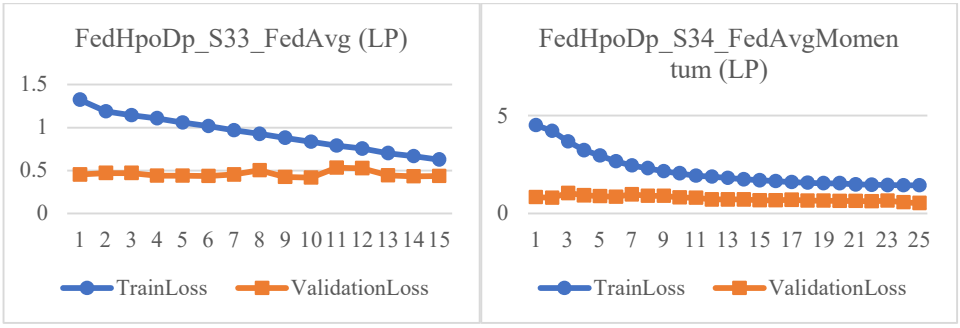

Figure 5.18 Federated Core with Hyperparameter Optimisation with Differential Privacy in Lower Privacy Settings (S33: FedAvg, S34: FedAvgMomentum)

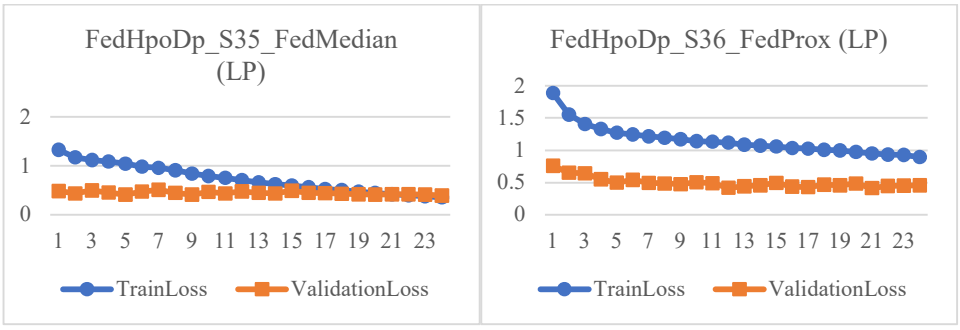

Figure 5.19 Federated Core with Hyperparameter Optimisation with Differential Privacy in Lower Privacy Settings (S35: FedMedian, S36: FedProx)

Consequently, as shown in Figure 5.20, Figure 5.21 and Figure 5.22 we observe that higher privacy concerns make the hyperparameter optimisation efforts harder because if higher privacy is prioritised over the model performance, the hyperparameter space gets resistant against obtaining the best hyperparameters, thus, halting or decreasing the model performance.

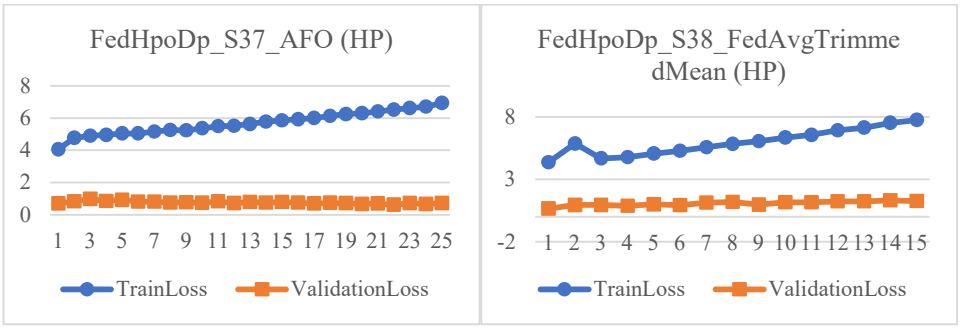

Figure 5.20 Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings (S37: AFO, S38: FedAvgTrimmedMean, S39: FedAvg)

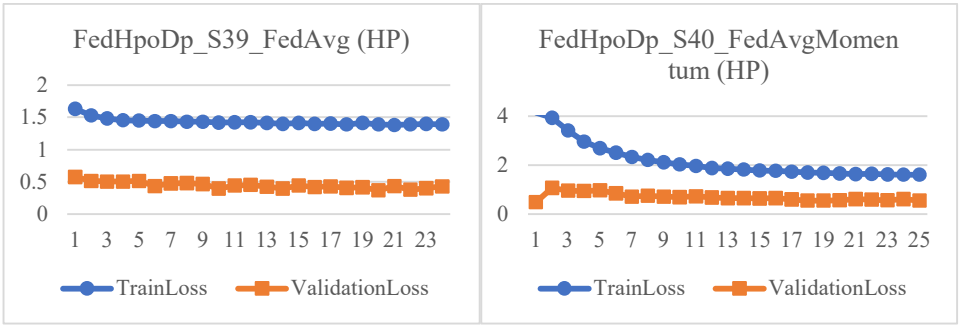

Figure 5.21 Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings (S39: FedAvg, S40: FedAvgMomentum)

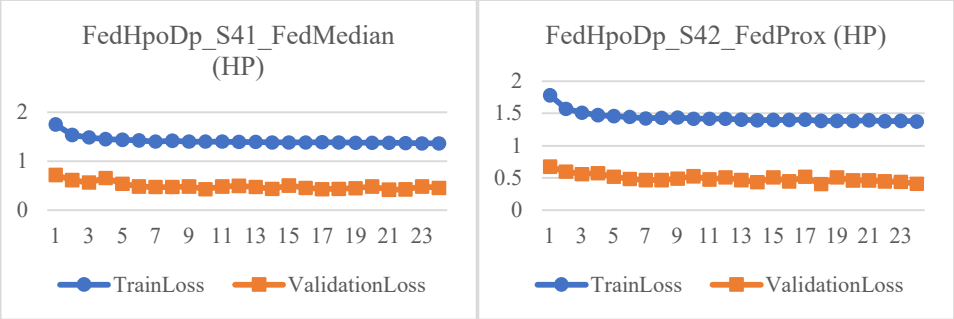

Figure 5.22 Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings (S41: FedMedian, S42: FedProx)

5.3.2 Comparative Analysis

In addition to the previous demonstration, we aim to enrich our study by making a comparative analysis. In the second part of the analysis, we segment and group the most crucial scenarios which perform well during our evaluation and be convenient in real-time use cases. Especially, observing that commonly used aggregation algorithms, such as FedAvg, FedProx and FedMedian, which also work well in our experimentation environment, affected our choice in that respect.

Centralised Core vs. Federated Core: As shown in Figure 5.23, we see a 29% performance loss in average in the federated core model compared to the centralised core model. We note that the main reason for the loss is that centralised models are more advantageous when the main concern is having a rich dataset. On the other hand, federated models are more advantageous when the main concern is having a low computational overhead as we mainly concerned in our study. Meanwhile, we prove that this trade-off is eliminated in favour of federated model by adding some additional measures later on.

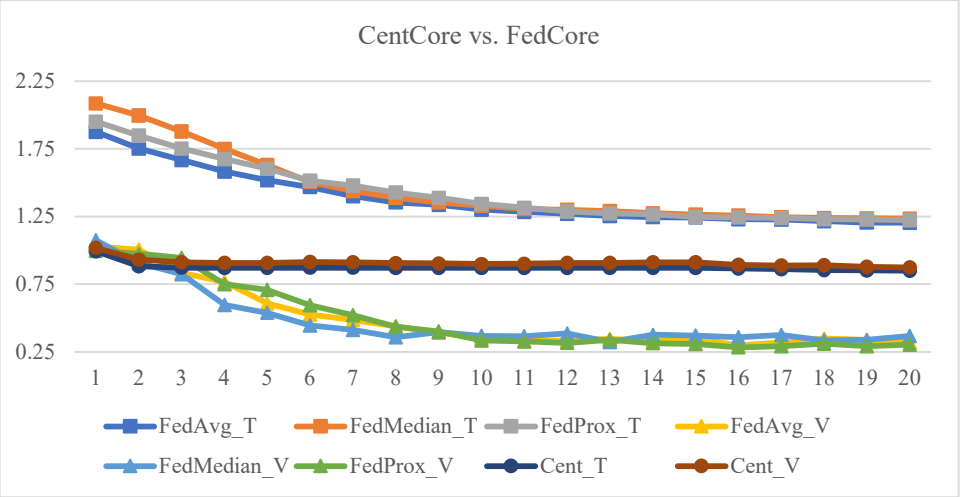

Figure 5.23 Centralised Core vs. Federated Core

Centralised Core with DP vs. Federated Core with DP: Close to our previous finding, we see a 31% performance loss in average in the federated core model under lower privacy settings compared to the centralised core model under lower privacy settings. As shown in Figure 5.24, we observe that having lower privacy settings affect the performance of our models less than 1%.

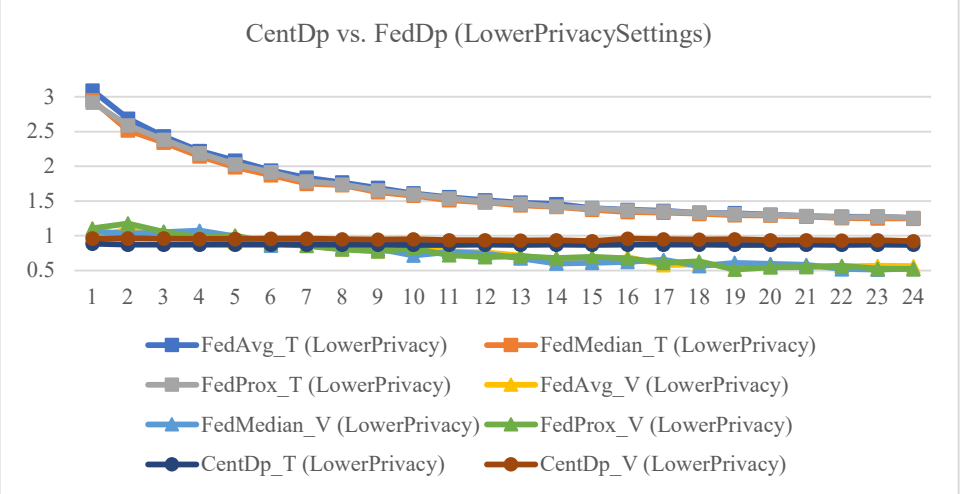

Figure 5.24 Centralised Core with Differential Privacy in Lower Privacy Settings vs. Federated Core with Differential Privacy in Lower Privacy Settings

Not performing better than the models under lower privacy settings, we see a 25% performance loss in average in the federated core model under higher privacy settings compared to the centralised core model under higher privacy settings. As shown in Figure 5.25, we observe that having higher privacy settings affect the performance of our centralised and federated models 29% and 10%, respectively.

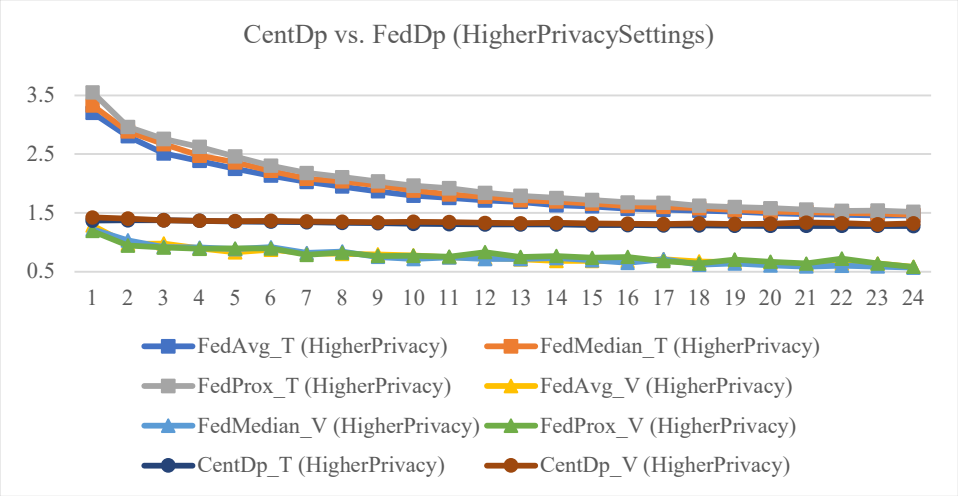

Figure 5.25 Centralised Core with Differential Privacy in Higher Privacy Settings vs. Federated Core with Differential Privacy in Higher Privacy Settings

Centralised Core with HPO vs. Federated Core with HPO: The addition of hyperparameter optimisation capabilities into the models significantly increases the model performance while removing manual effort and minimising human error to maintain the model hyperparameters. As shown in Figure 5.26, all model performances increases when hyperparameter optimisation capabilities are applied effectively. When compared to the centralised core model, the federated core model with hyperparameter optimisation capabilities is now performing better. Additionally, the impact of the hyperparameter optimisation capabilities can be further extended if hyperparameter searching space and number of trials are increased. In that

respect, it may be good practice that adding an automatic mechanism to search the hyperparameter space to improve model performance when a conversion from centralised to federated is planned without solely relying on the performance of the core model. Moreover, choosing the right hyperparameter becomes a bigger tackle when the available dataset is very limited where the model becomes more prone to overfitting and more resistant to convergence. No matter what, the practicality of this approach is delicate and must be carefully measured because each added capability may bottleneck the IoT devices with very limited computing resources.

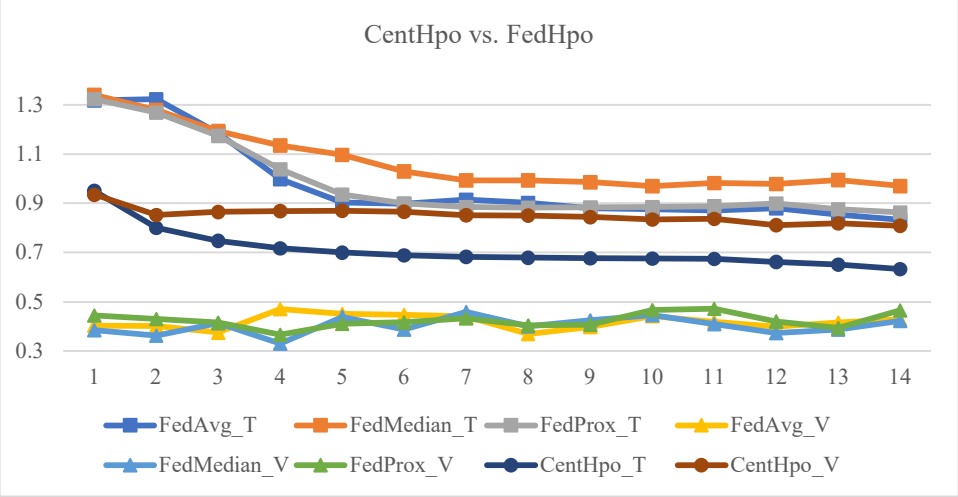

Figure 5.26 Centralised Core with Hyperparameter Optimisation vs. Federated Core with Hyperparameter Optimisation

Centralised Core with HPO and DP vs. Federated Core with HPO and DP: In the final stage of our study, with the most advanced features, we observe a strong improvement in the results. As we previously discussed, our federated core model can already outperform our centralised core model when the hyperparameters are carefully selected. Moreover, as a good practice, we further extend the models as shown in Figure 5.27. Even though the system is now working in federated settings and additional lower privacy guarantee exists, the

federated models with FedAvg and FedMedian aggregation algorithms perform 45% better than the centralised model within the same context and none of the models show any sign of overfitting.

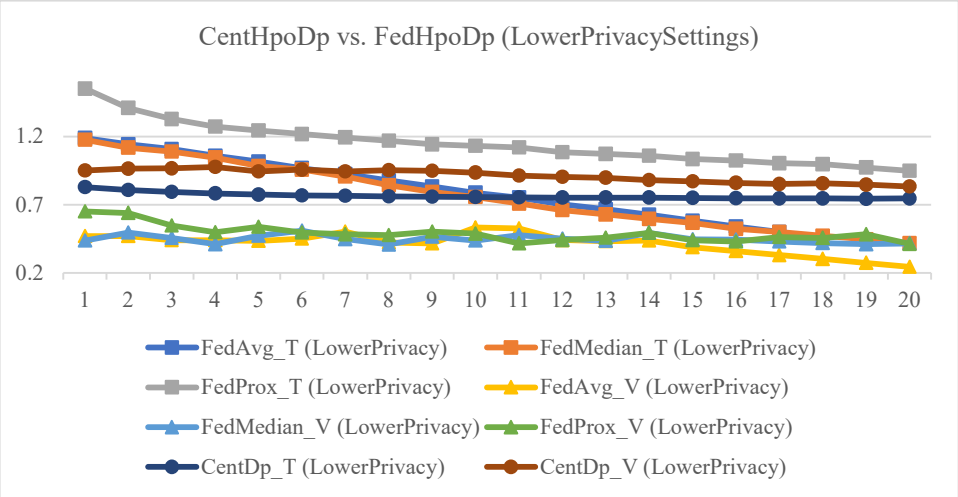

Figure 5.27 Centralised Core with Hyperparameter Optimisation and Differential Privacy in Lower Privacy Settings vs. Federated Core with Hyperparameter Optimisation and Differential Privacy in Lower Privacy Settings

Similarly, even though the system is now working in federated settings and additional higher privacy guarantee exists, as shown in Figure 5.28, all federated models perform 2% better than the centralised model within the same context and none of the models show any sign of overfitting. Consequently, compared to previous scenario under lower privacy guarantee, we note that performance degradation still exists as expected.

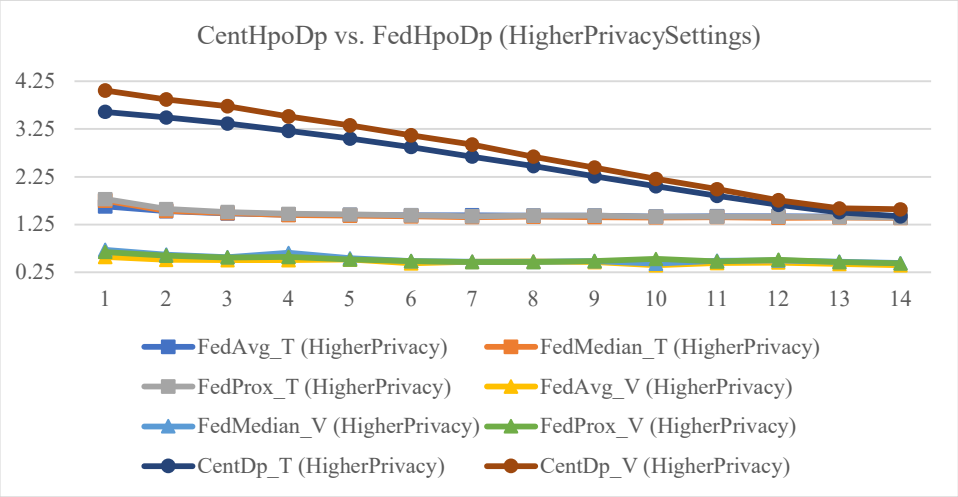

Figure 5.28 Centralised Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings vs. Federated Core with Hyperparameter Optimisation and Differential Privacy in Higher Privacy Settings

CONCLUSION AND RECOMMENDATIONS

As we conclude our study, this section represents the culmination of our research findings, reflections on the implications of our work, and directions for future investigation.

According to our experimentations, our additional inferences are summarised as below:

- Training a centralised model typically demands more computational resources and effort compared to a federated model. However, the advantage of a federated model becomes apparent when computational resources are limited, making it more adaptable to architecture changes. There is not a universal correct answer from this perspective but, generally, federated models are more reactive to changes made in the architecture. In contrast, centralised models are more demanding to adapt to changes.
- Privacy enhancing technologies, such as differential privacy, offers a commendable level of data privacy but the privacy budget must be balanced against other concerns: if the privacy budget is too high, the model utility decreases; if the privacy budget is too low, the data privacy decreases.
- Mechanisms such as hyperparameter optimisation can be useful to minimise the manual labour. It is also worth noting that the selection of hyperparameter intervals of such mechanisms should be carefully considered. Especially for complex models, optimisation may require considerable time.
- Considering centralised system consumes a lot of resources, converting a machine learning model from centralised to federated may be beneficial, especially in IoV context where such IoT units are widely dispersed and resource-constrained.

- In federated systems, the model may be more prone to overfitting due to resource constraints. In that respect, choosing the federated aggregation algorithm and adjusting hyperparameters gain more importance compared to centralised systems.
- Federated system can be easily deployed compared to centralised systems. In federated systems, the local datasets are very distributed, and a system can still be trained even with small computational resources. Moreover, when a participating client is disconnected from the network, the training process can continue.
- Federated systems offer more flexibility against abrupt system changes compared to centralised systems. Additional adjustments may take more effort in centralised systems than they take in federated systems.

Our evaluation yields valuable results as we demonstrate, and along with these achievements, we identify some key areas for improvement during our research. These improvement areas hold the potential to enhance our future researchers in the field. In that respect, our findings are summarised as below:

- Building on Privacy Enhancements: Our research already employs a privacy-enhancing technology. To further strengthen user privacy, our future work can explore encrypting device communication alongside implementing homomorphic encryption.
- Improving Fault Tolerance: While our current research necessitates a server for operation, our future work can explore two alternative approaches: implementing a serverless architecture or establishing a disaster recovery plan with a two-server structure.
- Exploring a Semi-Centralised Learning Approach for Enhanced Privacy and Efficiency: To explore the potential benefits of a semi-centralised privacy-preserving learning, our future work can investigate a hybrid system where training data remains local on client devices, validation and testing datasets are stored on a central server. This approach can offer privacy advantages while potentially improving communication efficiency.

- Optimising Privacy-Preserving Efforts for Large Datasets: To address the increasing processing time associated with differential privacy on large datasets, our future work can explore sweeping architectural changes aimed at accelerating computations.
- Improving Model Efficiency through Dynamic Background Computation Management: To further enhance the model effectiveness, our future work can focus on developing techniques to dynamically detect and reduce unnecessary background computations. The approach can involve implementing lightweight monitoring tools or exploring selective execution strategies to minimise resource consumption.
- Improving Model Generalisability: To enhance the model generalisability and real-world applicability, our future work can include a two-layered approach. A technical and specific root cause analysis can be conducted on models with lower performance. This root cause analysis identifies the factors contributing to these shortcomings via additional debugging and logging. Based on the findings, we can focus on extending the model capabilities to ensure robust performance in diverse real-world situations. This comprehensive approach not only improves overall model performance but also enhances its utility in practical applications.

REFERENCES

Bao, W., Wu, C., Guleng, S., Zhang, J., Yau, K. A., & Ji, Y. (2021). Edge computing-based joint client selection and networking scheme for federated learning in vehicular IoT. *China Communications, 18*(6), 39–52. https://doi.org/10.23919/jcc.2021.06.004

Chen, X., Li, X., & Li, P. (2020). Toward Communication Efficient Adaptive Gradient Method. *arXiv.* Retrieved August 06, 2024, from https://doi.org/10.1145/3412815.3416891

Duan, M., Liu, D., Chen, X., Tan, Y., Ren, J., Qiao, L., & Liang, L. (2019). Astraea: Self-Balancing Federated Learning for Improving Classification Accuracy of Mobile Deep Learning Applications. *IEEE 37th International Conference on Computer Design (ICCD)*. https://doi.org/10.1109/iccd46524.2019.00038

Hard, A., Kiddon, C. M., Ramage, D., Beaufays, F., Eichner, H., Rao, K., Mathews, R., & Augenstein, S. (2018). Federated Learning for Mobile keyboard prediction. *arXiv.* Retrieved August 06, 2024, from https://doi.org/10.48550/arxiv.1811.03604

Harper, F. M., & Konstan, J. A. (2015). The MovieLens datasets. *ACM Transactions on Interactive Intelligent Systems*, *5*(4), 1–19. https://doi.org/10.1145/2827872

Jiang, J., Hu, L., Hu, C., Liu, J., & Wang, Z. (2020). BACoMBO—Bandwidth-Aware Decentralized Federated Learning. *Electronics*, *9*(3), 440. https://doi.org/10.3390/electronics9030440

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2018). Federated optimisation in heterogeneous networks. *arXiv.* Retrieved August 06, 2024, from https://doi.org/10.48550/arxiv.1812.06127

Liu, L., Zhang, J., Song, S., & Letaief, K. B. (2019). Edge-Assisted Hierarchical Federated Learning with Non-IID Data. *arXiv.* Retrieved August 06, 2024, from https://arxiv.org/pdf/1905.06641.pdf

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. a. Y. (2016). Federated Learning of Deep Networks using Model Averaging. arXiv. Retrieved August 06, 2024, from https://doi.org/10.48550/arxiv.1602.05629

McMahan, H. B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. a. Y. (2017b). Communication-Efficient Learning of Deep Networks from Decentralized Data. *International Conference on Artificial Intelligence and Statistics*, *1*(3), 1273–1282. http://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf

Mills, J., Hu, J., & Min, G. (2020). Communication-Efficient federated Learning for Wireless edge intelligence in IoT. *IEEE Internet of Things Journal*, *7*(7), 5986–5994. https://doi.org/10.1109/jiot.2019.2956615

Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečný, J., Kumar, S., & McMahan, H. B. (2020). Adaptive federated Optimization. *arXiv.* Retrieved August 06, 2024, from https://doi.org/10.48550/arxiv.2003.00295

Wang, H., Kaplan, Z., Niu, D., & Li, B. (2020). Optimizing Federated Learning on Non-IID Data with Reinforcement Learning. *IEEE Conference on Computer Communications*. https://doi.org/10.1109infocom41043.2020.9155494

Xu, Z., Zhang, Y., Andrew, G., Choquette, C., Kairouz, P., Mcmahan, B., Rosenstock, J., & Zhang, Y. (2023). Federated learning of GBoard language models with differential privacy. *arXiv.* Retrieved August 06, 2024, from https://doi.org/10.18653/v1/2023.acl-industry.60

Yang, X., & Dong, Z. (2022). Kalman Filter-Based Differential Privacy Federated Learning Method. *Applied Sciences*, *12*(15), 77-87. https://doi.org/10.3390/app12157787

Zeng, S., Mi, B., & Huang, D. (2023). Emergency Vehicle Identification for Internet of Vehicles Based on Federated Learning and Homomorphic Encryption, *IEEE 12th Data Driven Control and Learning Systems Conference (DDCLS)*, 208-213. https://doi.org/10.1109/DDCLS58216.2023.10165866

Zhang, T., Gao, L., He, C., Zhang, M., Krishnamachari, B., & Avestimehr, A. S. (2022). Federated Learning for the Internet of Things: Applications, challenges, and opportunities. *IEEE Internet of Things Magazine*, *5*(1), 24–29. https://doi.org/10.1109/iotm.004.2100182

APPENDICES

APPENDIX A: MODEL MSE, MAE, NDCG PERFORMANCE METRICS

Context	ScenarioNo	MSE	MAE	NDCG	AdditionalNotes
CentCore	1	0.947313	0.75	0.9905	-
CentDp (LowerPrivacySettings)	8	0.979506	0.7564	0.9907	-
CentDp (HigherPrivacySettings)	9	1.317215	0.9056	0.9847	-
CentHpo	22	0.907447	0.7352	0.992	-
CentHpoDp (LowerPrivacySettings)	29	0.948871	0.7509	0.9914	-
CentHpoDp (HigherPrivacySettings)	30	4.209063	1.6113	0.9773	-
FedCore_AFO	2	3.268864	1.4452	0.9557	-
FedCore_FedAvgTrimmedMean	3	2.020378	1.1333	0.9522	-
FedCore_FedAvg	4	1.848784	1	0.9602	Closer to CentCore
FedCore_FedAvgMomentum	5	2.909754	1.362	0.9574	-
FedCore_FedMedian	6	1.990357	1.1316	0.9592	Closer to CentCore
FedCore_FedProx	7	1.906056	1.1087	0.9587	Closer to CentCore
FedDp_AFO (LowerPrivacySettings)	10	3.104996	1.407	0.9562	-
FedDp_FedAvgTrimmedMean (LowerPrivacySettings)	11	3.339391	1.4414	0.951	-
FedDp_FedAvg (LowerPrivacySettings)	12	2.007606	1	0.9604	Closer to CentDp

FedDp_FedAvgMomentum (LowerPrivacySettings)	13	2.900209	1.3585	0.9562	-
FedDp_FedMedian (LowerPrivacySettings)	14	1.925711	1.1121	0.9608	Closer to CentDp
FedDp_FedProx (LowerPrivacySettings)	15	1.937664	1.1137	0.9607	Closer to CentDp
FedDp_AFO (HigherPrivacySettings)	16	3.61114	1.5289	0.9535	-
FedDp_FedAvgTrimmedMean (HigherPrivacySettings)	17	4.14326	1.6019	0.95	-
FedDp_FedAvg (HigherPrivacySettings)	18	2.2506	1	0.9541	-
FedDp_FedAvgMomentum (HigherPrivacySettings)	19	3.28842	1.452	0.9532	-
FedDp_FedMedian (HigherPrivacySettings)	20	2.32471	1.224	0.9539	-
FedDp_ FedProx (HigherPrivacySettings)	21	2.40405	1.2459	0.9539	-
FedHpo_AFO	23	1.885129	1.0984	1.9672	-
FedHpo_FedAvgTrimmedMean	24	1.194649	0.8731	0.9658	Closer to CentHpo
FedHpo_FedAvg	25	0.843275	1	0.9819	Closer to CentHpo
FedHpo_FedAvgMomentum	26	1.022728	0.8057	0.9791	Closer to CentHpo
FedHpo_FedMedian	27	0.866575	0.7397	0.9816	As equal as CentHpo
FedHpo_FedProx	28	0.844561	0.7325	0.9821	As equal as CentHpo
FedHpoDp_AFO (LowerPrivacySettings)	31	1.730277	0.9117	0.9729	-
FedHpoDp_FedAvgTrimmed Mean (LowerPrivacySettings)	32	3.339391	1.4414	0.951	-
FedHpoDp_FedAvg (LowerPrivacySettings)	33	2.040327	1	0.9624	Closer to CentHpoDp

FedHpoDp_FedAvgMomentum (LowerPrivacySettings)	34	0.656748	0.6388	0.9882	Closer to CentHpoDp
FedHpoDp_FedMedian (LowerPrivacySettings)	35	1.388862	0.9368	0.9726	Closer to CentHpoDp
FedHpoDp_FedProx (LowerPrivacySettings)	36	0.570025	0.5954	0.9903	Better than CentHpoDp
FedHpoDp_AFO (HigherPrivacySettings)	37	1.050625	0.8146	0.979	Better than CentHpoDp
FedHpoDp_FedAvgTrimmed Mean (HigherPrivacySettings)	38	1.831962	1.0806	0.96	Better than CentHpoDp
FedHpoDp_FedAvg (HigherPrivacySettings)	39	3.496152	1	0.9529	As equal as CentHpoDp
FedHpoDp_FedAvgMomentum (HigherPrivacySettings)	40	1.667714	1.0322	0.9609	Better than CentHpoDp
FedHpoDp_FedMedian (HigherPrivacySettings)	41	1.460714	0.9738	0.9593	Better than CentHpoDp
FedHpoDp_FedProx (HigherPrivacySettings)	42	1.464826	0.974	0.9596	Better than CentHpoDp

APPENDIX B: SYSTEM HYPERPARAMETERS

	Centralised Core	**Federated Core**	**Centralised Core with DP**	**Centralised Core with HPO**	**Centralised Core with DP and HPO**	**Federated Core with DP**	**Federated Core with HPO**	**Federated Core with DP and HPO**
Epochs	25	25	25	25	25	25	25	25
Batch Size	32	32	32	32	32	32	32	32
Hidden Dimension	50	50	50	Inconstant	Inconstant	50	Inconstant	Inconstant
LSTM Hidden Dimension	32	32	32	Inconstant	Inconstant	32	Inconstant	Inconstant
Weight Decay	1.00E-03	1.00E-03	1.00E-03	Inconstant	Inconstant	1.00E-03	Inconstant	Inconstant
Learning Rate	1.00E-03	1.00E-03	1.00E-03	Inconstant	Inconstant	1.00E-03	Inconstant	Inconstant
Number of Groups	32	32	32	32	32	32	32	32
Dropout Rate	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Number of Clients	N/A	25	N/A	N/A	N/A	25	25	25
Number of Trials	N/A	N/A	N/A	5	5	N/A	5	5
Patience	5	5	5	5	5	5	5	5
Privacy Budget	N/A	N/A	1 | 5	N/A	1 | 5	1 | 5	N/A	1 | 5
Noise Multiplier	N/A	N/A	0.1 | 0.01	N/A	0.1 | 0.01	0.1 | 0.01	N/A	0.1 | 0.01

CURRICULUM VITAE