2021

A Deep learning-based system for de-identification of personal health information on mobile devices.

Musila, Daniel Mutiso
Faculty of Information Technology
Strathmore University

A DEEP LEARNING-BASED SYSTEM FOR DE-IDENTIFICATION OF PERSONAL HEALTH INFORMATION ON MOBILE DEVICES

Musila, Daniel Mutiso

Submitted in partial fulfilment of the requirements for degree of Master of Science in Computer-Based Information Systems at Strathmore University

School of Computing and Engineering Sciences

Strathmore University

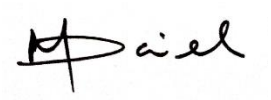
Nairobi, Kenya

September 2021

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Daniel Mutiso Musila



04-SEP-2021

Approval

The thesis of Daniel Mutiso Musila was reviewed and approved by the following:

Dr. Vincent Omwenga,
Senior Lecturer, School of Computing and Engineering Sciences,
Strathmore University

Dr. Julius Butime,
Dean, School of Computing and Engineering Sciences,
Strathmore University

Dr. Bernard Shibwabo,
Director of Graduate Studies,
Strathmore University

Abstract

Communication in healthcare has evolved from older technologies like pagers to present day smartphone devices. The change has been largely driven by the capability of smartphones to facilitate information exchange at greater speed and efficiency to manage the rising patient numbers, complexity of cases and the multiple disciplines in modern medicine. Instant messaging services like WhatsApp offer a channel which meets most of these needs. This communication often involves exchange of patient clinical data containing Protected Health Information (PHI). Various laws and policies have been enacted in various geographies and jurisdictions to safeguard the confidentiality of patients through strict management of PHI. During the normal course of care provision, healthcare professionals and organizations are expected to maintain full confidentiality and integrity of the data against unauthorized exposure. Whenever patient data needs to be shared with external parties for research use, informed consent must be obtained from the data subjects along with an oversight of their activities by a relevant review board. The widespread use of smartphones and popular instant messaging applications in modern healthcare however presents security and data protection challenges which need urgent addressing. De-identification of the data offers an avenue to address these concerns, allowing clinical data containing PHI to be shared among healthcare providers and/or researchers with minimized risks. Deep learning de-identification systems demonstrate superior performance over other approaches. They are generally deployed on high-end workstations in medical facilities and research centres, or on cloud-based infrastructure. However, on-premises deployments present infrastructural, connectivity and cost implications while cloud de-identification services may involve transmitting sensitive data across different jurisdictions therefore potentially breaching data residency regulations. On the other hand, smartphone use worldwide continues to see incredible growth with mobile processors becoming more powerful and versatile. Deep learning models can be deployed on Android-based smartphones to perform complex tasks such as de-identification of PHI. This is in line with the growing interest and research in edge computing, where computations are carried out as close to data sources as possible as an alternative to cloud computing. Concretely, this research proposes a mobile-based de-identification system, in which the deep learning model is optimized and embedded onto a smartphone application from which de-identification can be done. Specifically, Long Short-Term Memory (LSTM) artificial neural networks will be leveraged to develop a deep learning model which can then be ported onto the Android operating system to be embedded into a mobile de-identification application.

Keywords: protected health information, de-identification, neural network, smartphone

Table of contents

List of Tables

TABLE of Figures

Abbreviations and Acronyms

ANN	-	Artificial Neural Network
EMR	-	Electronic Medical Records
GPU	-	Graphical Processing Unit
IoT	-	Internet of Things
LSTM	-	Long Short-Term Memory
NLP	-	Natural Language Processing
RNN	-	Recurrent Neural Network
HIPAA	-	Health Insurance Portability and Accountability Act
NER	-	Named-Entity Recognition
PHI	-	Protected Health Information
PoS	-	Part-of-Speech

Definition of Terms

Data Residency

The physical location(s) of an organization's data and the branch of data storage management involved with issues concerned with managing data in the location(s). It is sometimes also referred to as data sovereignty.

Informed Consent

A person's voluntary agreement based upon adequate knowledge and understanding of relevant information, to participate in research or to undergo a diagnostic, therapeutic, or preventive procedure. In giving informed consent, subjects may not waive or appear to waive any of their legal rights, or release or appear to release the investigator, the sponsor, the institution, or agents thereof from liability for negligence.

Institutional Review Board (IRB)

A specially constituted, federally mandated review body established or designated by an entity to protect the welfare of human subjects recruited to participate in biomedical or behavioural research.

Health Insurance Portability and Accountability Act (HIPAA)

The rule which protects the privacy of individually identifiable health information in the United States. The privacy rule provides federal protections for personal health information held by covered entities and gives patients specific rights with respect to that information.

Chapter 1: Introduction

1.1 Background to the Study

Communication in healthcare has evolved from using older technologies like the use of pagers introduced in 1950s to present day smartphone devices. This change has been largely driven by the capability of smartphones to facilitate information exchange at greater speed and efficiency over the older technology. With rising patient numbers, complexity of cases to be addressed and the involvement of multiple disciplines in modern medicine, the requirement for healthcare provider communication to be rapid and more efficient increases. Instant messaging services like WhatsApp have offered healthcare professionals a channel which meets their communication needs, accessed via smartphones which are already ubiquitous within the larger society (O'Sullivan et al., 2017). The communication often involves exchange of patient clinical data which is crucial in delivering timely and appropriate care to individuals as well as for health and medical research purposes (Institute of Medicine (U. S.) and Grossmann, 2010; Yang et al., 2019).

Patient clinical data contains Protected Health Information (PHI). This is information which can be used to identify an individual and various laws and policies have been enacted in various geographies and jurisdictions to safeguard the confidentiality of patients through strict management of PHI. For example, Kenya's Data Protection Act (2019) defines data subject rights including a wide variety of grounds on which they may object to processing of their data, deletion, and portability. It also outlines data localization in which the government may enforce that certain processing of the data to only take place through servers located in Kenya (Greenleaf and Cottier, 2020). Generally, these laws require that informed consent be obtained from the data subjects along with an oversight of their activities by a relevant review board if the data is to be used in research and related activities. When the data is de-identified however, these restrictions are not necessary anymore and the data can be more easily shared with researchers (Cohen and Mello, 2018). For use in the normal course of care provision, the healthcare professionals and organizations are expected to maintain full confidentiality and integrity of the data against unauthorized exposure.

While smartphones along with popular instant messaging applications have already been informally adopted into modern healthcare, security and data protection remain key challenges which need urgent addressing. Some proposals have been made to provide healthcare workers with alternative approved versions of the instant messaging solution to mitigate these concerns (O'Sullivan et al., 2017). But other forms of data security breaches such as loss of mobile phones, transmission of sensitive healthcare data outside

the allowed jurisdiction and opaque ownership of the data from which serious data breaches can occur remain largely unaddressed (Corbett et al., 2019).

De-identification of the data offers an avenue to address these concerns. It refers to the process of removing all elements in the data which contain Protected Health Information to preserve patient confidentiality. This process can be done manually by human annotators, but the process is slow, error-prone, and expensive to carry out at scale (Dernoncourt et al., 2016). The de-identification process can also be automated. Two broad categories for this process are the use of rule-based systems, or alternatively implementations based on machine learning. The rule-based systems have a large industry presence since they do not require labelled data and are easy to implement and maintain. This is because they are simpler by design, being based on patterns such as regular expressions, and often include other rules designed by human experts. They are however not flexible to accommodate language changes or context of key phrases. Machine-learning de-identification systems on the other hand have demonstrated better performance without requiring hand-crafted rules and complex feature engineering otherwise required in the rule-based systems (Cohn et al., 2019). They achieve this using Artificial Neural Networks (ANN) trained on large sets of labelled data. The resultant system forms the core of an end-to-end pipeline, which can then be deployed on high-end workstations in medical facilities and research centres, or on cloud-based infrastructure.

In resource-constrained settings such as in many Low- and Middle-Income Countries (LMICs), deployment of these end-to-end de-identification systems on-premises can be challenging due the infrastructural, connectivity and cost implications. The de-identification cloud services may also be faced with the challenge of moving patient sensitive data across different jurisdictions, especially in places where data residency rules and policies have been put in place. This is besides the requirement for secure transmission of these data quantities securely to and from the cloud and the equally high bandwidth requirements not present in these areas (Wahl et al., 2018). On the other hand, smartphone use worldwide continues to see incredible growth, with mobile processors becoming more and more powerful and versatile. Deep learning models can now be deployed on Android-based smartphones using TensorFlow's Object Detection API as well as other platforms such as Facebook's PyTorch Mobile (Al-Azzoa et al., 2018). Industry key players including Samsung, Apple and Google are indeed facilitating this shift by releasing incrementally powerful smartphones which can locally host and execute complex machine learning models (Benedetto et al., 2018).

The progress made in enabling complex deep learning models onto smartphones coincides with a growing interest and research into edge computing. This approach involves computations being carried out as close to data sources as possible and generally

limiting cloud traffic (Merenda et al., 2020). Other potential benefits with this approach include lower communication costs, lower delays in obtaining responses, and lower exposure of private data.

1.2 Problem Statement

Healthcare professionals share information with peers and patients as part of their daily work during care provision. Indeed, exchange of information between care providers forms a fundamental part in modern medicine. The use of smartphones in general and instant messaging applications such as WhatsApp has replaced the traditional approved communication channels in what some researcher have termed as a case of "back-door adoption", where technologies which are very simple to use are incorporated without initial approval, adoption approaches or policy definition (De Benedictis et al., 2019). Of particular concern is the fact that while healthcare professionals perceive the usage of these applications as not safe for themselves and the patients, they still use it anyway for communication and information sharing. Serious data breaches can occur as a result.

Research has shown that with the training of users and introduction of a web application suitable for sending de-identified patient information, fewer healthcare professionals were noted to send and receive patient PHI. The frequency with which clinical images were saved on the smartphones was observed to decrease (Corbett et al., 2019). Therefore, de-identification of data to be communicated reduces the privacy and security risks involved. This is especially so in the case of files stored on a smartphone or shared online or in a forum: it is the responsibility of the poster to take care against breaching any confidentiality or information protection for the patient (Kamel Boulos et al., 2016). Advanced automated de-identification approaches based on Artificial Neural Networks can carry out this task with greater flexibility, adaptability, and generalizability (Dernoncourt et al., 2016).

The deployment of these machine-learning based systems, specifically the ones based on deep learning models is computationally intensive (Agarwal and Alam, 2020). The de-identification functionality is therefore usually provided as a web service running on on-premises or cloud servers. This set up presents a few challenges in that sensitive data is transmitted over networks presenting a security risk, while in other cases the sensitive data may transit out of a country where data residency rules are enforced. In keeping with the current interest and growing trend into edge computing, a localized solution in which de-identification is done on the smartphone would mitigate these challenges. This research seeks to develop a deep-learning model for de-identification then port and integrate it onto a mobile application. This would allow de-identification to be possible on smartphones without necessarily transmitting the data to online de-identification services.

1.3 Objectives

1.3.1 General objective

The purpose of this study is to explore de-identification as a Named Entity Recognition problem based on deep learning networks with an aim of developing a Protected Health Information de-identification model that can be trained and embedded onto a mobile application. This application will provide the smartphone users with the ability to de-identify sensitive data on their device therefore helping mitigate any security and privacy risks associated with sharing of sensitive data.

1.3.2 Specific objectives

1. To review the process of de-identification of protected health information
2. To appraise existing models, algorithms and frameworks that can be applied in de-identification of PHI
3. To design and develop a deep learning-based de-identification model that can be embedded into a mobile application
4. To test the developed de-identification system

1.4 Research Questions

1. How is de-identification of protected health information done?
2. Which existing models, algorithms and frameworks can be applied in de-identification of protected health information?
3. How can a deep learning-based mobile de-identification system be developed?
4. How can the developed system be tested?

1.5 Justification

Sharing of patient information between clinicians is an indispensable component of modern medicine. The communication needs to be fast and appropriate, while responsibly handling PHI to minimize security and privacy risks. Mobile de-identification offers a way to mitigate these risks by providing smartphone users with the capability to only share data which has all PHI elements redacted or replaced with appropriate surrogates. While de-identification services are available as web services, mobile versions of the deployment are needed as well especially where connectivity is a challenge or data residency rules apply. This is also in keeping with the interest and

adoption of edge computing where part of the compute operations is moved to the edge devices in a network.

Upon solving the mobile de-identification problem, the functionality can indeed be incorporated into approved versions of instant messaging applications for use in healthcare. Clinicians will then have instant access to de-identification on their smartphones which are overwhelmingly used as a primary communication means. They are therefore able to de-identify data on-demand before sharing helping mitigate the associated risks. Sometimes the shared files can remain on the smartphone devices, and for this they can configure the application to carry out automated de-identification of any textual files containing PHI in select directories. Policy-wise, healthcare regulatory bodies and policy makers can consider the incorporation of de-identification services onto instant messaging solutions for healthcare given the critical role they play in modern medicine.

On the research front, there are several growing online repositories where individual patients can share their own clinical data onto public repositories for research purposes. However, privacy concerns remain especially given that the de-identification services may not be publicly available or may be controlled by the platform providers. The capability for users to de-identify their own clinical data for donation towards research indeed would enhance both the availability and quantity of data available for many areas of research in healthcare.

1.6 Scope and Limitation

The research is limited to de-identification of PHI in patient notes in textual format. It does not consider other formats such as audio recordings, or images. The mobile based de-identification system will be developed targeting deployment on the Android Mobile Operating System only (OS). A "mobile device" in the context of this research indeed refers to a smartphone running the Android OS.

1.7 Dissemination of the study results

The researcher intends to publish the research in a journal approved by the Strathmore University Faculty of IT. All the results and learnings from the study will be shared in this publication to benefit the larger research community.

1.8 Utilization of the results

By publishing this research as outlined in section 1.7, the researcher intends to have the results utilized by other researchers to create mobile applications based on more advanced deep learning models and further research in the field of edge computing, especially in the field of health IT applications.

VT OMNES
VNVM SINT

Chapter 2: Literature Review

2.1 Introduction

This section commences with reviewing Protected Health Information as contained in medical records in depth. The concept of de-identification of PHI is then introduced with a comparison of manual and automated de-identification systems. The chapter then delves into the various implementations of automated de-identification methods, specifically focussing on how RNNs are applied for Named Entity Recognition. The PyTorch Machine Learning Framework and its implementation on smartphones is then introduced. The researcher finally presents a conceptual framework.

2.2 Protected Health Information in Medical Records

Electronic medical records systems have resulted in the volumes clinical data in digital format, which is useful in medical investigations and research applications. Patient notes are particularly key as the information they contain cannot be found in other structured elements of the EMR. However, large amounts of protected personal and health information are contained in these records. This information comprises Protected Health Information (PHI), which in essence can be used to uniquely identify a person (Moore and Frye, 2019). There are laws and regulations in place which act to safeguard the confidentiality of persons referenced in the clinical data. In the United States for example, both the Common Rule and the HIPAA jointly require that consent be obtained from the persons whose data is contained in the clinical records, alongside approval and oversight by a relevant board for researchers to be allowed access to clinical data (Liu et al., 2015). HIPAA defines 18 different types of PHI as outlined in table 2.1. They must be removed from the clinical data, and only then can one consider that data to have been de-identified.

Table 2.1 HIPAA-defined PHI types

Categories	PHI	Details
AGE	AGE	All ages greater than 90
CONTACT	PHONE FAX EMAIL URL IP ADDRESS	Phone and fax, IP, URLs, and e-mail addresses
DATE	DATE	Dates (month and day parts)
ID	IDNUM MEDICAL RECORD DEVICE	Social Security identifiers Account identifiers Certificate numbers License numbers Medical record numbers

	HEALTH PLAN	Vehicle or vehicle identifiers
	BIOID	Health plan numbers
		Full-face photographs
LOCATION	CITY	City
	STREET	Street
	ZIP	Zip code
	ORGANIZATION	Employers
NAME	PATIENT	Patient and family member names
PROFESSION	PROFESSION	Profession

2.3 De-identification

De-identification refers to the process of removing PHI elements from a dataset such that the patient cannot be identified again (Dernoncourt et al., 2016). This process mitigates any privacy risks to patients and other individuals from whom the data is drawn. According to the Privacy Rule of HIPAA, patients need to grant authorization in writing specifying that they allow their PHI data to be disclosed. There are exceptions to this rule of course, examples being situations where the data is required for medical or legal reasons (Cohen and Mello, 2018; Moore and Frye, 2019). In cases where obtaining patient authorization is difficult or unrealistic, Institutional Review and/or Privacy Boards must indeed certify the impracticality of obtaining the consent from the patients. The must also determine that the research proposed does not introduce considerable risks to any party. However, data missing PHI elements can be freely distributed to cater for uses whether in research or commercial contexts.

Two ways through which de-identification can happen are outlined in the Privacy Rule, as illustrated in figure 2.1. The first method involves having an expert who is qualified in a particular domain determine the PHI data to be de-identified. In the second method, identifiers are removed combined with the fact that no knowledge should be present to suggest that the information retained can be used to uniquely identify an individual, whether alone or when combined with other information (Committee on Health Research and the Privacy of Health Information: The HIPAA Privacy Rule et al., 2009). Regardless of the method used, it's crucial to consider that the data produced from the de-identification still retains some risk of re-identification, meaning it could possibly be traced back to an individual and thereby violating the privacy and confidentiality requirements.

The de-identification standard for PHI is the HIPAA Privacy Rule. As per the standard, health information is considered fully de-identified in the absence of any rationale to suggest that it can be traced back to identify an individual. The "Expert Determination" method is the first method used in line with the standard to de-identify data. In this method, an expert (someone with in-depth knowledge and experience in general

statistical, scientific, and mathematical concepts) examines and concludes that it's improbable that such concepts could be applied to re-identify an individual. They hence indicate the re-identification risk to be very low, even when the de-identified data is combined with other information. For "Safe Harbor" technique, all PHI tags which can identify a person including related entities like family are. The entity concerned must also determine the likelihood of re-identification of the data to be very low, even when combined with other information. The de-identification standard is considered fulfilled if either of the methods discussed is satisfied.

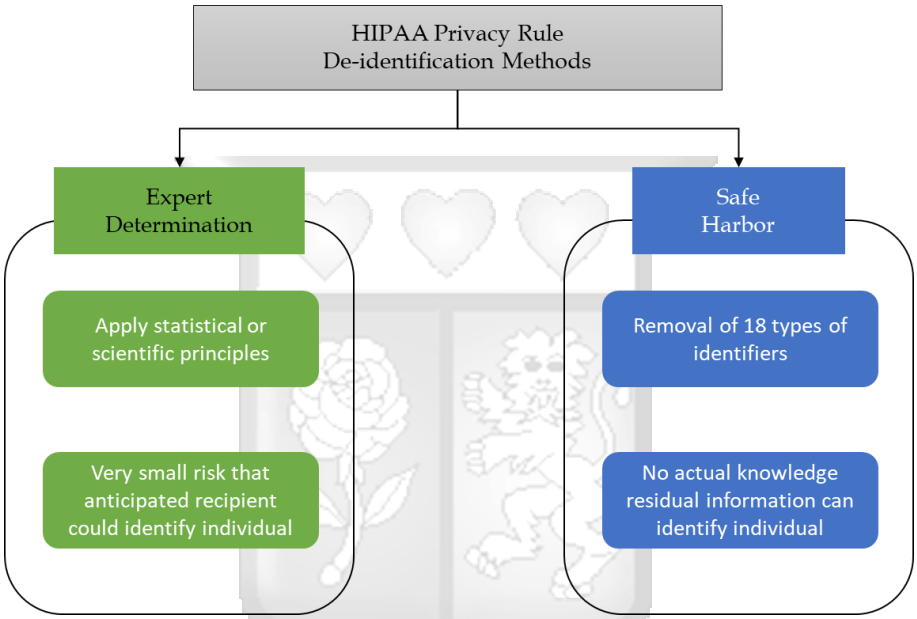


Figure 2.1: Pathways for de-identification of PHI according the Privacy Rule

Implementation-wise, de-identification may be automated or done manually. In manual de-identification, human annotators label the data and remove any PHI elements. There are some shortcomings with this method namely: access to clinical data is regulated, to typically only a few people per setting and as a result crowdsourcing cannot be used for the de-identification; human annotators make numerous mistakes; and the process can be costly as multiple annotators would be needed for the same task (Dernoncourt et al., 2016). A study established sample payment rates of 50USD each hour for human annotators. At the same time, the top hourly performance peaked at 20000 words in the same study (Douglass et al., 2005).

2.3.2 Automated De-Identification Systems

Given the shortcomings and limited applicability of using human annotators, automated de-identification systems offer an alternative that can be fast and inexpensive. Two types based on their implementation exist, and these are rule-based systems and machine-

learning systems. As implied in the name, rule-based implementations are fundamentally based on patterns, usually regular expressions, and human-identified rules.

Douglass et al. presented an automated method for de-identifying free-text nursing notes using regular expressions, simple rules, and lookup tables. They reported achieving an overall sensitivity score of 0.92 which was a remarkable improvement over an average human's sensitivity, with 0.44 as the predictive value (Douglass et al., 2005). The algorithm had a positive predictive value of 0.44. Neamatullah et al further demonstrated software package based on lookups from a dictionary, regular expressions, and other rules. The exact implementation was done on Perl with the goal of pattern matching. Performance-wise, the scored a precision and recall values of 0.749 and 0.967 respectively from a clinical dataset derived from nursing notes (Neamatullah et al., 2008).

The review of the rule-based systems highlights their key shortcomings: quality features for rules are challenging and time-consuming to develop; models require fine-tuning per each set of data, and any syntax or linguistic changes or typing errors inversely affect their robustness.

2.3.3 Machine Learning-Based De-Identification Systems

To address the shortcomings encountered with the rule-based de-identification systems, substantial research has been undertaken towards the use of machine-learning techniques, specifically the supervised approaches to learning (Dernoncourt et al., 2016). Machine learning refers to a set of methods for automatically detecting patterns in data, then using the patterns for some decision-making tasks under uncertainty. Most state-of-art models and systems for de-identifying data apply supervised machine learning techniques, or a hybrid setup combining both supervised machine learning and rules. In either implementation, the task of de-identifying clinical information is solved using Named Entity Recognition.

2.3.3.1 Named Entity Recognition (NER)

This is an activity of identifying the all the entities in some text (phrases/words) then classifying them according to some scheme (Nadkarni et al., 2011). NER is quite challenging owing to variations in word/phrase order, derivations where word forms can morph a of speech to a different one, word inflections either through tenses or comparative/superlative forms, synonymy of various words, and homographs (e.g., polysemy where homographs have related meanings or abbreviations which have different meanings. Nevertheless, NER is an appropriate way to formulate standard clinical NLP tasks concerned with identifying medical concepts and determining their semantic categories. For de-identification specifically, the NER task focusses on identifying, classifying, and removal of patient PHI data rather than just identifying semantic categories.

2.3.3.2 Machine Learning-based Clinical NER for De-identification

Machine learning-based techniques have been leveraged to solve the challenges of de-identifying medical records in the form of a general labelling of sequences approach. The model developed systematically maps an input word sequence (derived from a patient note for example) to predefined labels, which correspond to the PHI categories. A set of training data with all PHI elements manually labelled is required, and the model extracts the various linguistic and lexical features. Since the features are learnt in the training process, these machine-learning methods have superior generalizability to new data compared to the rule-based methods. This means that they will perform better when deployed to de-identify data which does not contain vocabulary encountered in the training set.

Guo et al. explored the use of Support Vector Machines (SVM) in de-identification tasks. Using a combination of basic token-level features, entities recognized by a press agency system for scooping information comprising both rules and other entity-specific knowledge and scored 0.9869 for the weighted F score (Guo et al., 2006). The authors however noted that the system performed less well when trained with the entire corpus, pointing to a problem of overfitting. He et al. proposed a system to be applied in de-identifying PHI in medical data whose basis was Conditional Random Fields. The system had a CRF layer on top of a separate module which processed the data to tokenize it using via regular expressions (He et al., 2015). The scores for this system were 0.9232 for the F1-score evaluating i2b2 data at an entity level. Indeed, there are many machine learning-based approaches based on CRFs and SVMs as discussed above, as well as Maximum Entropy (ME) and Structured Support Vector Machined (SSVMs). All generally perform well in clinical NER tasks such as de-identification. In a survey of deep learning methods for de-identification, one study found that these approaches achieved state-of-the-art performance in a number of NLP challenges on de-identification (Yang et al., 2019).

2.4 Deep Learning Models for NER

The critical factor in the development of these machine learning identification systems is their ability to extract useful features from the text. In this regard, deep learning models have demonstrated even better performance in NER tasks in the clinical domains (Yang et al., 2019). This is enabled by a of a breakthrough in distributed word representation using word embedding algorithms. Deep learning NLP methods have demonstrated their ability to capture numerous rich features on a low-dimension matrix, solving the problem of feature engineering encountered in rule-based approaches. Most deep learning architectures for NLP applications are built upon RNNs.

2.4.1 Recurrent Neural Networks

A RNNs memory is built upon on previous information, enabling it to predict the present output conditioned on past features encountered. For sequential data as is the case for most NLP tasks, RNNs are capable of capturing and retaining temporal information (Agarwal and Alam, 2020). In NER tasks, the RNN will predict information on the current feature based on neighbouring features. Figure 2.2 illustrates the conceptual structure of a RNN, consisting of consecutive input, hidden and output layers:

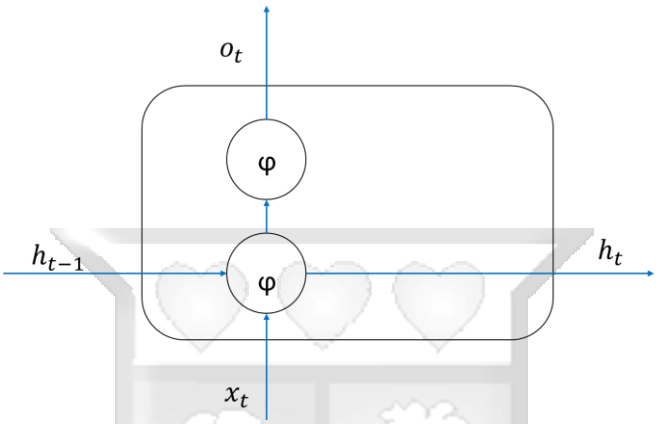


Figure 2.2: The Recurrent Neural Network node

For each input x_t and previous hidden state h_{t-1}, the RNN node produced an output o_t and a new hidden state h_t based on equations (1) and (2) below:

$$h_t = \varphi(w_{hh}h_{t-1} + w_{ih}x_t + b_h) \quad \text{(I)}$$

$$y_t = \varphi(w_{ho}h_t + b_o) \quad \text{(II)}$$

φ here refers to an activation function, examples being the ReLU (Rectified Linear Unit), sigmoid functions. w_{ih}, w_{hh}, w_{ho} are weight matrices parameterizing the input-to-hidden, hidden-to-hidden, and hidden-to-output connections. b_h and b_o are hidden and output biases for the two respectively.

2.4.2 LSTM RNN

RNNs have two drawbacks: the challenge of exploding gradients and diminishing gradients. This arises because of long-term dependencies in words in an input sequence. During training, the gradients propagate in reverse up to the first layer. One implication is that gradients emanating from the deep layers are propagated via many matrix multiplications. Over these propagations, the gradients will either: become smaller and smaller in value hence eventually diminishing and rendering the model unable to learn; or become larger and larger and eventually render the model unable to learn (exploding gradient problem). Unlike typical RNNs, LSTMs are designed to capture long-term dependencies in input sequences and can capture a wide range of features and

dependencies on words in a sequence making them suitable for Named Entity Recognition tasks where such long-term dependencies among words in input sequences abound. Figure 2.3 below demonstrates a classic LSTM node:

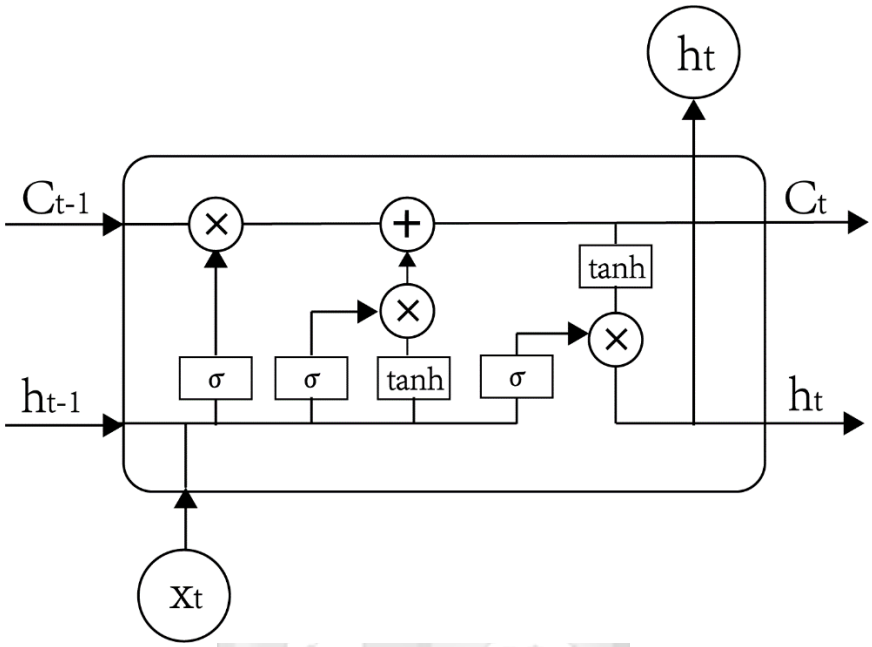


Figure 2.3: A classic LSTM node (Qiu et al., 2020)

The LSTM contains distinctive memory units in place of the typical inner (hidden) layers in a typical recurrent neural network. These memory units process the output state of the preceding unit (h_{t-1} *and* C_{t-1}) as well as the input for the current time step x_t as input. It combined these two in a vector $[h_{t-1}, x_t]$ to produce:

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f) \quad \text{(III)}$$

W_f represents the weight matrix and b_f represents the forget gate bias.

σ : sigmoid function.

In the current cell, the proportion of the state of the previous cell which will be kept is determined by the forget gate. Next, the input gate regulates the proportion of the input maintained in the current cell state according to the equations below:

$$i_t = \sigma(b_i + W_t . [h_{t-1}, x_t]) \quad \text{(IV)}$$

$$\hat{C}_t = tanh(b_c + W_c . [h_{t-1}, x_t]) \quad \text{(V)}$$

$$C_t = i_t * \hat{C}_t + f_t * C_{t-1} \quad \text{(VI)}$$

Finally, the output gate regulates the proportion of the state of the current cell kept:

$$O_t = \sigma(W_\sigma . [h_{t-1}, x_t] + b_o) \quad \text{(VII)}$$

$$h_t = \tanh(C_t) * O_t \quad \text{(VIII)}$$

2.4.3 LSTM RNNs for Named Entity Recognition

Studies have shown that models implemented based on LSTM and a Conditional Random Field (CRF) layer achieve superior performance, and state-of-art results in de-identification challenges (Yang et al., 2019). Additional considerations to improve model performance involve the use of Bidirectional LSTM and a token embedding layers in the initial layers of the model (Dernoncourt et al., 2016). Token embedding in offers an opportunity to further optimize the model and make it generalizable to new datasets. The common embedding schemes comprise of: token embedding which locate semantically similar words in the vector space, character-level token encoding through which models to identify sub-word patterns like roots and their variations. Another architecture for sequence labelling, a NLP task close to NER, employs a CNN layer for character-level representation, a bi-directional LSTM to model the context information for each word and finally a sequential CRF layer for jointly decoding the entire sentence (Ma and Hovy, 2016). This architecture achieves state-of-art performance on POS tagging and NER tasks, and were explored in this research to come up with a de-identification system

2.5 PyTorch Framework

There are many deep learning libraries, each having different features, performance, and optimization techniques in how the deep learning algorithms are implemented. Since this research relates to implementation of deep learning de-identification algorithms on smartphones, key metrics to consider are hardware utilization, hardware temperature and execution time. Smartphones have modest processing power and memory, alongside limitations in temperature and execution time to offer intuitive user experience. In an analysis between two popular frameworks namely PyTorch and Tensorflow, it has been shown that PyTorch summarily outperforms Tensorflow on six metrics namely: execution time for inference algorithm; execution time for training algorithm, CPU utilization rate, GPU utilization rate; GPU temperature and CPU temperature (Florencio et al., 2019). This study proposes to use this framework for implementation of the deep learning models and their conversion to mobile-compatible implementations.

PyTorch's TorchScript framework provides functionality for users to save models once training is completed to allow them to reference or use it later. It provides two high level decorators which produce Torch Script code from the PyTorch model (Moldovan et al., 2019). Torch Script code is a subset of Python saved from a Python process and can be loaded into a process with no Python dependency. This allows the TorchScript program to be run independent of the Python such as a standalone C++ program. The first decorator, *torch.jit.trace*, does tracing on the model during execution to come up with the computation graphs. This allows the resultant models to be optimized to a great degree and therefore easy to deploy on target platforms. Unfortunately, it has significant limitations in that the shape of the input data is expected to remain static and does not

allow the modules functions to be conditional. The other decorator, *torch.jit.script*, allows the use of dynamic functionalities as it only records the tensor operations during execution of the model. It creates an intermediate representation of the PyTorch module which is an internally optimised module utilizing PyTorch Just-In-Time (JIT) compilation at runtime.

2.6 Conceptual Framework

In this section, the literature reviewed in the preceding sections is linked to the research problem and objectives. It comprises a three-part conceptual model which outlines a pipeline detailing the implementation proposal for the de-identification system.

The first part of the model entails training a lightweight LSTM-CRF neural network capable of both de-identifying all PHI categories from textual input from patient notes. The model will be trained on a sample dataset and its performance evaluated against a validation dataset. The second stage covers the conversion of the model into a mobile-compatible format, considering all the necessary data pre-processing, transformation, and output extraction and processing. The last part entails embedding the LSTM-CRF in a mobile application and deployment on an Android smartphone.

The developed mobile de-identification system should be able to perform Named-Entity Recognition given a text file, identify all PHI tokens based on the weights and parameters learned during the training process. The system should then remove the PHI elements from the dataset hence de-identifying the text on the smartphone, without the need for establishing a network connection to use cloud-based de-identification systems. Figure 2.4 demonstrates this conceptual structure.

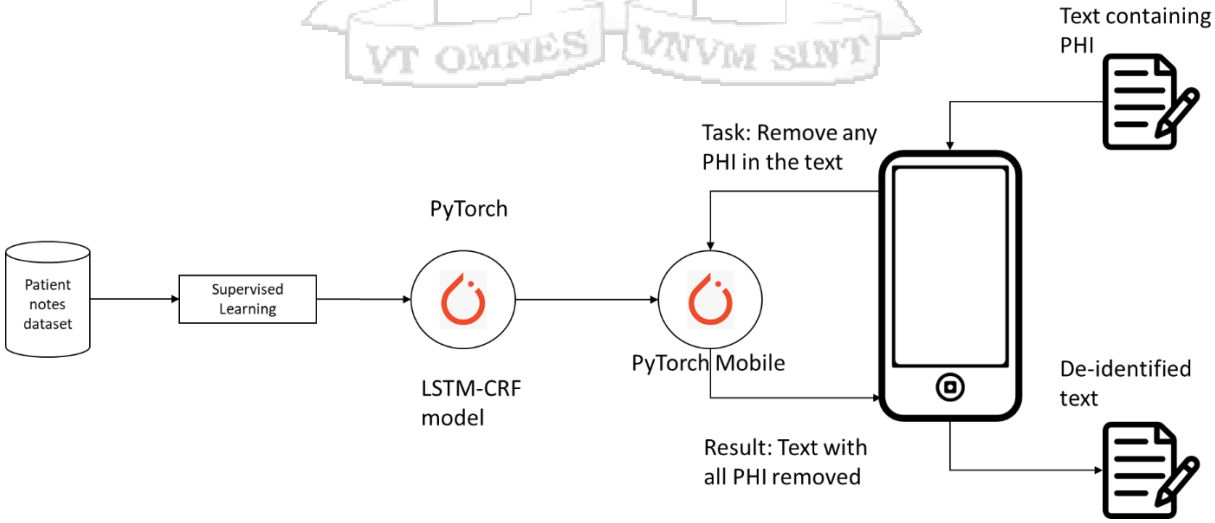


Figure 2.4: Mobile-based PHI De-Identification System conceptual framework

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

Research methodology is the organized way through which problem-solving is done. Both the steps used in the solving of a problem as well as the reasoning for each step are studied in depth (Kothari, 2004). In the context of information systems, methodology refers to the framework that is employed to organize, plan, and control the steps in the development of an information system. This chapter details the research design, the processing steps that were performed on the data, and the techniques that were employed to train and deploy a deep learning-based neural network model onto a mobile application which can de-identify PHI in data.

In line with the overarching objective of this research, the prototyping-based methodology was used. In this methodology, analysis, design and implementation phases are performed concurrently (Dennis et al., 2009). A key advantage of this methodology is that it helps in requirements refinement more quickly, as a user interacts with a prototype to understand its properties and capabilities.

3.2 Research Design

In research design, a conceptual framework within which the study is undertaken is established. Efficiency can then be realized, in which the study generates maximum information (Kothari, 2004). This research will take a prototyping approach to develop the lightweight LSTM-CRF model for PHI de-identification as well as the mobile de-identification application onto which the model will be embedded.

In the prototyping-based methodology, analysis, design and implementation phases are performed concurrently (Dennis et al., 2009). Essentially, a system prototype is developed once the basics of analysis and design are done to capture the minimal functionalities expected. This quick and dirty system is then evaluated, and the feedback used to reanalyse, redesign and re-implement another prototype incorporating the noted improvements. This methodology is summarized in figure 3.1. A key advantage of this methodology is that it helps in requirements refinement more quickly, as a user interacts with a prototype to understand its properties and capabilities.

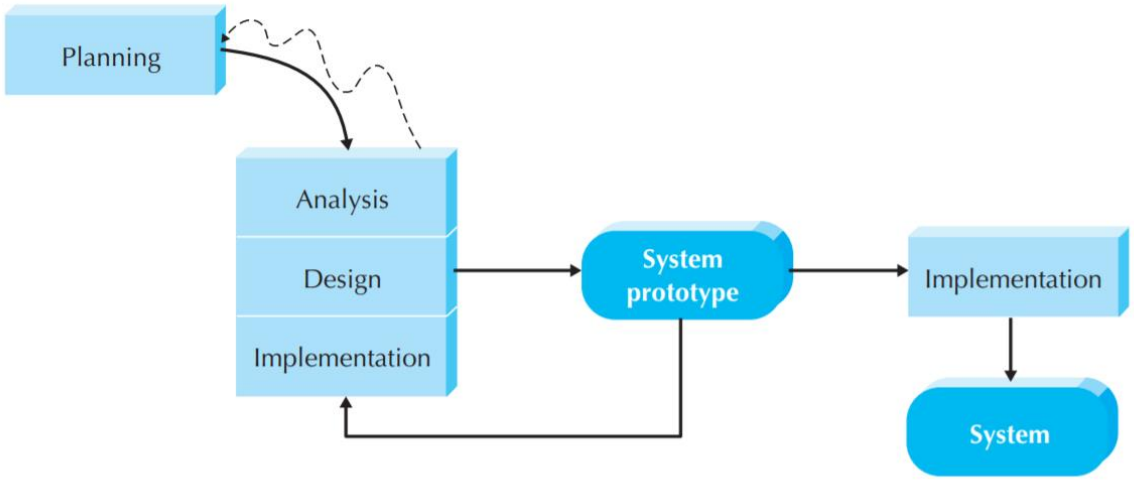


Figure 3.1: A Prototype-Based Methodology (Dennis et al., 2009)

Specifically, evolutionary prototyping was the methodology employed. It is differentiated from the other prototyping methodologies in that it produces models that comprise the eventual operational system (Ogedebe and Jacob, 2012). In the software context, a developer begins with determination of overall system objectives and functional system requirements. A prototype (functional model) is then rapidly developed, incorporating part of the features of the expected system. This model then has its functionality evaluated by a user and recommendations issued. The process is iteratively repeated until the model functionality is acceptable. The repetitive iterations allow the system to progress towards the defined performance, contrasted to throw-away prototyping.

This research was divided into two core components: development of the LSTM-CRF deep learning model for PHI de-identification and development of an Android Application to perform de-identification using the deep learning model.

3.3 LSTM-CRF Model Development

The LSTM-CRF deep learning model formed the basis for de-identification, being trained to identify PHI tokens from input text. This process was especially suited to evolutionary prototyping, in which a simple initial prototype is produced and evaluated for feedback followed by additional prototypes with added functionality and improvements until the final prototype is achieved.

Following the data collection activity as outlined in the subsequent sections, data pre-processing was done to format the data into a format suitable for processing by a neural network. The initial format of the data only involved tokens of words matched to respective output classes but over the course of the model development and training, the

structure of the data was updated to conform to the input parameter size of the neural network. The model itself was also incrementally determined, starting off with a simple LSTM network and subsequently adding pre-LSTM layers to merge word and character tokens as well as post-LSTM layers like a CRF layer to predict tag sequences.

During training, the hyper parameters (sizes of the various model layers, number of training steps, depths of the model layers) were experimentally determined. Initially, only a small portion of the training data was ingested into the model as the shapes of the input and output tensors and various transformations within the model were determined. Then training as done with various settings of the hyperparameters to determine the optimal values, beginning with a sufficiently low value and incrementally modifying them to determine the optimal combination. Finally, model evaluation was done upon the completion of each epoch to calculate the cross-validation loss on the validation dataset to determine the best–performing model parameters. The overall process was split into the following subcomponents:

3.3.1 Data Collection

In this research, the medical records the 2014 i2b2 Track 1 dataset were used. The records are fully de-identified, but maintain their longitudinal aspects with reasonable replacements put in place of the de-identified PHI instances (Stubbs et al., 2015). The Track 1 data is specific for de-identification but has been categorized into 6 main divisions namely: name, location, age, profession, identifiers, date, as well as 25 sub-categories. The 25 sub-categories comprise the i2b2-PHI, with additional categories for PHI on top of the ones defined by HIPAA to further secure the de-identification.

The i2b2 Track 1 data (2014) used in this research forms part of a larger dataset for clinical NLP, which is managed by the Medical School's Department of Biomedical Informatics at Harvard. For access to the data, terms of data access required the researcher to agree to and sign a Data Use Agreement, while outlining their purpose in a Research Purpose document. The dataset is presently available at https://portal.dbmi.hms.harvard.edu/. The use of this data was motivated by the fact it is the most notable openly-available datasets geared for de-identification tasks and has been extensively used in research to develop and evaluate algorithms for de-identification as a Named Entity Recognition task (Dernoncourt et al., 2016).

3.3.2 Sample Size

The i2b2 2014 dataset comprises manually annotated 1,304 medical records of 297 patients. The training set, drawn from PHI Gold Set 1 and PHI Gold Set 2, contains 17045 PHI instances while the test set contains 11462 PHI instances covering 25 PHI sub-categories covering all the defined HIPAA PHI categories.

In most cases, the size of the gold standard corpus in many NLP tasks is determined by ad hoc procedures which are guided by financial and personnel constraints, and no rationale provided for the sample sizes (Juckett, 2012). In this research, the i2b2 2014 dataset was chosen based on its use as a benchmark for evaluating state-of art de-identification systems for patient notes (Dernoncourt et al., 2016), being also the largest publicly available dataset on PHI NER for patient notes.

3.3.3 Data pre-processing

In the original format, the dataset is organized as follows: PHI Gold Set 1 for training, PHI Gold Set 2 for Training, PHI Gold Set-Fixed for testing. Each of these categories comprise of a compressed list of clinical notes in xml format. The schema of each xml file is as illustrated below.

```
<deIdi2b2>
        <TEXT>…</TEXT>
        <TAGS>
                <DATE id="" start="" end="" text="" TYPE="" comment=""/>
        </TAGS>
</ deIdi2b2>
```

Data pre-processing was therefore required to extract the clinical notes from the files along with the annotations for the different entities. This was achieved by tokenizing the text into sentences, and further into sentences of words with each word annotated with the correct PHI type as per the start and end indexes from the text.

Next, tagging was done on the data to tag every word in the text to a particular entity in the PHI entities. Specifically, the data was processed to the BIOES tagging scheme as it has been found to offer much more meaningful results over other tagging schemes (Lample et al., 2016). To achieve this, every token in the sentences was annotated as B-label if at the beginning of a named entity, an I-label if inside a named entity and otherwise annotated as O. Additionally, single entities were annotated as I-labels and tokens at the end of named entities as E-label.

The original dataset as retrieved is usually already split into training, validation, and testing sets in a 42.2%: 18.9%: 38.9% split respectively. The same ratios were maintained to have the data split into training and validation datasets to be used during the model training phase, and a testing dataset for evaluating the performance of the model.

3.3.4 Training of the LSTM-CRF Model

Training of the model was done on a Virtual Machine instance running on the Google Cloud Platform with 26GB memory, 4 x vCPU, 20GB SSD and 1 x NVIDIA Tesla T4 GPU. The dataset was transferred onto the virtual machine using the *gcloud* command line tool

from where pre-processing was carried out as described in section 3.3.3. The PyTorch framework was used for model implementation, along with the Jupyter Notebook web application to allow remote execution of training steps on the virtual machine. Training was done over 40 epochs, with evaluation of the cross-entropy loss being done after the end of every epoch. The minimum loss was stored and used to evaluate and store a snapshot of the model parameters corresponding to the lowest validation loss overall.

3.3.5 Testing the model

The training dataset as described in section 3.3.3 was used provide the gold standard for testing the model of the model. This data was not used/seen by the model during training therefore offered a good measure of the performance of the model to de-identify previously unseen data. *Precision*, *Recall,* and the *F-measure* were the measures used to evaluate the performance of the deep learning model. These were chosen over the more straightforward Accuracy metric, which may not provide the complete picture of the analysis especially when dealing with class-imbalanced data sets with a considerable disparity between the number of Positive and Negative labels, as is the case in this research.

Positive and Negative refers to two arbitrary outcome classes depending on the objective of the application. In this context, **Positive** would be a correct prediction that an entity is a PHI element and **Negative** being a prediction that an entity is not a PHI element. True Positive (TP) refers to an outcome in which the model correctly predicts the positive class while for a True Negative (TN) outcome the model would correctly predict the negative class. A False Positive (FP) on the other hand is an outcome whereby the model incorrectly labels the Positive class. Finally, in False Negative (FN), the model incorrectly predicts the negative class.

Precision quantifies the correct positive labels to all positive labels while Recall (or sensitivity) quantifies the correct positive class labels to all correct predictions (both negative and positive). F1 is the mean average of both the precision and the recall. These values are calculated as per the formulae below (Stubbs et al., 2015).

$$Precision\ (P) = \frac{true\ positives\ (TP)}{true\ positives\ (TP) + false\ positives\ (FP)}$$

$$Recall\ (R) = \frac{true\ positives\ (TP)}{true\ positives\ (TP) + false\ negatives\ (FN)}$$

$$F1\ = 2 * \frac{P * R}{P + R}$$

3.4 Embedding the LSTM-CRF model onto an Android Application

And Android application was developed to incorporate the deep learning model obtained to enable users to carry out de-identification on compatible smartphones in line with the overall objective of this research. The development of this application was done through prototyping in which an initial version of the application providing some of the desired functionality was developed based on the requirements and design and iteratively reanalysed, redesigned, and improved to accommodate improvements. PyTorch Mobile modules were used to facilitate the integration as illustrated in figure 3.2.

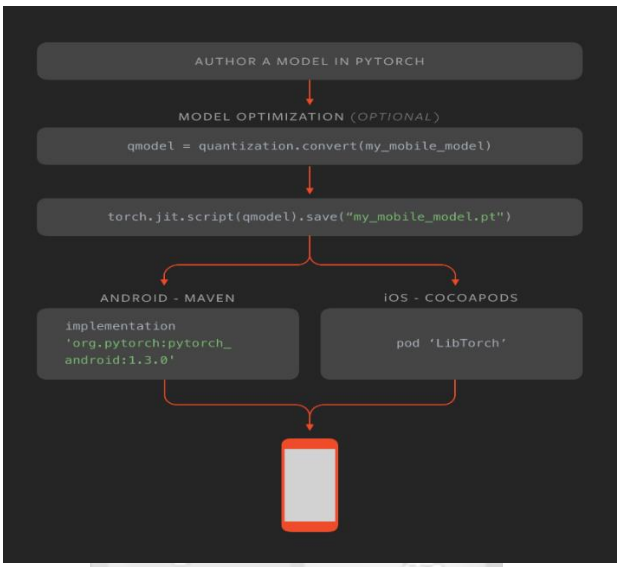


Figure 3.2: An end-to-end workflow for PyTorch model deployment on a smartphone

The main components in this activity were:

3.4.1 Model Optimization on PyTorch

Optimization of the model that was saved following the training process was necessary in order to have it run on the Android platform. The first step was to use TorchScript to transition the model from a Python program to a TorchScript program that can be executed independent of Python such as a standalone C++ program via the Android NDK. This is the Native Development Kit that allows implementation of parts of an application using native code in languages like C/C++. Specific attention was paid to the shape of the input data matrices to make sure that the Android application would process the input data into similar formats. The eventual model was saved for later inclusion into the Android application.

3.4.2 Android Application Development

The development of the Android Application was done using Android Studio 4.0, which is the official Integrated Development Environment (IDE) for Android based on IntelliJ IDEA software. The process was iteratively done by incrementally building prototypes with necessary improvements to incorporate design requirements. The initial steps involved creating skeleton sections of the main Activities (screens on the Android app User Interface) conforming to the wireframes created as part of the design process. The layout of the Activity was done in line with the wireframe components and design with inclusion of all UI components as stubs with no interactivity as this would be added in later stages. Later the functionality of these Activities was incremented over iterations to incorporate all functional and non-functional requirements.

3.4.3 Model Embedding onto the Android Application

The optimized TorchScript module was incorporated into the Android application in this step. The process involved placement of the model file into the respective *resources* folder in the application source code, and addition of the functionality to load the model and process input to perform de-identification. Owing to the arbitrary time that may be taken to de-identify input text, the de-identification logic was implemented as a Background Asynchronous Task. This enhanced the responsiveness of the application, also being in line with general Android design guidelines which recommend delegating any tasks whose execution may last more than a few milliseconds to a background task.

3.4.4 Testing and Evaluation of the Android Application

Upon completion of the mobile application development, a number of tests were formulated to ascertain functionality of the application. These tests ranged from usability tests as guided by the functional user needs as well as evaluation of the model performance in de-identification of both files containing clinical notes as well as user-provided free text. The latter was guided by the non-functional requirements defined for the system during the design process.

3.5 Research Quality

It is important for the research data and findings to be valid and reliable. Reliability entails the attainment of same findings should the research be replicated by other people. Validity on the other hand entails the degree to which the data accurately measures the intended metrics. Best practises were adopted to ensure proper documentation in all stages of the research to ensure reliability for the research. Validity can be categorised into content and predictive validity. The former relates the degree to which sufficient coverage of a subject under study is done, while in the latter predicted performance is determined by use of a test (Kothari, 2004). The emphasis on this research was on the

predictive validity where tests were formulated to evaluate the performance of the mobile application in carrying out de-identification tasks.

3.7 Ethical Considerations

The data used in this research was obtained from The Harvard Medical Schools' Department of Biomedical Informatics. The contents of the datasets are fully de-identified and replaced with realistic surrogates before being released for use by researchers. However, the datasets are not publicly available and therefore both the Data Use Agreement and NLP Research Purpose documents were reviewed and signed by the researcher before access to the dataset was granted. Utmost care was taken to ensure confidentiality of the data, as well as restriction in the use of the data for the goals of this research only. All previous works were appropriately cited, and the respective authors acknowledged. Ethical approval to carry out this research was obtained from The Strathmore University Institutional Ethics Review Committee (SU-IERC) and National Commission for Science, Technology and Innovation (NACOSTI), these being the local IRB body and the mandated government body respectively.

CHAPTER 4: SYSTEM ANALYSIS, DESIGN AND ARCHITECTURE

4.1 Introduction

This chapter covers the system design of the deep learning-based mobile PHI de-identification system in line with the third objective of this research. The architecture of the system is also outlined as per the conceptual framework presented in section 2.6. The topics covered include the analysis of both functional and non-functional requirements, the system components and the related user interactions modelled using the Unified Modelling Language (UML).

4.2 System Analysis

This section covers the analysis and requirements for the system as per the objectives set forth in this research, with the key goal being to outline what the user needs and how to achieve it. System analysis is a critical stage which answers a number of questions including: who will use the system, what the system will do, and where/when the system will be used (Ifeanyi Cosmas, 2018). Essentially, the goal is to understand and document the essential characteristics of a system to finally come up with its specification.

4.2.1 Requirement Gathering

The set of requirements for this system are guided by its definition as a Deep Learning-based System for De-identification of Personal Health Information on Smartphones. Chapters 2 and 3 of this research covered the first two research objectives namely the review of the process of de-identification of protected health information, and an appraisal of existing models, algorithms and frameworks that can be applied in de-identification of PHI. In specific, requirements for the de-identification system were derived from the general properties of de-identification systems based on deep learning models. For instance, a de-identification system must at a basic level be able to identify and remove the 18 types of protected health information data types as defined by HIPAA (Dernoncourt et al., 2016). The performance aspects of the application like execution time were contrasted with similar initiatives to use mobile applications for performing tasks based on deep learning models (Pang et al., 2019).

For mobile healthcare-focussed applications, the migration process for the model execution is especially important to ensure the memory constraints, app installer file size, execution time and app-versioning are accounted for and adequately addressed (Benedetto et al., 2018). Section 2.5 goes into detail on how PyTorch and TorchScript addressed these needs. The requirements obtained from these findings were then subjected to a comprehensive content-analysis to come up with the analysis/requirements, broadly categorized into two groups: functional and non-functional requirements.

4.2.2 Functional Requirements

These are requirements which encompass features of the system and services (Sunner and Bajaj, 2016), typically specifying a behaviour or a function. The functional requirements for this system are:

i) A user should be able to choose a textual file from local phone storage for de-identification.
ii) A user should be able to type/paste free text for de-identification
iii) A user should be able to secure the application with a 4-digit PIN
iv) The user should be able to view all completed de-identification tasks in an historical view.
v) The user should be able to perform configuration for certain application properties like data folders and notifications.
vi) The user should be able to perform de-identification of textual data using the application.
vii) Performance and responsiveness – the system should be able to de-identify text within a reasonable time via a background task to remain responsive to user input
viii) Ability to run on a smartphone with the Android Operating System

4.2.3 Non-functional Requirements

Non-functional requirements cover the emergent properties of the system, as well as any associated constraints (Sunner and Bajaj, 2016), in essence specifying how the system should work.

The non-functional requirements for this system are:

i) The user interface should be user friendly and adhere to Android design guidelines.
ii) The system should operate without crashing
iii) The system should be secure in the sense that it should not inadvertently expose user information.

The incorporation of these requirements into the de-identification system is covered under section 4.4, System Designs. They informed the total architecture of the system, as well as the human, physical and communication components, to create an overall system that primarily satisfied its essential requirements. For instance, Use Cases were drawn from the functional requirements, going a level lower to specify the low-level interactions of the user(s) and the system. Moreover, section 5.3, System Testing, references both

functional and non-functional requirements to ascertain that the system implementation indeed fulfilled the requirements set out.

4.3 System Architecture

The architecture of the mobile de-identification system is illustrated in figure 4.1. Beginning with a dataset of patient notes, pre-processing on the data is done after which it is split into a training, validation, and test sets. This data is used to train a LSTM-based neural network. The trained model parameters are then converted into a portable format enabling embedding onto an Android application. The embedded model is then used in inference mode to de-identify text in the mobile application. The user can specify a file or configure the application to automatically monitor a specified folder for file changes. Once an input file is received, a de-identification task is triggered within the application service and the de-identification task status stored in the local database. This status is continually updated until the task is complete and can be queried to update the user interface.

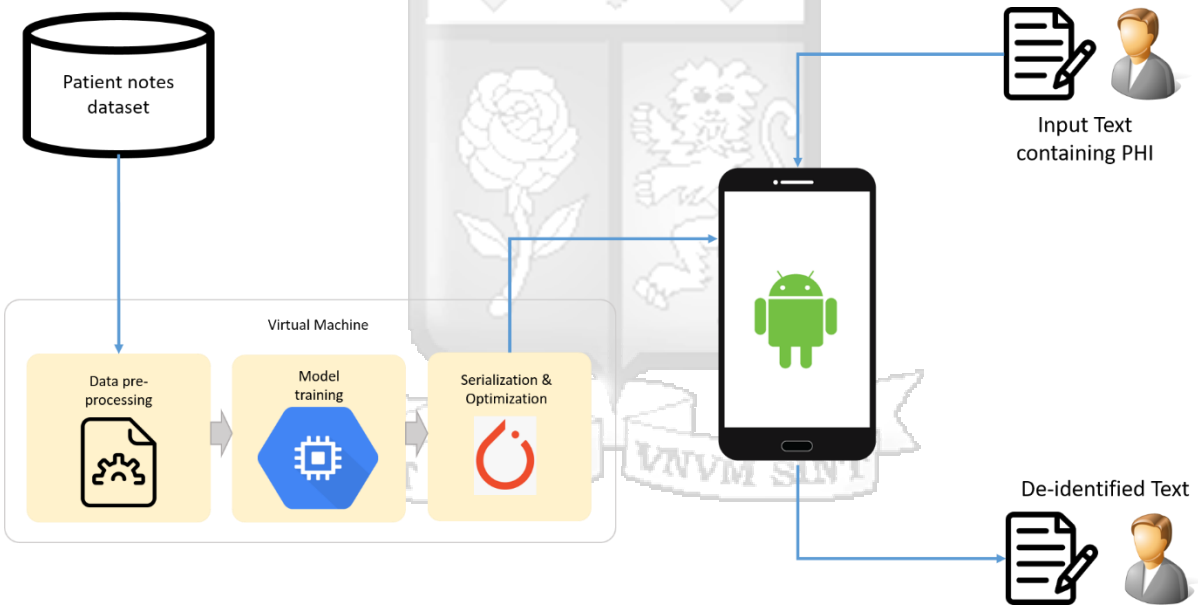


Figure 4.1: System architecture

4.4 System Designs

System design seeks to determine the total architecture of the system, incorporating all the human, physical and communication components, that will satisfy its essential requirements. The primary purpose is to successfully deliver a system that implements both the functional and non-functional requirements in an affordable and maintainable manner (Dennis et al., 2009). This research adopted the object-oriented approach, where a bottom-up view was implemented. Data and processes were combined into objects

(with the processes becoming methods). These objects represented users, tasks, etc. Diagrams/models were then used to represent the system functionalities and views, as part of the Unified Modelling Language (UML).

4.4.1 Use-Case Diagram

A Use Case represents the interactions of a system with its environment in a formal manner, depicting all the activities performed by the system users. Concretely, they describe what a system does from the standpoint of an external observer, emphasising on *what* a system does rather than *how*. Figure 4.2 shows the Use Case Diagram for the mobile de-identification system.

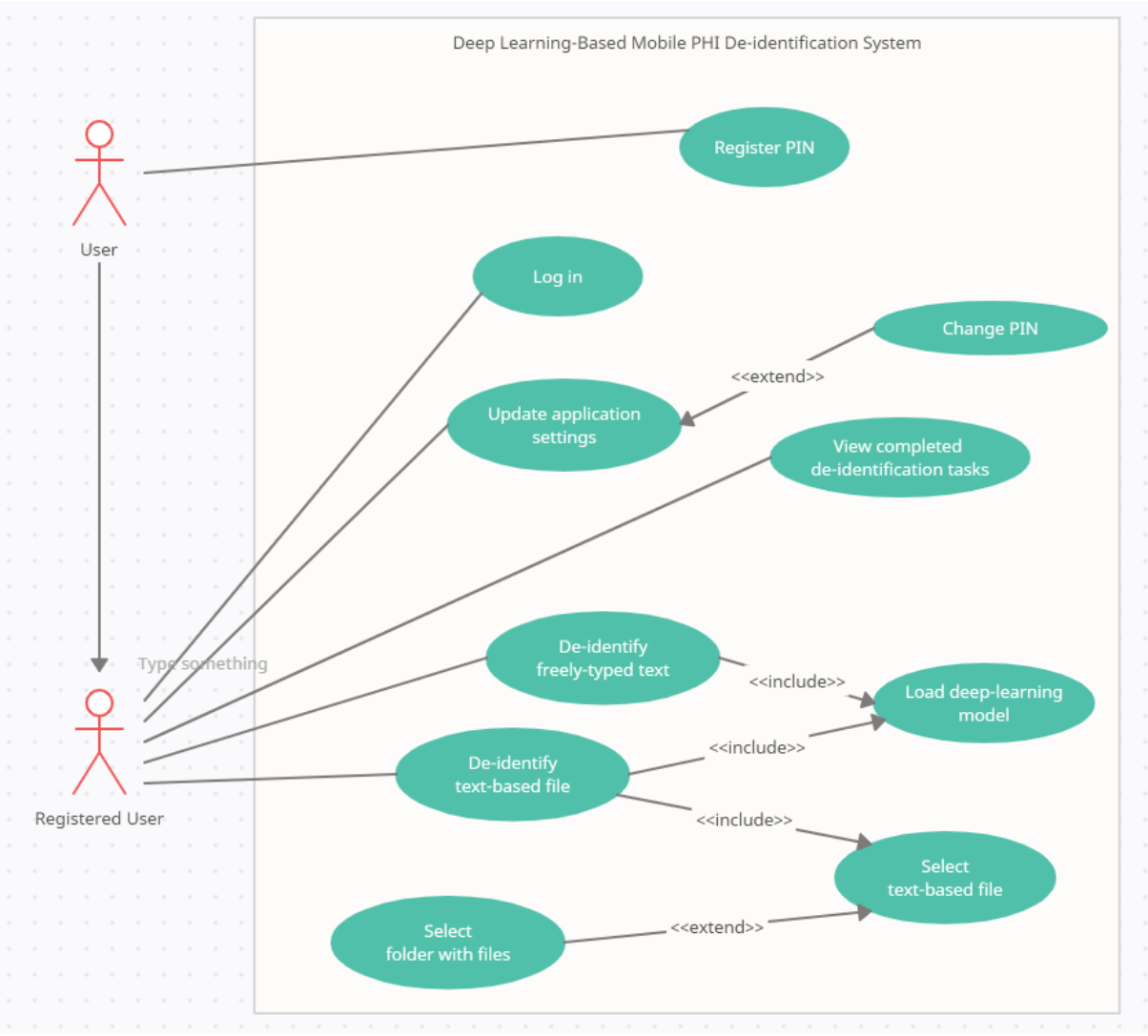


Figure 4.2: A Use Case diagram for the de-identification system

There are four main components in a Use Case: a use case task representing a feature needed in a software system; an actor(s) who triggers the use case to activate it; a communication line to illustrate how the actor(s) interact with the use case; and a system

boundary separating use cases internal to the system from actors who are external to the system.

In the *Register PIN Use Case*, the primary actor is a User whereby a 4-digit PIN is provided to register the user, then hashed and stored in the local Shared Preferences for future authentication of the user to allow access to the application functionalities by the System. Following this, a Registered User can use the same PIN to log into the application whereby the System will fetch the hashed local password and compare it to the provided PIN for authentication. The logged-in user can thereby access all the application functionalities. A Registered User can as well update the various customizations for the application as depicted in the Update Application Settings Use Case, whereby the user will specify the values to be updated and the System will store the updated values in a local Shared Preferences file. Optionally, a Registered user can change their PIN from the application settings menu.

Every de-identified file is processed as a task within the application and a list of all successful tasks is maintained within the local SQLite database. The View Completed De-Identification Tasks Use Case captures this functionality whereby the user can see a list of all completed de-identification tasks as well as a key metadata such as start and end timestamps for the task. Regarding the core functionality of the de-identification system, a Registered User can de-identify freely typed text, or text contained in a selected file/files inside a folder. In either case, these use cases include the loading of the deep learning model packaged alongside the application which does the processing to achieve de-identification.

4.4.2 Sequence Diagram

Figure 4.3 depicts the system level sequence diagram for the system. A sequence diagram depicts the interaction of components of a system, clearly indicating in a sequential manner how a system fulfils an objective. A system-level sequence diagram is drawn directly from the Use Case diagram illustrated in figure 4.2, illustrating the interactions between the actors and the system. It facilitates visualization and validation of the logic of the common scenarios in the system.

In this illustration, the steps taken to de-identify user-specified file or text using the de-identification system are clearly shown, alongside the secondary/auxiliary steps. A user, who is initially not registered initiates a registration request by providing a PIN. This is a synchronous process, as the system will wait for the PIN to be validated, hashed, and stored for future logins. The user is subsequently considered to be registered and can access the system functionality by logging in with the PIN. When the user loads a file for de-identification or enters freely typed text, the system will take up the asynchronous request and perform a series of operations in the background. These include loading the

deep learning model, using the model in inference mode to identify any PHI in the user-specified text, saving the status of the task into the local database, and finally a response to the user as a notification or the de-identified text. The user can view a list of all the completed tasks via an asynchronous call based on which the system will query the SQLite database to retrieve a list of the tasks, then format for display.

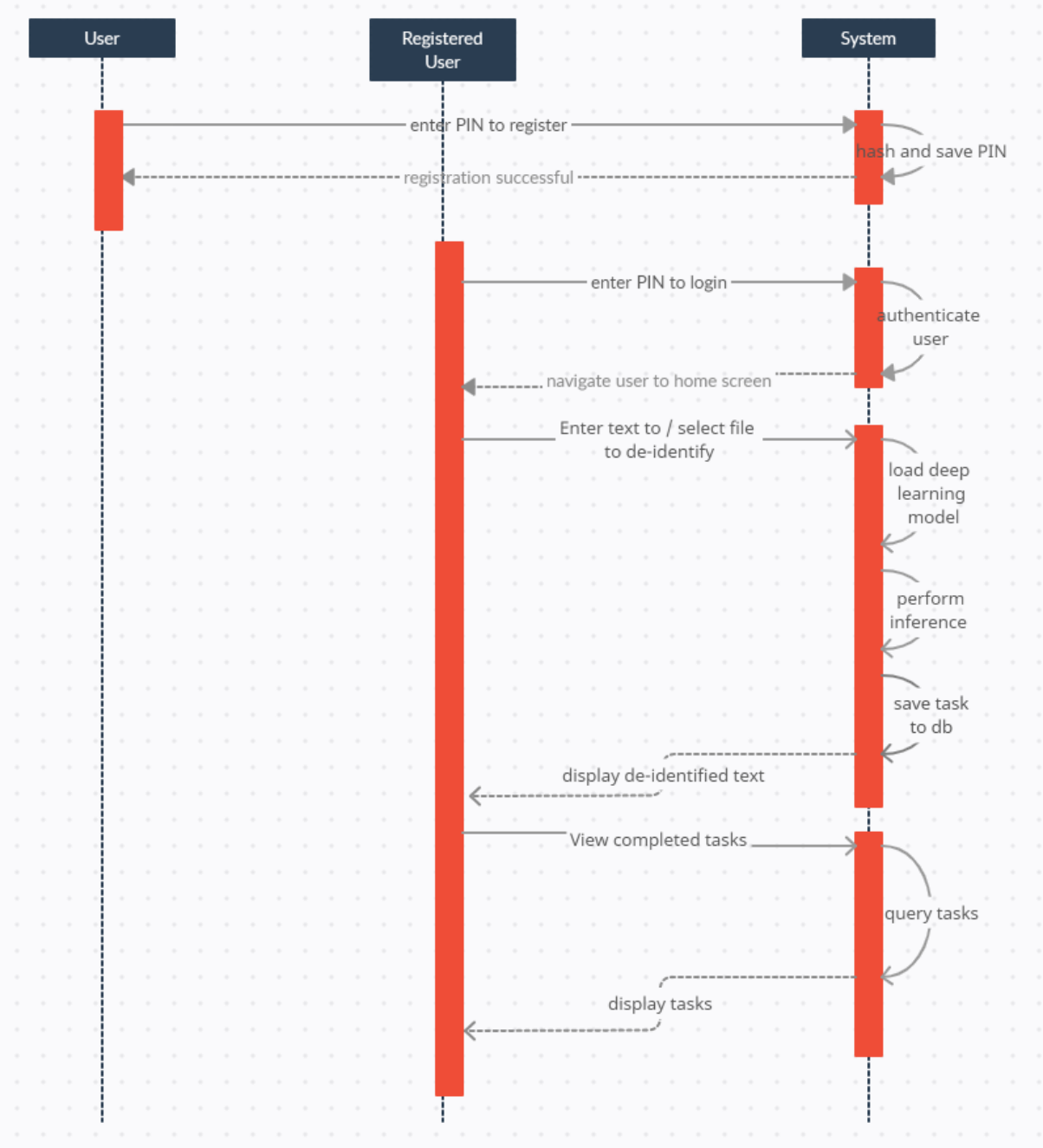


Figure 4.3: A system-level sequence diagram for the system

4.4.3 Entity Relationship Diagram

This is a diagram illustrating the information created, stored, and used in a business system, with entities being the building blocks for the data model. Entities refer to

person(s), places, events or things about which data is collected. The relationships between the entities are illustrated by use of lines. Each entity has attributes (information about the entity) and associations connect two or more properties.

In this system, the entity relationship diagram is a depicted in figure 4.4. A User has 4 attributes, namely an id, a name, a username, and a hashed password. Each user has a specific configuration, with the configuration entity having an id attribute as well as a *user_id* foreign key. The association between the two is One to One, implying a user can only have 1 configuration and vice versa. The Configuration entity stores the user preferences namely the string to replace for redacted text (*genericRedactText*) and a boolean value indicating whether to delete the original file after de-identification (*autoDeleteFiles*).

A user triggers a de-identification task, which captures the user as a foreign key, as well as that of the file being processes. The task also has attributes to capture information on task start and end date. Upon completion of the file de-identification, the *status* field captures the success/failure in a boolean field.

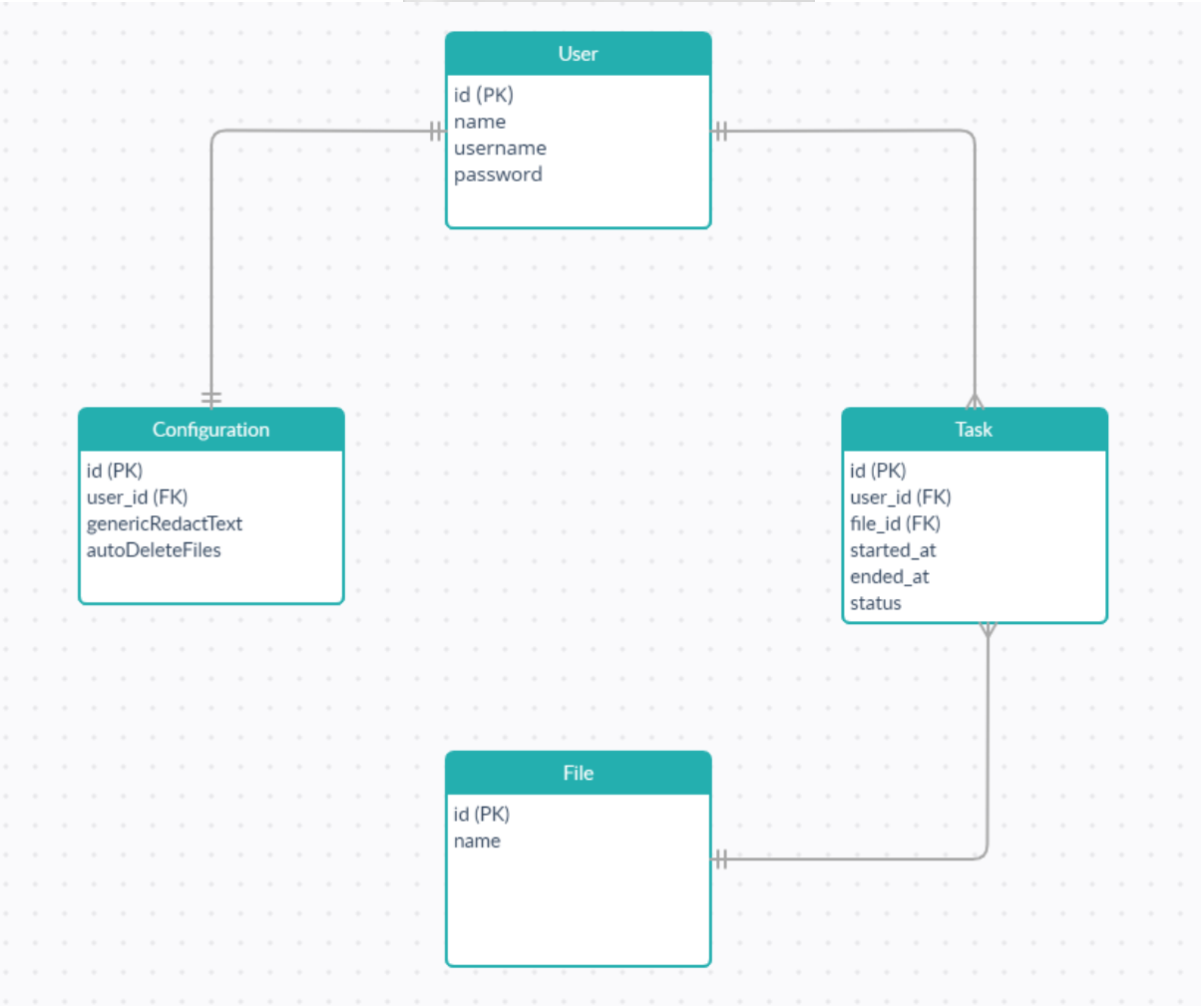


Figure 4.4: An Entity Relationship Diagram (ERD) for the system

4.4.4 Class Diagram

Class diagrams are a key component in object-oriented systems modelling, giving a high-level model of the system. They ensure the system is well understood by developers, by giving a high-level view of the system structure, which is translated into classes alongside the various interactions. As depicted in figure 4.5, a user is a central block in the system model, loading a File model for de-identification. This File is passed into a Pre-Processor class which contains parameters as attributes specifying pre-processing parameters, and a method for pre-processing the input file into a format suitable for input into the de-identification step. The Pre-processor subsequently provides parameters to the Model class as input. The Model class in essence is the de-identification core class and has methods for either training the model or running it in inference mode to de-identify an input string.

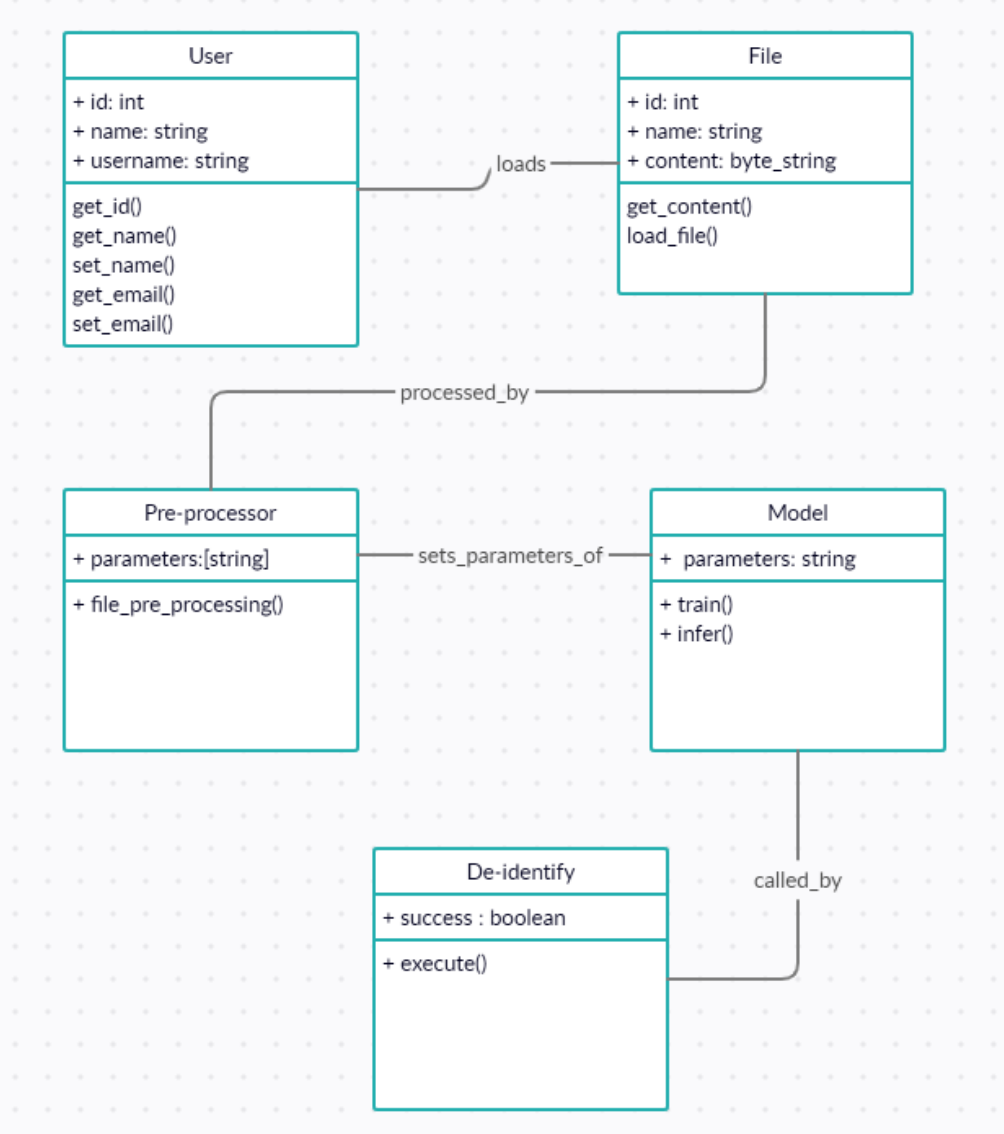


Figure 4.5: Class diagram

The Model is however not exposed directly to the user but is encapsulated in a De-identify class, with an execute method to run the model, as well as a boolean attribute to indicate whether the de-identification process was successful or not.

4.4.5 Mobile Application Wireframes

The wireframes for the mobile application are as illustrated in figure 4.6. The user interface is simple and uncluttered, presenting the user with a straightforward process for executing the desired action. A user can select a file for de-identification, following which the de-identification will happen in the background via an Android Foreground Service, after which a task item is created and stored with details of the de-identified file, as well as execution time. A user may also select the *Free Text* option from where they can freely type text for de-identification.

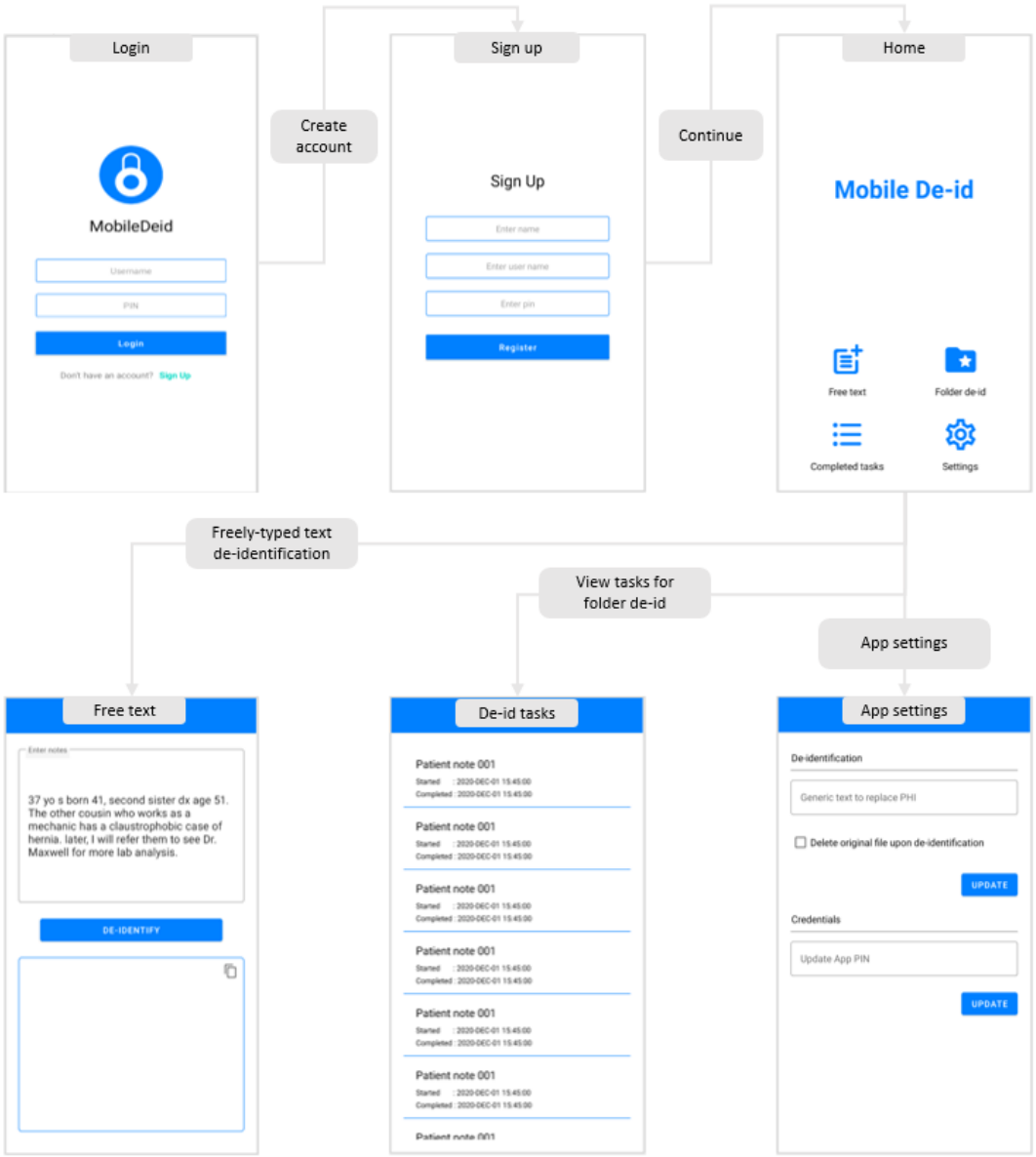


Figure 4.6: Mobile application wireframes

CHAPTER 5: SYSTEM IMPLEMENTATION AND TESTING

5.1 Introduction

The system implemented in this research was aimed at enabling de-identification of PHI on smartphones using a mobile application. This chapter outlines the implementation steps for creating the various components which make up the system. It also covers the procedures used to test these components against the desired/expected outcomes, individually and collectively.

The de-identification system was developed by training a LSTM-based neural network model with data containing classified PHI, which is then embedded into an Android application. Functional tests were then conducted to ascertain that the system components perform their functions in the manner expected. Usability tests were also carried out to determine whether intended use cases could be carried out to accomplish a de-identification task.

5.2 System Implementation

5.2.1 The Development Environment

The development environment entails the characteristics of both the hardware and software components necessary to successfully develop the system. Development work was done on the researcher's laptop running with the following hardware specifications: Intel Core i7-6280HQ; 16GB RAM, 200GB hard disk, and NVIDIA Quadro M1000M with 2GB dedicated GDDR5 video memory. The laptop was running Ubuntu 20.04.1 LTS Operating System. Model development was done in Python using the PyTorch Framework version 1.7.0, Jupyter Notebook version 6.0.3, Jupyter Core version 4.7.0, PyTorch version 1.4. Due to software version incompatibilities, specifically for PyTorch and the CUDA toolkit, it was not possible to leverage on the built-in graphics card to speed up the computations. Moreover, the model architecture had high memory requirements for training to hold and update the parameters in a timely manner. A cloud-based compute instance was used instead for model training after which the saved models and files would be exported to the laptop. For development of the mobile application, Android Studio version 4.1.2 was used.

For model training and evaluation, a Google Cloud Compute Instance with the following specifications was used: 4 x 1vCPU, 26GB RAM, 1 x NVIDIA Tesla K80 graphics card. The same software versions for PyTorch and Jupyter as listed for the laptop development environment were retained. On the cloud instance it was possible to employ the CUDA parallel computing platform and API from NVIDIA to speed up the training process via the associated GPUs on the graphic card.

5.2.2 LSTM-CRF model

The architecture of the neural network used is as illustrated in figure 5.1. It comprises of 4 main layers namely: character embedding layer; word embedding layer; bi-LSTM layer; and a final CRF layer. This follows a novel network architecture that has been shown to achieve state-of-art performance in PoS tagging and NER tasks with automatic inclusion of word and character-level representations (Ma and Hovy, 2016).

Each text input is tokenized into sentences, and further into word tokens with input processing done sentence-wise. For each word token, X_1c is the character-wise mapping and, X_1t the token embedding mapping. The token and character embedding are concatenated to form input for the LSTM layers. The output of the LSTM is fed into a CRF layer which will output sentence-level likelihoods for the optimal tag sequences.

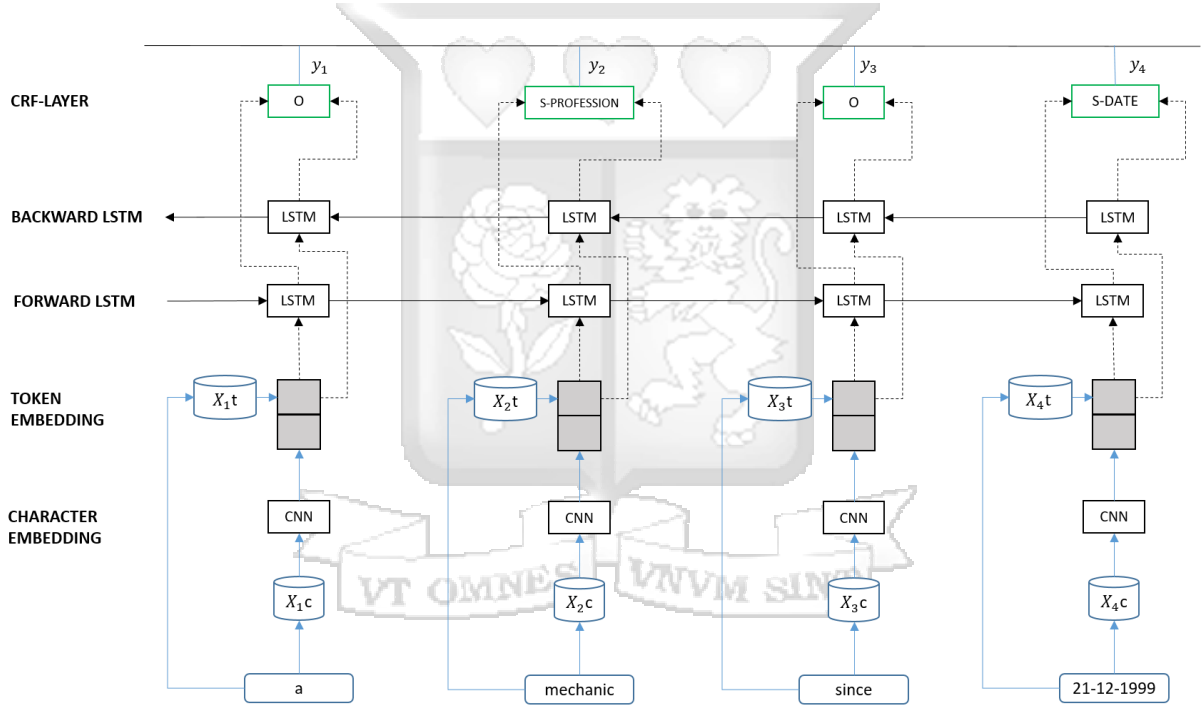


Figure 5.1: Architecture of the neural network model

Token and Character Embedding Layer

For each token in the input sentence, both the character-level and word embedding are obtained, then concatenated to form the input into the LSTM layers of the model. This is done to account for the fact that data scarcity may affect the token embedding if used alone, for example due to misspellings in the tokens, tokens not found in the vocabulary, and different endings for the verbs (Dernoncourt et al., 2016). Research has shown that Convolutional Neural Networks can be used to effectively extract morphological information from characters in a token and encode this in appropriate representations (Ma and Hovy, 2016). The characters in each token are therefore mapped to numerical

values, then fed into a CNN model. In this model, a convolutional layer will produce the spatial mapping across the characters, which is then pooled to extract the meaningful features from the representation. This forms the output of the character embedding layer.

For the token embedding, the GloVe 100-dimensional embeddings were used for parameter initialization for the token embedding layer. These embeddings have been pre-trained on 6 billion words drawn from the web and Wikipedia (Pennington et al., 2014).

LSTM Layer

The output of both the character embedding and token embedding layer is concatenated and forms input for the LSTM layer, as illustrated in figure 5.2. Specifically, a bidirectional LSTM model is employed which has two layers (forward and backwards). The forward layer takes in token vector sequence and generates a new vector based on the current and previous input in the forward direction. Likewise, the backwards layer will generate an output vector based on the current input as well as the text to the end of the sentence. These two vectors (from the forward and backwards layers) are concatenated to form the unified representation of the LSTM output.

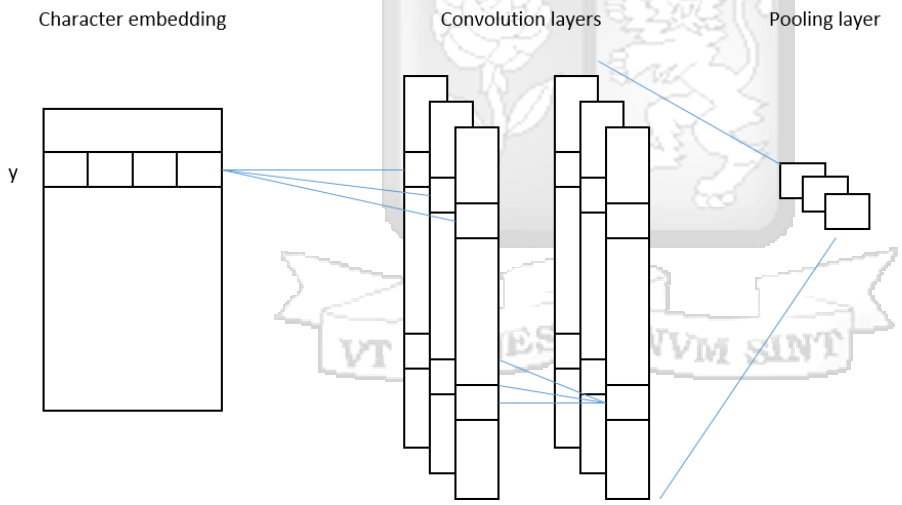


Figure 5.2: The CNN model used for character-level embedding

Finally, this unified output is fed into a linear layer which serves to map the output of the LSTM layers to tag space, having the dimensions of the (LSTM hidden dimensions * 2) and the number of tags.

CRF Layer

There are two ways to process the output of the LSTM output. We could use the Softmax function, which normalizes the output into some vector. This vector is subsequently interpreted as the probability that a token belongs to one of the classes in the tag classes. This probability of a particular sequence of the tags is therefore the product of all the tags.

Unfortunately, Softmax does not consider the dependencies between tokens. In Named-Entity Recognition, the context will significantly influence the tag that is assigned. To solve this challenge, CRF was applied to jointly decode the output as research has demonstrated this to offer superior performance over other implementations (Ma and Hovy, 2016). The final model architecture is as illustrated in figure 5.3 below.

```
In [103]: if params['skip_training']:
              model.load_state_dict(torch.load(params['model_path']))
              print("model reloaded :", parameters['model_path'])

          model.cuda()

Out[103]: CNN_BiLSTM_CRF(
            (char_embeds): Embedding(94, 25)
            (char_cnn3): Conv2d(1, 25, kernel_size=(3, 25), stride=(1, 1), padding=(2, 0))
            (word_embeds): Embedding(36669, 100)
            (dropout): Dropout(p=0.5, inplace=False)
            (lstm): LSTM(125, 200, bidirectional=True)
            (hidden2tag): Linear(in_features=400, out_features=82, bias=True)
          )
```

Figure 5.3: The final model architecture of the deep-learning model with the various layers

5.2.3 Dataset Collection

In this research, the medical records the 2014 i2b2 Track 1 dataset was used. For access, the researcher agreed to and signed a Data Use Agreement, while outlining their purpose in a Research Purpose document. The dataset is presently available at https://portal.dbmi.hms.harvard.edu/

5.2.4 Data Pre-processing

In the original format, the dataset is organized as follows: PHI Gold Set 1 for training, PHI Gold Set 2 for Training, PHI Gold Set-Fixed for testing. Each of these categories comprise of a compressed list of clinical notes in xml format. The schema of each xml file is as illustrated below.

```
<deIdi2b2>
        <TEXT>…</TEXT>
        <TAGS>
                <DATE id="" start="" end="" text="" TYPE="" comment=""/>
        </TAGS>
</ deIdi2b2>
```

Data pre-processing was therefore required for extracting the clinical notes from the files along with the annotations for the different entities. Next, tagging was done on the data to tag every word in the text to a particular entity in the PHI entities. Specifically, the data was processed to the BIOES tagging scheme as it has been found to offer much more meaningful results over other tagging schemes (Lample et al., 2016). To achieve this, every token in the sentences was annotated as B-label if at the beginning of a named entity, an I-label if inside a named entity and otherwise annotated as O. Additionally,

single entities were annotated as I-labels and tokens at the end of named entities as E-label.

The total dataset was found to have 1057722 words with 36669 being unique, 94 unique characters and 82 named entity tags as per the BIOES tagging scheme as illustrated in figure 5.4. Finally, the dataset was split into training and testing datasets with an 80:20 split. The training dataset was further split into training and validation datasets in an 80:20 split as well. The final training, validation and testing datasets had 85111, 21278, and 26958 sentences respectively.

```
In [51]: dico_words,word_to_id,id_to_word = word_mapping(master_sentence_list)
         dico_chars, char_to_id, id_to_char = char_mapping(master_sentence_list)
         dico_tags, tag_to_id, id_to_tag = tag_mapping(master_sentence_list)

         Found 36669 unique words (1057722 in total)
         Found 94 unique characters
         Found 82 unique named entity tags

In [96]: def prepare_dataset(sentences, word_to_id, char_to_id, tag_to_id):
             """
             Prepare the dataset. Return a list of lists of dictionaries containing:
                 - word indexes
                 - word char indexes
                 - tag indexes
             """
             data = []
             for s in sentences:
                 str_words = [w[0] for w in s]
                 words = [word_to_id[lower_case(w) if lower_case(w) in word_to_id else '<UNK>']
                          for w in str_words]
                 # Skip characters that are not in the training set
                 chars = [[char_to_id[c] for c in w if c in char_to_id] for w in str_words]
                 tags = [tag_to_id[w[-1]] for w in s]
                 data.append({
                     'str_words': str_words,
                     'words': words,
                     'chars': chars,
                     'tags': tags,
                 })
             return data

         master_data = prepare_dataset(master_sentence_list, word_to_id, char_to_id, tag_to_id)

In [97]: print(len(master_data))
         train_data, test_data = train_test_split(master_data,test_size=0.2)
         train_data, val_data = train_test_split(train_data,test_size=0.2)
         print("{} / {} / {} sentences in train / dev / test.".format(len(train_data), len(val_data), len(test_data)))

         132987
         85111 / 21278 / 26598 sentences in train / dev / test.

In [98]: master_word_embeds = {}
         for i, line in enumerate(codecs.open(params['glove_embedding_path'], 'r', 'utf-8')):
             s = line.strip().split()
```

Figure 5.4: Sample data pre-processing steps carried out

5.2.5 Model Training

The LSTM-CRF model was trained using the PyTorch framework. The word embeddings were initialized using Stanford's GloVe 100-dimensional embeddings. After initial experimentation with values of 0.005, 0.01 and 0.015, the best learning rate was found out to be 0.015. Gradient clipping helps address the challenge of exploding gradients, while momentum helps accelerate convergence. Both the two parameters were set at 5.0 and 0.9 respectively. The optimization algorithm used was Stochastic Gradient Descent (SGD). Initially, model performance was evaluated with a few iterations over one epoch. This was done to ascertain that the losses were decreasing and to fine-tune the hyperparameters. A snapshot of these is illustrated in figure 5.5.

```
In [105]: learning_rate = 0.015
momentum = 0.9
number_of_epochs = params['epochs']
decay_rate = 0.05
gradient_clip = params['gradient_clip']
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)

losses = [] #list to store all losses
loss = 0.0 #Loss initialization
val_losses = [] #list to store all validation losses
best_dev_F = -1.0 # Current best F-1 Score on Dev Set
best_test_F = -1.0 # Current best F-1 Score on Test Set
best_train_F = -1.0 # Current best F-1 Score on Train Set
all_F = [[0, 0, 0]] # List storing all the F-1 Scores
eval_every = len(train_data) # Calculate F-1 Score after this many iterations
plot_every = 4000 # Store loss after this many iterations
count = 0 #Counts the number of iterations
```

```
In [106]: if not params['skip_training']:
    tr = time.time()
    model.train(True)
    for epoch in range(1,number_of_epochs):
        for i, index in enumerate(np.random.permutation(len(train_data))):
            count += 1
            data = train_data[index]

            ##gradient updates for each data entry
            model.zero_grad()

            sentence_in = data['words']
            sentence_in = Variable(torch.LongTensor(sentence_in))
            tags = data['tags']
            chars2 = data['chars']

            ## Padding the each word to max word size of that sentence
            chars2_length = [len(c) for c in chars2]
            char_maxl = max(chars2_length)
            chars2_mask = np.zeros((len(chars2_length), char_maxl), dtype='int')
            for i, c in enumerate(chars2):
                chars2_mask[i, :chars2_length[i]] = c
            chars2_mask = Variable(torch.LongTensor(chars2_mask))

            targets = torch.LongTensor(tags)

            #we calculate the negative log-likelihood for the predicted tags using the predefined function
            neg_log_likelihood = model.neg_log_likelihood(sentence_in.cuda(), targets.cuda(),
```

Figure 5.5: Training steps excerpt for the deep learning model

Training was done over 40 epochs, with loss calculation and evaluation done every 4000 iterations. As illustrated in figure 5.6, both training and validation loss decreased over the iterations and epochs. If the validation loss was less than the previous value, a snapshot of the model was saved or updated.

```
                adjust_learning_rate(optimizer, lr=learning_rate/(1+decay_rate*count/len(train_data)))

    print(time.time() - tr)
    plt.plot(losses)
    plt.show()

if not parameters['reload']:
    #reload the best model saved from training
    model.load_state_dict(torch.load(model_name))
```

```
r(0.0307, device='cuda:0', dtype=torch.float64)
Count :  984000  Training loss :  tensor(0.0145, device='cuda:0', dtype=torch.float64)  Validation loss :  tenso
r(0.0328, device='cuda:0', dtype=torch.float64)
Count :  988000  Training loss :  tensor(0.0124, device='cuda:0', dtype=torch.float64)  Validation loss :  tenso
r(0.0330, device='cuda:0', dtype=torch.float64)
Count :  992000  Training loss :  tensor(0.0135, device='cuda:0', dtype=torch.float64)  Validation loss :  tenso
r(0.0330, device='cuda:0', dtype=torch.float64)
Count :  996000  Training loss :  tensor(0.0129, device='cuda:0', dtype=torch.float64)  Validation loss :  tenso
r(0.0306, device='cuda:0', dtype=torch.float64)
Count :  1000000  Training loss :  tensor(0.0149, device='cuda:0', dtype=torch.float64)  Validation loss :  tens
or(0.0311, device='cuda:0', dtype=torch.float64)
Count :  1004000  Training loss :  tensor(0.0132, device='cuda:0', dtype=torch.float64)  Validation loss :  tens
or(0.0309, device='cuda:0', dtype=torch.float64)
Count :  1008000  Training loss :  tensor(0.0159, device='cuda:0', dtype=torch.float64)  Validation loss :  tens
or(0.0292, device='cuda:0', dtype=torch.float64)
Count :  1012000  Training loss :  tensor(0.0205, device='cuda:0', dtype=torch.float64)  Validation loss :  tens
or(0.0294, device='cuda:0', dtype=torch.float64)
Count :  1016000  Training loss :  tensor(0.0153, device='cuda:0', dtype=torch.float64)  Validation loss :  tens
or(0.0334, device='cuda:0', dtype=torch.float64)
Count :  1020000  Training loss :  tensor(0.0112, device='cuda:0', dtype=torch.float64)  Validation loss :  tens
```

```
In [107]: plt.plot(losses, label="training loss")
```

Figure 5.6: Training and validation dataset loss output for every 4000th iteration

Figure 5.7 illustrates the Loss(Y-axis) plotted against the number of iterations(X-axis). The training process took 11 hours on a Google Cloud Compute instance with the specifications described in section 5.2. At some point in the training process, the validation loss began increasing with decreasing training loss. This was indicative of the model beginning to over-fit the data and subsequently training was stopped in subsequent training processes before the 40 epochs were completed. The saved model for Android integration was already designed to be the snapshot with the lowest validation loss.

```
In [107]: plt.plot(losses, label="training loss")
          plt.plot(val_losses, label="validation loss")
          plt.title('Loss')
          plt.legend()
          plt.show()
```

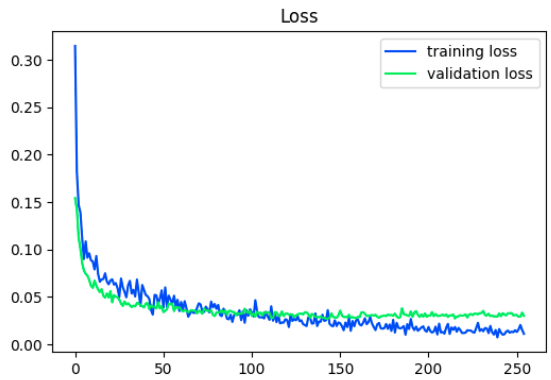


Figure 5.7: The validation loss plotted against no. of iterations

To enable the model to be deployed on Android as described in 5.2.6, all the different mappings used in the data pre-processing and training process were exported to JSON format to enable them to be used on the mobile application to perform inference. Figure 5.8 illustrates a code snippet that was used to achieve this. These mappings are word-to-id mappings, character-to-id mappings, and the class-to-id mapping. In Python code, these were simple lookup dictionaries mapping out the words, classes and characters and words in the vocabulary set to numeric ids. In JSON these were mapped to lists of objects where look-up could be easily done in Java code on Android.

```
In [99]: with open(params['data_mapping_path'], 'wb') as f:
             mappings = {
                 'word_to_id': word_to_id,
                 'tag_to_id': tag_to_id,
                 'char_to_id': char_to_id,
                 'parameters': params,
                 'word_embeds': word_embeds
             }
             cPickle.dump(mappings, f)

         print('word_to_id: ', len(word_to_id))

         word_to_id:  36669
```

Figure 5.8: The different mappings saved for JSON conversion

An *assets* directory was created on the Android Project where these JSON mappings were placed. The three files were *char_to_id.json*, *tag_to_id.json* and *word_to_id.json*. A visualization of this directory is illustrated in figure 5.9.

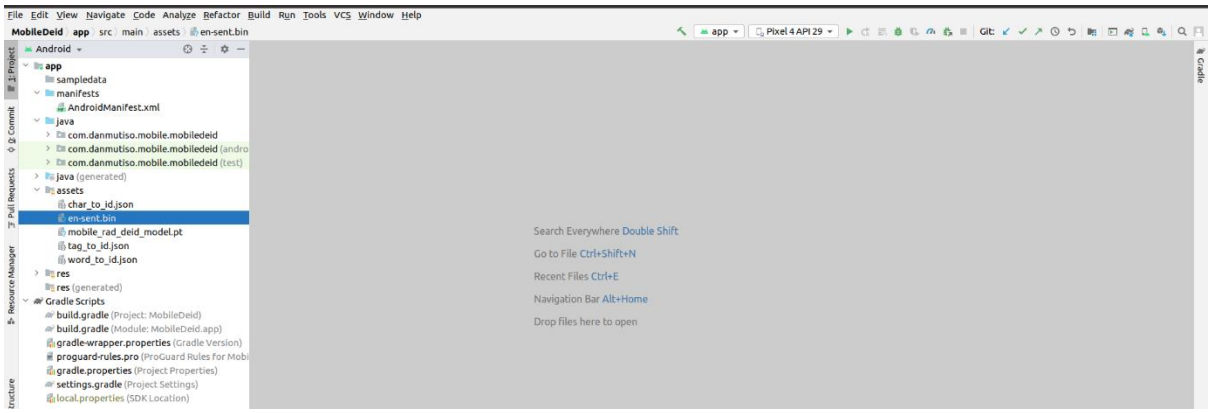


Figure 5.9: The mapping files from data pre-processing included in the Android project

5.2.6 Converting the LSTM Model into Android-compatible format

Converting the saved PyTorch model (which must be a subclass of PyTorch's nn.Module) requires the use of TorchScript, which is an intermediate representation of a PyTorch model which can be run in a high-performance environment such as C++. On Android this is supported via the Native Development Kit (NDK) which allows C++ components to be incorporated into the Android application. TorchScript code can then be invoked on its own interpreter, a form of a restricted Python interpreter.

There are two approaches for this conversion. The first option is to perform tracing of the model, using *torch.jit.trace* function of TorchScript. Here, both the PyTorch model and sample input are provided as input arguments. The input is then fed through the model as is the case during normal inference and all the operations executed are traced and stored on TorchScript. Unfortunately, this method failed to work in this research since the input during normal inference is variable, because different tokens will have different lengths. Since tracing freezes the logical structure and expects the same input dimensions every time the model is running, this approach was inapplicable and subsequently dropped.

The second approach which was adopted in this research is scripting the module, using TorchScript's *torch.jit.script* function. Here, only the model is provided as a parameter input, and TorchScript is generated through static inspection of the model contents based on all called functions. The disadvantage over scripting is the slightly larger model size resulting from the scripting process compared to tracing.

The commands executed to complete the scripting process are illustrated in figure 5.10.

```
# serialize the model for Android
import torch
import torchvision


cpu_model = model.cpu()
model.eval() # put model in eval mode
traced_script_module = torch.jit.script(model)
traced_script_module.save("mobile_cpu_deid_model.pt")
```

Figure 5.10: The PyTorch scripting process carried out for model conversion

Since model training was done on a Cloud Compute instance with a GPU attached, most functions and Tensors in the model had been updated to set the *use_gpu* flag to True to speed up the training process. It was necessary to create a *set_cpu_mode*() method in the model to set the GPU flag to false when in inference, since on Android only the CPU mode is available.

5.2.7 Building the Android Application

The Android Application was developed on Android Studio 4.1.1. As outlined in section 4.4, this activity comprised of building the layout files for the UI in xml along with the corresponding activities and services in Java. The TorchScript model generated from section 5.2.6 was copied to the assets folder as well, alongside the JSON data mapping files. The overall structure of the project on Android Studio is as illustrated in figure 5.11. 5 sub-directories namely *db*, *mdagger*, *services*, *ui* and *utils* were added whose functions are detailed as follows:

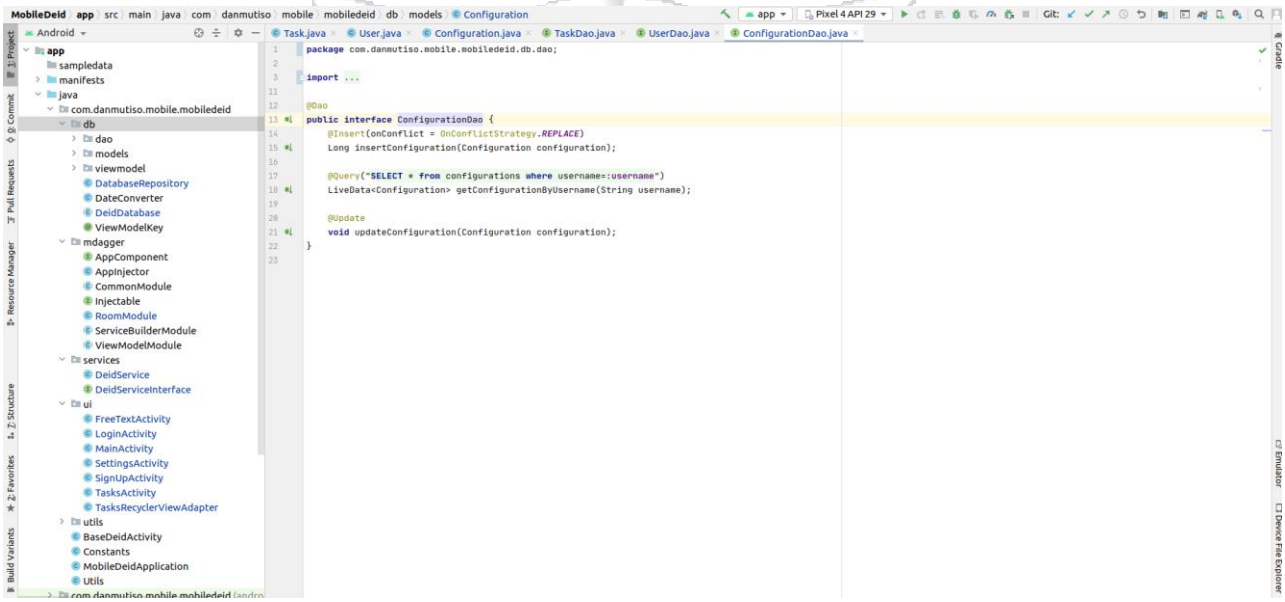


Figure 5.11: The PyTorch scripting process carried out for model conversion

The *db* directory contains all the database classes in line with the ER diagram illustrated in figure 4.4. The DAO (Database Access Object) interfaces in line with the architecture guidelines for Android Room package as well as the database declaration itself are contained in this directory as well. The *mdagger* directory contains all the Dagger2 class declarations for automatic dependency injection to address the challenge with object instantiation in the project. This is based on the Dagger2 Dependency Injection library which was incorporated to provide Dependency Injection functionality. This was an especially important component as it was necessary to declare some classes as Singletons to avoid multiple instances which would lead to state inconsistencies in the application as well as memory leakages. Some examples of such classes were the Room class instantiating the database, the *DatabaseViewModel* class for separation of UI logic from database business logic in an asynchronous manner among others.

The services directory holds the foreground service *DeidService* class via which the actual de-identification happens. This class contains methods for pre-processing input text, calling the embedded deep learning model via the NDK APIs and posting the outcome asynchronously to the UI thread. In the Android ecosystem, a service is an application component that can perform long-running tasks in the background. Two types are defined: Fore ground services and background services. A foreground service was chosen for this application because it is visible to the user via a notification and avoids the strict restrictions the system places on background services when the app itself is not on the foreground. Finally, an apk file was built and installed on to both virtual and actual Android devices for testing.

5.3 System Testing

Testing was carried out on both the neural network model as well as the mobile application as outlined in this section.

5.3.1 LSTM model Validation and Testing

Model validation and testing was done on every 4000th iteration during training as described in section 5.2.5. The best scores over 50 epochs were an F-score of 0.932 for the validation dataset as illustrated in figure 6.1. The model was also validated against the test dataset yielding a F-score of 0.943. A snapshot of the model at that stage was saved for later conversion and transfer onto the Android Application. Indeed, the negative log likelihood dropped with the number of iterations, indicating that the model was learning. The training and validation loss curves also tightly match one another indicating that the model was not overfitting either in figure 5.7. The model was tested with sample input sentences characteristic of text found in clinical notes and offered considerably similar performance. Figure 5.12 below shows the performance with a sample sentence provided to the model with the input words matched to the tags predicted by the model.

```
In [122]: model_testing_sentence = '37 yo s born 41, second sister dx age 51 Refer. to Dr Collins for follow up'
s=model_testing_sentence.lower().split()
str_words = [w for w in s]
words = [word_to_id[lower_case(w) if lower_case(w) in word_to_id else '<UNK>'] for w in str_words]
chars = [[char_to_id[c] for c in w if c in char_to_id] for w in str_words]
processed_sentence = {'str_words': str_words,'words': words,'chars': chars}

predictions = []
print(f"{'WORD':<10} {'TAG':<10}")
chars_length = [len(c) for c in chars]
char_maxl = max(chars_length)
chars_mask = np.zeros((len(chars_length), char_maxl), dtype='int')
for i, c in enumerate(chars):
    chars_mask[i, :chars_length[i]] = c
chars_mask = Variable(torch.LongTensor(chars_mask))
dwords = Variable(torch.LongTensor(words))
val,predicted_id = model(dwords.cuda(), chars_mask.cuda(), chars_length)
pred_chunks = get_chunks([ghg.item() for ghg in predicted_id],tag_to_id)
temp_list_tags=['NA']*len(words)
for p in pred_chunks:
    temp_list_tags[p[1]]=p[0]
for word,tag in zip(words,temp_list_tags):
    print(f"{id_to_word[word]:<10} {tag:<10}")
print('\n')
```

```
WORD       TAG
37         AGE
yo         NA
s          NA
born       NA
<UNK>      AGE
second     NA
sister     NA
dx         NA
age        NA
51         AGE
<UNK>      NA
to         NA
dr         NA
collins    DOCTOR
for        NA
follow     NA
up         NA
```

Figure 5.12: Model predictions with random test data

5.3.2 PHI de-identification on the Mobile Application

The mobile application apk generated in section 5.2.7 was deployed to an actual Android device by both typing free-text input for sample clinical notes, as well as running the de-identification service on files contained in the devices' storage. The application was secured with a 4-digit numeric PIN through which authenticated users could access the functionality of the application, in line with the functional requirements. A user could enter text to be de-identified though either typing in free text or selecting a file containing text. Another way a user could invoke the de-identification functionality was through specifying a folder on the smartphone which contained textual files. The de-identification system was able to detect PHI tokens from the user-provided text in all 3 scenarios and remove them, by replacing them with a generic user-configured string as illustrated in figure 5.13.

Configurability of the application was also possible, through which the user could change their PIN, alter the generic string to replace redacted text, and as well to specify the default behaviour of the application with regards to the input files post de-identification. A user could choose whether to leave the original files intact, or whether to remove them once the de-identification was completed. Performance-wise, the application was able to carry out the de-identification processes while remaining responsive to the users as the core functionality was implemented via a background service, which only interacted with the UI thread of the application once all processing was complete. Since no external communication was done with any services whether internal or external, security was guaranteed in terms of inadvertent exposure of user data.

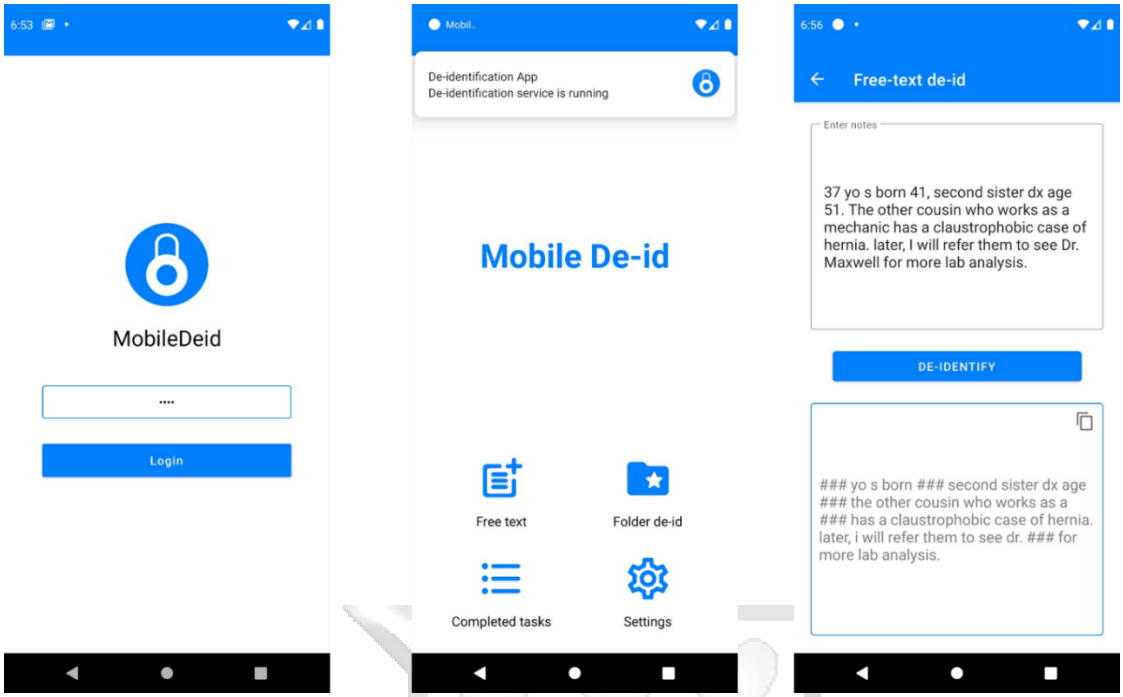


Figure 5.13: De-identification of user-provided free text on the mobile application.

The service which runs on the application to de-identify any text files in a user-specified directory was tested as well. Here, a user provides a folder from their storage and the service will traverse the directory in the background, open any text files and process the text to de-identify any tokens classified as PHI. Figure 5.14 illustrates this process in action. Successfully de-identified files are recorded as a task with start and completion times.

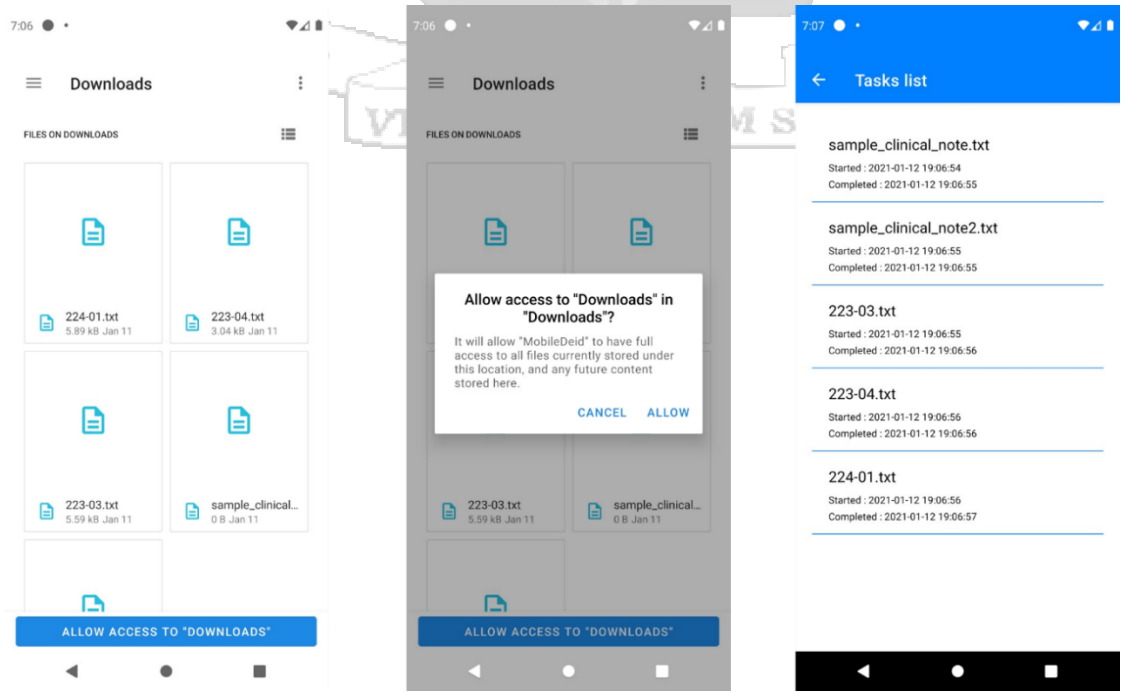


Figure 5.14: De-identification of PHI in user-specified folders by a background service

CHAPTER 6: DISCUSSIONS

6.1 Introduction

The findings of this research are discussed in this chapter along with a review of the developed system to ascertain whether the objectives of the research were accomplished. In this research, de-identification as a Named Entity Recognition problem based on deep learning networks was explored with an aim of developing a PHI de-identification model that can be trained and embedded onto a mobile application. Subsequently, and Android application was developed with the capability to de-identify sensitive data on text typed by the user or contained in textual files within the device, based on the trained deep learning model.

1. To review the process of de-identification of protected health information
2. To appraise existing models, algorithms and frameworks that can be applied in de-identification of PHI
3. To design and develop a deep learning-based de-identification model that can be embedded into a mobile application
4. To test the developed de-identification system

6.2 The process of de-identification of PHI

PHI consists of any information through which an individual patient can be identified and is used/disclosed during care provision. As covered in section 2.3, de-identification involves the removal of all PHI elements from a dataset such that a patient cannot be identified again. The two techniques used to achieve this are the Expert Determination method and the Safe Harbour Technique. Expert Determination achieves de-identification through the examination of the data by an expert to determine that the risk of the re-identification of the data via general statistical, scientific, and mathematical concepts is very low. The Safe Harbour technique on the other hand involves the removal of all PHI tags which can identify persons alone or in combination with other information. In either case, the risk of re-identification must be very low.

The Safe Harbor de-identification technique was adopted in this research based on which a deep learning model was developed to identify PHI tags in the provided text. In this implementation, the task was essentially Named-Entity Recognition (NER) focussing on identifying, classifying and removal of PHI rather than simple semantic categories as is the case in general NER implementations. The model was able to systematically map an input word sequence to pre-defined labels corresponding to any of the PHI categories to

determine whether it should be removed to achieve de-identification or not. The process is indeed like human/manual de-identification where human annotators label the data to identify and remove any PHI data albeit with much lower sensitivity compared to automated systems based on deep learning models as covered in sections 2.3.1 through 2.3.3.

The F-score of 0.943 attained eventually on the test dataset demonstrated that the model can indeed apply the principle behind the Safe Harbor technique to de-identify data via removal of PHI elements. Determining whether these results offer a sufficient performance level is a question of a careful balance between the compute/human requires necessary to achieve the highest level of privacy and the risk of re-identification (Hartman, 2020). If an organization judges the performance as insufficient, automated de-identification systems can still be used to provide an initial filter layer to reduce the PHI encountered by human annotators for security purposes. Automated de-identification systems can also indeed perform initial de-identification via the Safe Harbor principle with De-identification Experts further reviewing the data via a subsequent Expert Determination step (Hartman, 2020). This can indeed explain the adoption of automated de-identification systems based on deep learning models in that although perfect Precision and Recall performance may be difficult to achieve, a subsequent de-identification expert review step can catch any missed PHI tokens. Other organizations may deem the risk of re-identification to be low enough to deploy off-the-shelf de-identification systems without further review.

6.3 Models, algorithms, and frameworks for PHI de-identification

De-identification systems based on deep learning approaches offer superior performance and generalization over both rule-based and manual approaches as discussed in section 2.3 and subsequent sub-sections. By solving the de-identification challenge as a Named Entity Recognition task, various deep learning neural network architectures have been proposed and successfully applied to create de-identification systems among other NLP systems. In this research, the de-identification model adopted a CNN-LSTM-CRF architecture outlined in section 2.4 and subsequent sub-sections.

With the CNN-LSTM-CRF architecture, a F-score of 0.943 was achieved on the i2b2 de-identification dataset. While Dernoncourt et al achieved state-of-art F-score of 0.978 on the same dataset, the architecture comprised of 25-dimension character embedding layer, 25-dimension character-based token-embedding LSTM layer, a 100-dimension token embedding layer and a label prediction 100-dimension layer (Dernoncourt et al., 2016). The CNN-LSTM-CRF architecture on the other hand allowed for a simpler model comprising mainly of a 25-dimension CNN (kernel size 3) for character level encoding and a 200-state dimension bidirectional LSTM with a final CRF joint decoding layer. While the performance of this model does not match state-of-art, it nevertheless results

in a smaller model output with fewer weights and therefore suitable for export onto the mobile application. The use of a CNN for character-level encoding compared to a LSTM for the system with state—of-art performance contributes indeed to a lighter architecture at the expense of better performance offered by LSTMs in capturing long term dependencies among words and characters in input sequences.

6.4 A de-identification model for embedding into a mobile application

The model trained in this research adopted a CNN-LSTM-CRF architecture, with the aim of being able to identify and correctly classify whether tokens in input text belong to any of the PHI tags and subsequently de-identify the text by replacing these tags with generic text to show redaction. The researcher adopted a dataset comprising manually annotated 1,304 medical records of 297 patients. The training set, drawn from PHI Gold Set 1 and PHI Gold Set 2, contains 17045 PHI instances while the test set contains 11462 PHI instances covering 25 PHI sub-categories covering all the defined HIPAA PHI categories. This dataset was aggregated and split into an 80:20 split to form training and testing datasets, with the training dataset further split in an 80:20 proportion to obtain training and validation datasets. The testing dataset was used to predict the real-world performance of the dataset and was never used in the training or validation processes.

```
In [109]: print(best_dev_F)

          model.train(False)
          best_test_F, new_test_F, _ = evaluating(model, test_data, best_test_F,"Test")

          print(new_test_F)

          0.9322761810373307
          Test: new_F: 0.9428644632334101 best_F: -1.0
          0.9428644632334101
```

Figure 6.1: Model performance over the validation and test datasets

As illustrated in figure 6.1, the model achieved an F-Score of 0.943 which is representative of how the model would perform on data not previously seen by the model. The free-form nature of clinical notes also meant that the model would equally perform well in other forms of user-typed text which may have amorphous formats, as demonstrated in figure 5.13. This performance was achieved over 10 epochs. In line with the research objective, the researcher subsequently optimized the model for export and incorporation in the Android mobile application.

Validation of the PHI classifier was done with the test data, which was 20% of the original aggregated data not used/seen by the model during the training process. A confusion matrix was used to better evaluate the performance of the classification model as illustrated in figure 6.2. This is a n x n matrix where n represents the number of target classes with the rows representing the predicted classes and the columns the actual values of the classes. For each prediction, the value of the element representing the

intersection between the predicted class and the correct class was incremented by 1 for all the words in the test data. The values were then normalized to come up with the confusion matrix.

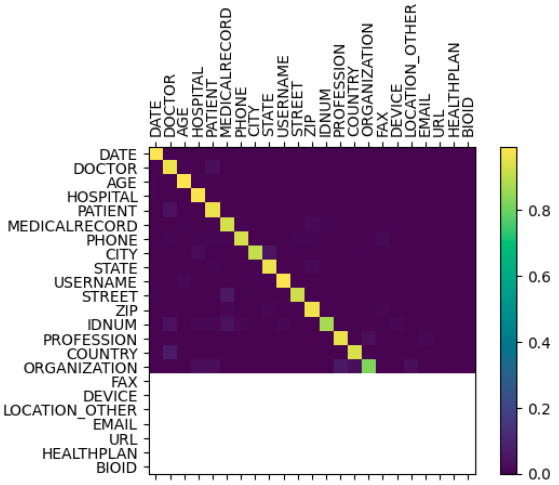


Figure 6.2: Fine-grained PHI classification confusion matrix

Some classes are completely missing from the test data, as can be deduced from the figure. These are FAX, DEVICE, LOCATION_OTHE, EMAIL, URL, HEALTHPLAN and BIOID. Figure 6.3 indeed illustrates the disproportionate distribution of all the classes within the test data. For the other frequently occurring classes the model performs well with a few notable challenging areas, particularly COUNTRY-DOCTOR.

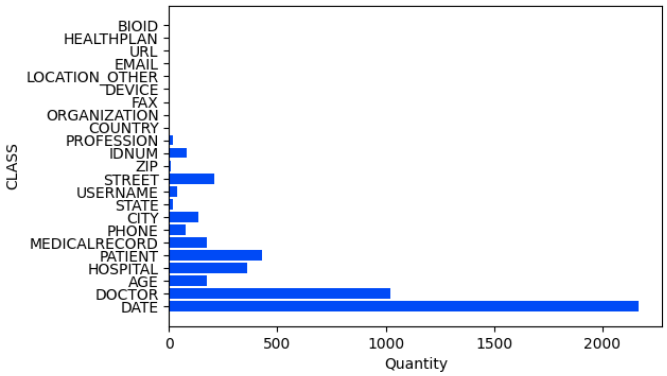


Figure 6.3: Distribution of the PHI classes over the test data

The distribution of the classes over the training data follows a similar pattern as illustrated in figure 6.3. This means the model may perform poorly should it encounter any of these classes in text to be de-identified. The same performance will ultimately be carried over to the mobile application when the model is ported and embedded onto the Android application. While an F-score of 0.943 is not as high as state-of-art systems which can achieve 0.98 on the same dataset, it is nevertheless acceptable given that the training

data does not have sufficient examples for some of the PHI classes, and the model can be used for PHI classification and de-identification.

6.5 Mobile Application De-identification

The researcher was able to de-identify freely typed text into the application which did not necessarily conform to the brief sentence structure and numerous abbreviations which characterize clinical notes on which the model had been trained. The mobile application can de-identify both the freely typed text as well as input textual files. For some classes of PHI, the de-identification performance was less than satisfactory, particularly for the classes not frequently encountered in the training data like hospital names, patient names, etc. This can be visualized in figure 6.4. For doctor names and dates, the application was able to de-identify these instances with better performance owing to the high occurrence of these classes in the training data. The application was able to de-identify sample text files taking an average of 1 second for files with average size of 5 kilobytes as was characteristic of the clinical note files used in this research.

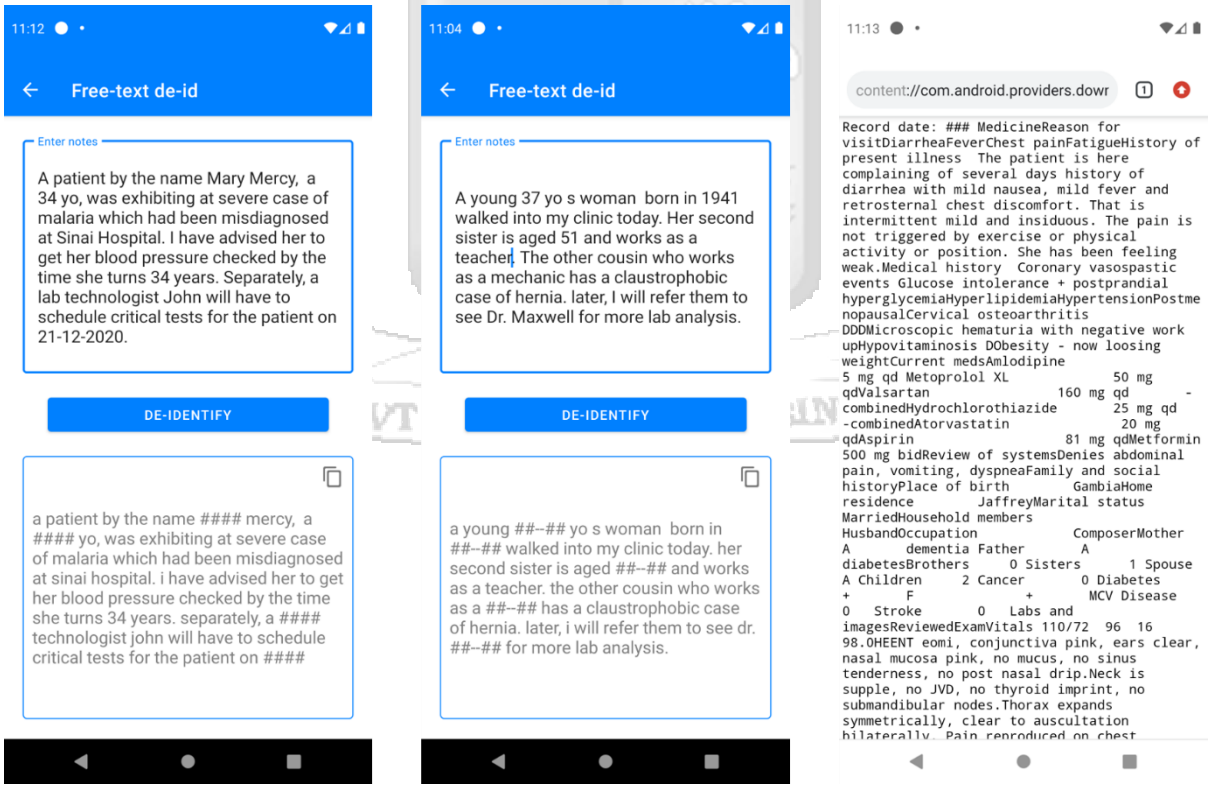


Figure 6.4: Performance of the mobile application on freely typed text and text-based files

The developed system brings de-identification capability into native mobile applications, which is the most significant advantage over current systems. Current deployments of de-identification systems are deployed on-premises or on cloud with significant connectivity requirements any clients using the service. On-premises systems require

local hardware and software setup with associated maintenance overhead. Cloud deployments on the other hand are more versatile but may be restricted due to data residency and transfer regulations especially where personally identifiable information is concerned. With a mobile de-identification system, the user has the capability to de-identify textual files on the smartphone by specifying a folder whose files are to be automatically de-identified in the background. The user can also type/paste text for instant de-identification. Both use cases can empower users to comply with data privacy regulations by ensuring files stored on their mobile health devices or shared with others are de-identified to mitigate privacy risks. Indeed, this research has demonstrated that a LSTM-based deep learning network model can be trained and deployed on a smartphone using a deep learning library such as PyTorch for application in NER tasks.

The mobile de-identification system resulting from this research suffered from several shortfalls. First, the dataset used for testing and training the deep learning model had a very disproportionate distribution of all PHI classes. The resultant model performed well on PHI classes frequently encountered in the dataset but poorly on PHI classes which had only a few samples in the dataset. This was directly reflected in the performance of the mobile application in classifying new data sets. The model also had many conditionals and loops hence the more efficient technique of conversion to high-performance deployment runtime via tracing could not be used and scripting had to be used instead. The researcher was also not able to implement an automated background task in the mobile application within the research timeframe. This task would have had a listener to trigger de-identification logic for any new files created/added to target folders.

CHAPTER 7: CONCLUSIONS & RECOMMENDATIONS

7.1 Conclusions

The objective of this research was to explore de-identification as a Named Entity Recognition problem based on deep learning networks with an aim of developing a Protected Health Information de-identification model that can be trained and embedded onto a mobile application. While de-identification systems exist today, designing and deploying this capability on smartphones is challenging as the trained deep-learning model needs to be converted to run on a mobile platform like Android using native C/C++ APIs and appropriately loaded via a service to execute the de-identification tasks asynchronously. Therefore, this study began with a review of the process of de-identification of protected health information (PHI). The researcher then appraised existing models, algorithms and frameworks which can be applied in the de-identification of PHI. Narrowing down to approaches based on deep-learning techniques, the researcher then explored de-identification of PHI as a named-entity recognition task (NER). The literature explored demonstrated indeed that advances in deep learning have made it possible to model complex neural networks architectures that can identify various tags in text and correctly classify them as PHI or non-PHI. Based on this, a detailed approach was laid out covering the steps required to design a PHI-de-identification model which can be transferred onto a mobile application. The steps to create the mobile application were also laid out. Based on this methodology and design, a deep learning model was trained from a dataset of clinical notes and an accompanying mobile application as well to use the trained model for de-identification tasks.

The resultant mobile application demonstrated satisfactory performance and was able to identify PHI tags from user-provided text and replace them with generic text for de-identification. A user could also select a directory for de-identification from which the application would read out all textual files contained therein and de-identify the contents and create new files/replace the original depending on the application settings.

7.2 Recommendations

Based on the outcomes of this research, the researcher recommends that there's a need to build richer datasets for named entity recognition, particularly in the healthcare sector. This is because the researcher was limited to a few choices of publicly accessible datasets. Moreover, there was no local datasets which would have had more contextual information and greater relevancy given that such medical notes are heavily context specific. An equally useful recommendation would be the formulation of policy guidelines to guide the distribution of target classes in a dataset, certainly for those intended for NER. This would help avoid datasets heavily skewed in terms of class distribution and help achieve more generalizable results.

7.3 Suggestions for Future Research

One area of this research that could be improved is the model performance by including more instances of the PHI class with few occurrences in the training data. The mobile application can also be improved by incorporating a listener which automatically initiates a de-identification task for any files created/added to certain target folders. Finally, a survey could be done with end-users to ascertain the extent to which the application can help complement workplace privacy and risk management measures by reducing storage and sharing of files containing PHI on workplace smartphones.

References

Agarwal, P., Alam, M., 2020. A Lightweight Deep Learning Model for Human Activity Recognition on Edge Devices. Procedia Comput. Sci. 167, 2364–2373. https://doi.org/10.1016/j.procs.2020.03.289

Al-Azzoa, F., Mohammed, A., Milanovab, M., 2018. Human Related-Health Actions Detection using Android Camera based on TensorFlow Object Detection API. Int. J. Adv. Comput. Sci. Appl. 9. https://doi.org/10.14569/IJACSA.2018.091002

Benedetto, J.I., Sanabria, P., Neyem, A., Navon, J., Poellabauer, C., Xia, B. (Ning), 2018. Deep Neural Networks on Mobile Healthcare Applications: Practical Recommendations. Proceedings 2, 550. https://doi.org/10.3390/proceedings2190550

Cohen, I.G., Mello, M.M., 2018. HIPAA and Protecting Health Information in the 21st Century. JAMA 320, 231. https://doi.org/10.1001/jama.2018.5630

Cohn, I., Laish, I., Beryozkin, G., Li, G., Shafran, I., Szpektor, I., Hartman, T., Hassidim, A., Matias, Y., 2019. Audio De-identification: A New Entity Recognition Task. ArXiv190307037 Cs.

Committee on Health Research and the Privacy of Health Information: The HIPAA Privacy Rule, Board on Health Sciences Policy, Board on Health Care Services, Institute of Medicine, 2009. Beyond the HIPAA Privacy Rule: Enhancing Privacy, Improving Health Through Research. National Academies Press, Washington, D.C. https://doi.org/10.17226/12458

Corbett, M., Chapman, L., O'Shea, J., Canning, P., Sebaouis, S., O'Sullivan, D., Power, N., O'Connor, M., 2019. Effects of Introducing a Secure Web-based Messenger Application for Communication Among Non-consultant Hospital Doctors (NCHDs). Cureus. https://doi.org/10.7759/cureus.5285

De Benedictis, A., Lettieri, E., Masella, C., Gastaldi, L., Macchini, G., Santu, C., Tartaglini, D., 2019. WhatsApp in hospital? An empirical investigation of individual and organizational determinants to use. PLOS ONE 14, e0209873. https://doi.org/10.1371/journal.pone.0209873

Dennis, A., Wixom, B.H., Tegarden, D., 2009. Systems Analysis & Design. John Wiley & Sons, Inc.

Dernoncourt, F., Lee, J.Y., Uzuner, O., Szolovits, P., 2016. De-identification of patient notes with recurrent neural networks. J. Am. Med. Inform. Assoc. ocw156. https://doi.org/10.1093/jamia/ocw156

Douglass, M.M., Cliffford, G.D., Reisner, A., Long, W.J., Moody, G.B., Mark, R.G., 2005. De-identification algorithm for free-text nursing notes, in: Computers in Cardiology, 2005. Presented at the Computers in Cardiology, 2005, IEEE, Lyon, France, pp. 331–334. https://doi.org/10.1109/CIC.2005.1588104

Florencio, F., Valen, T., Moreno, E.D., Junior, M.C., 2019. Performance Analysis of Deep Learning Libraries: *TensorFlow* and *PyTorch*. J. Comput. Sci. 15, 785–799. https://doi.org/10.3844/jcssp.2019.785.799

Greenleaf, G., Cottier, B., 2020. 2020 ends a decade of 62 new data privacy laws 5.

Guo, Y., Gaizauskas, R., Roberts, I., Demetriou, G., Hepple, M., 2006. Identifying Personal Health Information Using Support Vector Machines 5.

Hartman, T., 2020. Customization scenarios for de-identification of clinical notes 9.

He, B., Guan, Y., Cheng, J., Cen, K., Hua, W., 2015. CRFs based de-identification of medical records. J. Biomed. Inform. 58, S39–S46. https://doi.org/10.1016/j.jbi.2015.08.012

Ifeanyi Cosmas, N., 2018. Transitions in System Analysis and Design Methodology. Am. J. Inf. Sci. Technol. 2, 50. https://doi.org/10.11648/j.ajist.20180202.14

Institute of Medicine (U. S.), Grossmann, C., 2010. Clinical Data As the Basic Staple of Health Learning: Creating and Protecting a Public Good: Workshop Summary (Learning Health System Series). National Academies Press.

Juckett, D., 2012. A method for determining the number of documents needed for a gold standard corpus. J. Biomed. Inform. 45, 460–470. https://doi.org/10.1016/j.jbi.2011.12.010

Kamel Boulos, M., Giustini, D., Wheeler, S., 2016. Instagram and WhatsApp in Health and Healthcare: An Overview. Future Internet 8, 37. https://doi.org/10.3390/fi8030037

Kothari, C.R., 2004. Research Methodology Methods and Techniques. New Age International Publishers, New Delhi.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C., 2016. Neural Architectures for Named Entity Recognition, in: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Presented at the Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, San Diego, California, pp. 260–270. https://doi.org/10.18653/v1/N16-1030

Liu, Z., Chen, Y., Tang, B., Wang, X., Chen, Q., Li, H., Wang, J., Deng, Q., Zhu, S., 2015. Automatic de-identification of electronic medical records using token-level and character-level conditional random fields. J. Biomed. Inform. 58, S47–S52. https://doi.org/10.1016/j.jbi.2015.06.009

Ma, X., Hovy, E., 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Presented at the Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Berlin, Germany, pp. 1064–1074. https://doi.org/10.18653/v1/P16-1101

Merenda, M., Porcaro, C., Iero, D., 2020. Edge Machine Learning for AI-Enabled IoT Devices: A Review. Sensors 20, 2533. https://doi.org/10.3390/s20092533

Moldovan, D., Decker, J.M., Wang, F., Johnson, A.A., Lee, B.K., Nado, Z., Sculley, D., Rompf, T., Wiltschko, A.B., 2019. AutoGraph: Imperative-style Coding with Graph-based Performance. ArXiv181008061 Cs Stat.

Moore, W., Frye, S., 2019. Review of HIPAA, Part 1: History, Protected Health Information, and Privacy and Security Rules. J. Nucl. Med. Technol. 47, 269–272. https://doi.org/10.2967/jnmt.119.227819

Nadkarni, P.M., Ohno-Machado, L., Chapman, W.W., 2011. Natural language processing: an introduction. J. Am. Med. Inform. Assoc. JAMIA 18, 544–551. https://doi.org/10.1136/amiajnl-2011-000464

Neamatullah, I., Douglass, M.M., Lehman, L.H., Reisner, A., Villarroel, M., Long, W.J., Szolovits, P., Moody, G.B., Mark, R.G., Clifford, G.D., 2008. Automated de-identification of free-text medical records. BMC Med. Inform. Decis. Mak. 8, 32. https://doi.org/10.1186/1472-6947-8-32

Ogedebe, P.M., Jacob, B.P., 2012. Software Prototyping: A Strategy to Use When User Lacks Data Processing Experience 2, 6.

O'Sullivan, D.M., O'Sullivan, E., O'Connor, M., Lyons, D., McManus, J., 2017. WhatsApp Doc? BMJ Innov. 3, 238–239. https://doi.org/10.1136/bmjinnov-2017-000239

Pang, S., Wang, S., Rodríguez-Patón, A., Li, P., Wang, X., 2019. An artificial intelligent diagnostic system on mobile Android terminals for cholelithiasis by lightweight convolutional neural network. PLOS ONE 14, e0221720. https://doi.org/10.1371/journal.pone.0221720

Pennington, J., Socher, R., Manning, C., 2014. Glove: Global Vectors for Word Representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Presented at the Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, pp. 1532–1543. https://doi.org/10.3115/v1/D14-1162

Qiu, J., Wang, B., Zhou, C., 2020. Forecasting stock prices with long-short term memory neural network based on attention mechanism. PLOS ONE 15, e0227222. https://doi.org/10.1371/journal.pone.0227222

Stubbs, A., Kotfila, C., Uzuner, Ö., 2015. Automated systems for the de-identification of longitudinal clinical narratives: Overview of 2014 i2b2/UTHealth shared task Track 1. J. Biomed. Inform. 58, S11–S19. https://doi.org/10.1016/j.jbi.2015.06.007

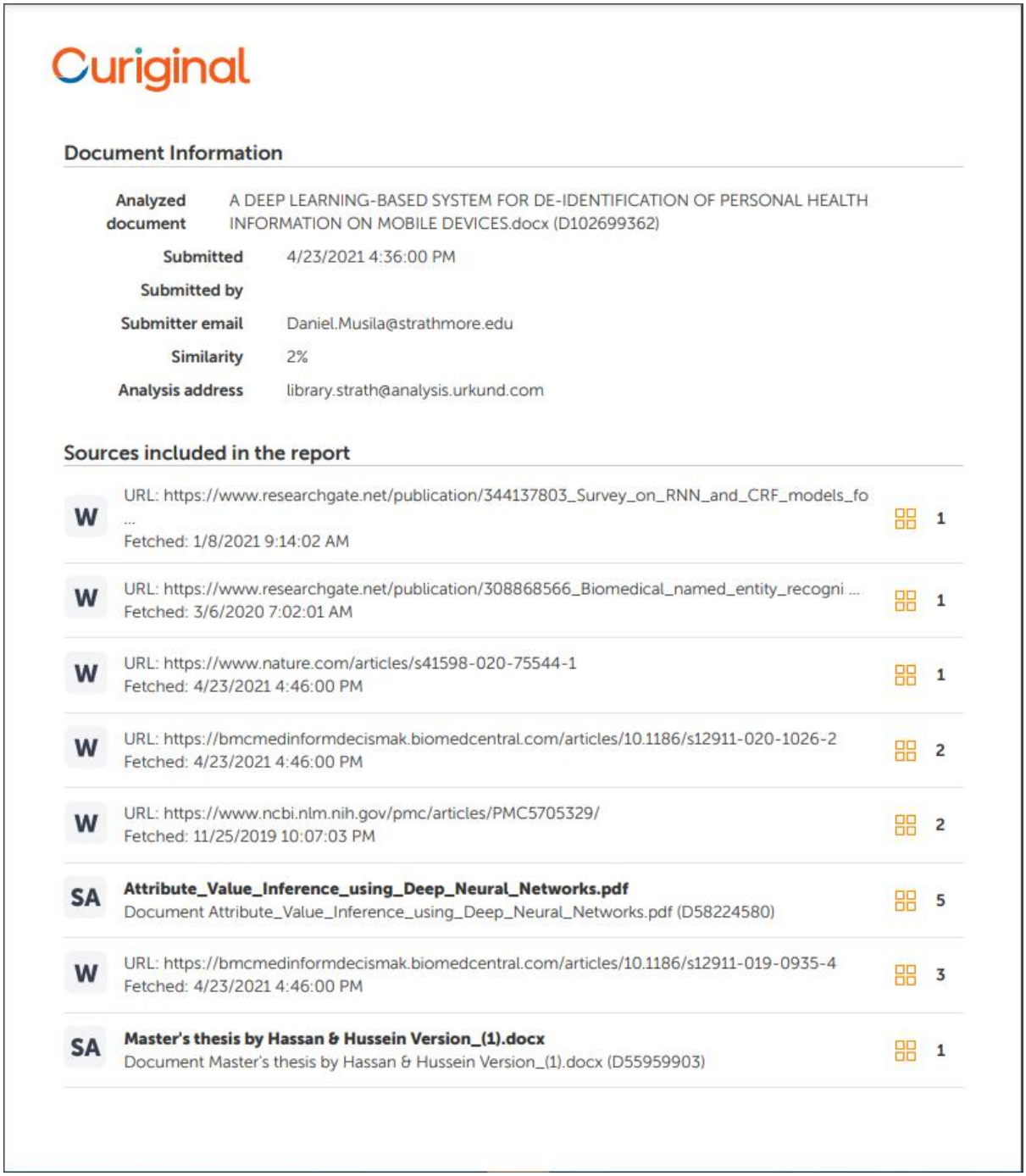
Sunner, D., Bajaj, H., 2016. Classification of Functional and Non-functional Requirements in Agile by Cluster Neuro-Genetic Approach. Int. J. Softw. Eng. Its Appl. 10, 129–138. https://doi.org/10.14257/ijseia.2016.10.10.13

Wahl, B., Cossy-Gantner, A., Germann, S., Schwalbe, N.R., 2018. Artificial intelligence (AI) and global health: how can AI contribute to health in resource-poor settings? BMJ Glob. Health 3, e000798. https://doi.org/10.1136/bmjgh-2018-000798

Yang, X., Lyu, T., Li, Q., Lee, C.-Y., Bian, J., Hogan, W.R., Wu, Y., 2019. A study of deep learning methods for de-identification of clinical notes in cross-institute settings. BMC Med. Inform. Decis. Mak. 19, 232. https://doi.org/10.1186/s12911-019-0935-4

Appendices

Appendix A: Originality report



Curiginal

Document Information

Analyzed document	A DEEP LEARNING-BASED SYSTEM FOR DE-IDENTIFICATION OF PERSONAL HEALTH INFORMATION ON MOBILE DEVICES.docx (D102699362)
Submitted	4/23/2021 4:36:00 PM
Submitted by	
Submitter email	Daniel.Musila@strathmore.edu
Similarity	2%
Analysis address	library.strath@analysis.urkund.com

Sources included in the report

W	URL: https://www.researchgate.net/publication/344137803_Survey_on_RNN_and_CRF_models_fo ... Fetched: 1/8/2021 9:14:02 AM	1
W	URL: https://www.researchgate.net/publication/308868566_Biomedical_named_entity_recogni ... Fetched: 3/6/2020 7:02:01 AM	1
W	URL: https://www.nature.com/articles/s41598-020-75544-1 Fetched: 4/23/2021 4:46:00 PM	1
W	URL: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1026-2 Fetched: 4/23/2021 4:46:00 PM	2
W	URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5705329/ Fetched: 11/25/2019 10:07:03 PM	2
SA	**Attribute_Value_Inference_using_Deep_Neural_Networks.pdf** Document Attribute_Value_Inference_using_Deep_Neural_Networks.pdf (D58224580)	5
W	URL: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-019-0935-4 Fetched: 4/23/2021 4:46:00 PM	3
SA	**Master's thesis by Hassan & Hussein Version_(1).docx** Document Master's thesis by Hassan & Hussein Version_(1).docx (D55959903)	1

Appendix B: Android Application build.gradle file

```
plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"

    defaultConfig {
        applicationId "com.danmutiso.mobile.mobiledeid"
        minSdkVersion 26
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {
    implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'
    def room_version = "2.2.6"
    def nav_version = "2.3.2"
    def versions_work = "2.3.3"

    implementation "androidx.navigation:navigation-fragment:$nav_version"
    implementation "androidx.navigation:navigation-ui:$nav_version"

    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'androidx.navigation:navigation-fragment:2.3.2'
    implementation 'androidx.navigation:navigation-ui:2.3.2'

    implementation 'com.google.android.material:material:1.3.0-beta01'

    implementation 'org.pytorch:pytorch_android:1.6.0'
    implementation 'org.pytorch:pytorch_android_torchvision:1.6.0'

    // https://mvnrepository.com/artifact/org.apache.opennlp/opennlp-tools
    implementation group: 'org.apache.opennlp', name: 'opennlp-tools', version: '1.9.3'

    implementation 'com.google.code.gson:gson:2.8.6'

    // https://mvnrepository.com/artifact/org.javatuples/javatuples
    implementation group: 'org.javatuples', name: 'javatuples', version: '1.2'

    implementation 'commons-codec:commons-codec:1.15'
```

Appendix C: Model Training Code Snippets

Training script parameters

```
In [3]: #parameters for the Model
parameters = OrderedDict()
parameters['train'] = "./data/train_conll" #Path to train file
parameters['dev'] = "./data/valid_conll" #Path to test file
parameters['test'] = "./data/test_conll" #Path to dev file
parameters['tag_scheme'] = "BIOES" #BIO or BIOES
parameters['lower'] = True # Boolean variable to control lowercasing of words
parameters['zeros'] =  False # Boolean variable to control replacement of  all digits by 0
parameters['char_dim'] = 30 #Char embedding dimension
parameters['word_dim'] = 100 #Token embedding dimension
parameters['word_lstm_dim'] = 200 #Token LSTM hidden layer size
parameters['word_bidirect'] = True #Use a bidirectional LSTM for words
parameters['embedding_path'] = "./data/glove.6B.100d.txt" #Location of pretrained embeddings
parameters['all_emb'] = 1 #Load all embeddings
parameters['crf'] =1 #Use CRF (0 to disable)
parameters['dropout'] = 0.5 #Droupout on the input (0 = no dropout)
parameters['epoch'] =  20 #Number of epochs to run"
parameters['weights'] = "" #path to Pretrained for from a previous run
parameters['name'] = "lite7-trained-model" # Model name
parameters['gradient_clip']=5.0
parameters['char_mode']="CNN"
models_path = "./models/" #path to saved models

#GPU
parameters['use_gpu'] = torch.cuda.is_available() #GPU Check
use_gpu = parameters['use_gpu']

parameters['reload'] = ""
# parameters['reload'] = "./models/lite7-trained-model"

#Constants
START_TAG = '<START>'
STOP_TAG = '<STOP>'
```

Sentence processing helper functions and data splitting

```
In [5]: def zero_digits(s):
    """
    Replace every digit in a string by a zero.
    """
    return re.sub('\d', '0', s)

def load_sentences(path, zeros):
    """
    Load sentences. A line must contain at least a word and its tag.
    Sentences are separated by empty lines.
    """
    sentences = []
    sentence = []
    for line in codecs.open(path, 'r', 'utf8'):
        line = zero_digits(line.rstrip()) if zeros else line.rstrip()
        if not line:
            if len(sentence) > 0:
                if 'DOCSTART' not in sentence[0][0]:
                    sentences.append(sentence)
                sentence = []
        else:
            word = line.split()
            assert len(word) >= 2
            sentence.append(word)
    if len(sentence) > 0:
        if 'DOCSTART' not in sentence[0][0]:
            sentences.append(sentence)
    return sentences
```

```
In [6]: train_sentences = load_sentences(parameters['train'], parameters['zeros'])
test_sentences = load_sentences(parameters['test'], parameters['zeros'])
dev_sentences = load_sentences(parameters['dev'], parameters['zeros'])

train_sentences = train_sentences
dev_sentences = dev_sentences
test_sentences = test_sentences

print(len(test_sentences))
```

Loading pre-trained word embeddings

```
In [14]: all_word_embeds = {}
for i, line in enumerate(codecs.open(parameters['embedding_path'], 'r', 'utf-8')):
    s = line.strip().split()
    if len(s) == parameters['word_dim'] + 1:
        all_word_embeds[s[0]] = np.array([float(i) for i in s[1:]])

#Intializing Word Embedding Matrix
word_embeds = np.random.uniform(-np.sqrt(0.06), np.sqrt(0.06), (len(word_to_id), parameters['word_dim']))

for w in word_to_id:
    if w in all_word_embeds:
        word_embeds[word_to_id[w]] = all_word_embeds[w]
    elif w.lower() in all_word_embeds:
        word_embeds[word_to_id[w]] = all_word_embeds[w.lower()]

print('Loaded %i pretrained embeddings.' % len(all_word_embeds))
```

```
Loaded 400000 pretrained embeddings.
```

Training step parameters

```
In [86]: learning_rate = 0.015
         momentum = 0.9
         # number_of_epochs = parameters['epoch']
         number_of_epochs = 40
         decay_rate = 0.05
         gradient_clip = parameters['gradient_clip']
         optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)

         losses = [] #list to store all losses
         loss = 0.0 #Loss initialization
         val_losses = [] #list to store all validation losses
         best_dev_F = -1.0 # Current best F-1 Score on Dev Set
         best_test_F = -1.0 # Current best F-1 Score on Test Set
         best_train_F = -1.0 # Current best F-1 Score on Train Set
         all_F = [[0, 0, 0]] # List storing all the F-1 Scores
         eval_every = len(train_data) # Calculate F-1 Score after this many iterations
         plot_every = 4000 # Store loss after this many iterations
         count = 0 #Counts the number of iterations
```

Trained model serialization for Android

```
In [71]: # serialize the model for Android
         import torch
         import torchvision


         cpu_model = model.cpu()
         model.eval() # put model in eval mode
         traced_script_module = torch.jit.script(model)
         traced_script_module.save("mobile_rad_deid_model.pt")
```