

Faculty of Natural and Mathematical Sciences

# Department of Engineering/Informatics

King's College London, Strand Campus, London, United Kingdom

7CCSMPRJ/7CCSMUIP (use the right module code)

Individual Project Submission 20XX/XX (Replace XX by the year)

Name: Changjoon Park

Student Number: K23030842

Degree Programme: Cyber Security

Project Title:

Supervisor:

Word count:

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

I **agree** to the release of my project

I **do not** agree to the release of my project

**Signature:** (An e-signature is acceptable, i.e., attach your signature as a picture, sign using the draw function)

Date:



Department of Informatics

King’s College London

United Kingdom

7CCSMPRJ/7CCSMUIP MSc Project

Project Title

Name: Changjoon Park

Student Number: K23030842

Degree Programme: Cyber Security

Supervisor’s Name: Fabio

This dissertation is submitted for the degree of MSc in Programme title goes here

Acknowledgement

It is a short paragraph to thank those whose have contributed to the project work.

Abstract

It is a precis of the report (normally in one page), which should include:

* A brief introduction to the project objectives
* A brief description of the main work of the project
* A brief description of the contributions, major findings, results achieved and principal conclusion of the project

Nomenclature

*a* The number of angels per unit area

*A* The area of the needle point

*c* Speed of light in a vacuum inertial frame

*h* Planck constant

LMI Linear Matrix Inequalities

*N* The number of angels per needle point

Contents

[1 Introduction 7](#_Toc20500399)

[1.1 Aims and Objectives 7](#_Toc20500400)

[1.1.1 Dissertation Length 7](#_Toc20500401)

[1.1.1.1 Structure 7](#_Toc20500402)

[1.1.1.1.1 Insert More Subsections if Necessary 7](#_Toc20500403)

[1.2 Background and Literature Survey 7](#_Toc20500404)

[1.3 Insert More Sections if Necessary 7](#_Toc20500405)

[2 Background Theories 8](#_Toc20500406)

[3 Objectives, Specifications and Design 9](#_Toc20500407)

[4 Methodology and Implementation 10](#_Toc20500408)

[5 Results, Analysis and Evaluation 11](#_Toc20500409)

[6 Legal, Social, Ethical and Professional Issues 12](#_Toc20500410)

[7 Conclusion 13](#_Toc20500411)

[8 References 14](#_Toc20500412)

[9 Appendices 15](#_Toc20500413)

[9.1 Appendix A: Heading 15](#_Toc20500414)

[9.2 Appendix B: Points to Note 15](#_Toc20500415)

List of Tables

List of Figures

# Introduction

The accelerating evolution of malware poses significant threats to systems security, necessitating advanced defenses that not only detect but also understand malware mechanisms. Machine Learning (ML) has emerged as a cornerstone in crafting sophisticated malware detection systems. Unlike traditional signature-based methods, ML-based malware analysis systems can classify unknown malware and detect new variants with high accuracy, offering a significant advantage in the constantly evolving cybersecurity landscape.

However, the complexity of ML models used in malware detection raises concerns about their transparency and reliability. These models often operate as black boxes, making it difficult to understand their decision-making processes. This lack of transparency can undermine trust in the effectiveness and accuracy of these models. One of the main issues is overfitting, where the model learns to recognize irrelevant features in the training data instead of generalizing from essential ones. This results in poor performance when the model encounters new, unseen data.

To address these challenges, this project aims to train ML-based classifiers and explore the use of existing explanation methods, such as SHAP (SHapley Additive exPlanations), to enhance our understanding of the learning process within classifiers. By focusing on the API features subset and analysing how SHAP identifies and evaluates the importance of features used in the dataset, we can better discern and eliminate redundant features, thus reducing overfitting during model training. This targeted feature selection and comprehensive model training are expected to lead to more robust and accurate malware detection systems.

## Aims and Objectives

The primary aim of this project is to increase our understanding of what ML models actually learn from the data towards achieving high performance in detecting malware and its variants.

### Optimize Feature Sets for Malware Classification

• Algorithm Selection: Identify the most effective ML algorithms for Android malware detection through experimental comparisons. This includes evaluating linear models like Linear SVM and non-linear models like Random Forest and Deep Learning (e.g., Convolutional Neural Networks).

• Feature Set Comparison: Evaluate the performance of different feature subsets, particularly focusing on API features, against the full feature set to determine their efficiency and accuracy.

• Feature Analysis: Systematically assess individual features within the API feature set to establish their significance and contribution to the effectiveness of the malware classification model.

### Evaluate the Use of Explanation Methods

• Explanation Method Analysis: Investigate and critically assess the capacity of SHAP to provide transparency in the ML decision-making process. This method will help identify which features are most important for the model’s decisions.

• Enhancement of Model Design: Determine how SHAP can be integrated into the malware classification workflow to avoid overfitting and enhance the design and understanding of detection algorithms. This involves using SHAP to refine feature selection and model parameters, leading to improved generalization and reduced false positives and negatives.

By focusing on these objectives, the project aims to improve the design and performance of ML-based malware detection systems, making them more transparent and reliable. The project will utilize a dataset derived from the “Transcending Transcend” paper, which uses the DREBIN feature space but is not the same dataset as the original DREBIN paper. This dataset includes five years of data (2014-2018) with malware-family labels, providing a rich source for analysis and model training. The project will specifically focus on the API features subset, allowing for a detailed investigation into their role in malware detection while training and evaluating ML classifiers.

## Background and Literature Review

### State-of-the-Art in Android Malware Detection

The landscape of Android malware detection has dramatically evolved over the past decade, transitioning from traditional signature-based methods to sophisticated machine learning (ML) models. Signature-based approaches, while effective against known threats, struggle to keep pace with the rapid proliferation of malware variants and novel threats. This challenge has spurred the development of ML models that can learn from vast datasets to identify and categorize malware based on behavioural patterns rather than static signatures.

A seminal contribution to this field is encapsulated in the "DREBIN" paper, which introduces a lightweight method for detecting Android malware directly on smartphones. Unlike traditional methods, DREBIN performs a broad static analysis that collects extensive features from Android applications, including requested permissions, API calls, and network addresses. These features are then embedded into a vector space, enabling the application of ML techniques to detect malicious patterns automatically. Crucially, DREBIN not only identifies malware but also provides explanations for its detections, making its decisions transparent and understandable to users [1].

### Machine Learning Models in Malware Detection

Among the various ML models employed in malware detection, Linear Support Vector Machines (SVM) and Random Forests stand out due to their efficacy and efficiency. Linear SVM is particularly valued for its ability to handle high-dimensional data as it constructs a hyperplane in a multidimensional space to differentiate between the classes (malicious vs. benign). This model is robust in environments with clear margin separation between classes and has been effectively used in systems like DREBIN [2].

On the other hand, Random Forests offer advantages in scenarios where the decision boundaries are more complex. As an ensemble learning method that builds multiple decision trees and merges them to get more accurate and stable predictions, Random Forests are a consistent and adaptable learning algorithm for classifying non-linear data. They are particularly useful in malware detection for their ability to perform feature selection and handle unbalanced data, which is common in cybersecurity threats. In experimental evaluations, Random Forest achieved a high accuracy rate of 99.78% when combined with Variance Threshold feature reduction, significantly outperforming deep learning models which reached accuracy rates of around 98.99% [3]. This demonstrates Random Forest’s robustness in complex malware detection scenarios, excelling in environments with intricate decision-making requirements.

### Explanation Methods in Machine Learning

As ML models have become more prevalent in high-stakes domains like cybersecurity, the need for explainability in these models has grown. Explanation methods such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) address this need by providing insights into the decision-making processes of ML models. SHAP values interpret the impact of having a certain value for a given feature in comparison to the prediction we'd make if that feature took some baseline value, thus offering an in-depth look at the contribution of each feature to the output of the model [4].

### Critical Review of Literature

Machine learning models have revolutionized the approach to malware detection by focusing on behavioural patterns rather than static characteristics. A prime example of this evolution is the "DREBIN" approach, which leverages a broad static analysis to gather extensive application features from Android apps, such as used permissions, API calls, and network addresses. These features are processed in a machine learning framework to identify indicative patterns of malware.

"DREBIN" stands out for its effectiveness in real-time environments. In an evaluation involving over 120,000 applications, including more than 5,000 confirmed malware samples, "DREBIN" achieved a detection rate of 94% with a low false-positive rate of 1%. This performance significantly outstrips traditional antivirus solutions, which struggle with rapidly mutating malware signatures and often fail to detect new, unknown threats [6].

However, while the static analysis method used by "DREBIN" is powerful, it is not without limitations. Static methods can be evaded by sophisticated malware that employs dynamic code execution or obfuscation techniques, which are not detectable without executing the application. Despite these challenges, "DREBIN" enhances the transparency of its process through explainable detection, providing justifications for each of its detections, which is a considerable advantage over black-box models that offer no insight into their internal decision-making processes [7].

### Novel Contributions of the “DREBIN” Methodology

The novel contributions of the "DREBIN" methodology are particularly noteworthy. First, its comprehensive feature extraction strategy that encompasses both manifest and code properties of Android applications allows for a granular analysis of potential threats. By embedding these features into a joint vector space, "DREBIN" facilitates the use of linear classifiers to efficiently identify malware, a method that has proven effective even on constrained devices like smartphones.

Furthermore, "DREBIN" introduces explainability to the realm of malware detection. Using techniques akin to those in the SHAP and LIME frameworks, "DREBIN" provides explanations for its detections by highlighting the specific application features that most contributed to its decision. This not only aids security analysts and researchers in understanding the basis of the detection but also builds trust among users by clarifying why certain applications are flagged as malicious [8].

### Limitations and Areas for Improvement

While "DREBIN" and similar ML-based methods have significantly advanced Android malware detection, they also have limitations that must be addressed. The reliance on static features, while resource-efficient, misses dynamically loaded code and runtime behaviours that could signify sophisticated malware attacks. Additionally, the static nature of the analysis may not keep pace with the rapid evolution of Android malware, which continuously develops new evasion techniques.

Moreover, the explainability provided, while pioneering, is often limited to the scope of the features considered in the analysis. As malware developers increasingly incorporate complex obfuscation and evasion techniques, the explanatory models may need to evolve to maintain their effectiveness and accuracy [9].

References

[1] A. Dan, S. Michael, H. Malte, G. Hugo, and R. Konrad, “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), 2014.

[2] C. Cortes and V. Vapnik, "Support-vector networks," \*Machine Learning\*, vol. 20, no. 3, pp. 273-297, 1995.

[3] L. Breiman, "Random forests," \*Machine Learning\*, vol. 45, no. 1, pp. 5-32, 2001.

[4] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in \*Advances in Neural Information Processing Systems\* (NIPS), 2017.

[6] L. Liu, W. Baosheng, Y. Bo, and Z. Qiuxi, “Automatic malware classification and new malware detection using machine learning,” \*Frontiers of Information Technology & Electronic Engineering\*, vol. 18, pp. 1336-1347, 2017.

[7] G. Daniel, M. Carles, and P. Jordi, “The rise of machine learning for detection and classification of malware: Research developments, trends, and challenges,” \*J. Netw. Comput. Appl.\*, vol. 153, p. 102526, 2020.

[8] S. Erwan and J.-P. V. Biau, “Consistency of Random Forests,” \*Annals of Statistics\*, vol. 43, pp. 1716-1741, 2014.

[9] R. Hemant, A. Swati, and S. Mohit, “Malware Detection Using Machine Learning and Deep Learning,” in \*Lecture Notes in Computer Science\*, Springer International Publishing, 2018, p. 402–411.

# Background Theories

The background theories supporting the work should be given in this section. Provide references when someone’s work is recalled.

## Machine Learning Models in Malware Detection

This project focuses on three primary machine learning models: Linear Support Vector Machines (SVM), Random Forests, and Deep Learning (specifically Convolutional Neural Networks). These models have been selected for their effectiveness and complementary strengths in malware detection.

### Linear Support Vector Machines (Linear SVM)

Linear Support Vector Machines are a type of supervised learning model that aims to find the optimal hyperplane that separates data points of different classes with the maximum margin. The linear SVM is particularly effective in high-dimensional spaces and is well-suited for binary classification tasks such as malware detection.

- \*\*Advantages\*\*: Linear SVMs are computationally efficient and can handle large feature spaces, making them ideal for malware detection where the input feature space can be extensive. They also provide clear decision boundaries which are easy to interpret.

- \*\*Applications in Malware Detection\*\*: SVMs are used to classify malware by learning the characteristics that differentiate malicious files from benign ones based on features extracted from the code and behaviour of applications [12].

### Random Forests

Random Forests are an ensemble learning method that builds multiple decision trees and merges their predictions to improve accuracy and robustness. Each tree in the forest is trained on a random subset of the data, and the final prediction is made by averaging the predictions of all the trees.

- \*\*Advantages\*\*: Random Forests are highly robust to overfitting, especially when dealing with noisy data. They can handle large datasets and provide estimates of feature importance, which are useful for understanding the model’s decisions.

- \*\*Applications in Malware Detection\*\*: Random Forests are effective in malware detection as they can manage the complexity and variability of malware features. They are particularly useful for identifying which features (e.g., API calls, permissions) are most indicative of malicious behaviour [13].

## Explainable AI in Malware Detection

Despite the success of ML models in malware detection, their black-box nature poses significant challenges for interpretability and trust. Explainable AI (XAI) aims to enhance transparency by providing insights into the decision-making processes of these models. This transparency is crucial for several reasons: it helps build trust among users and stakeholders, facilitating broader adoption of these technologies in security operations; it aids in uncovering and correcting biases in the training data or the model itself, leading to more robust and fair detection systems; and it helps meet regulatory requirements that mandate transparency in automated decision-making systems by providing clear and understandable justifications for the model’s decisions.

### Methods of Explainable AI

Several methods have been developed to provide explanations for ML models, with SHAP (SHapley Additive exPlanations) being one of the most prominent in the context of malware detection.

1. **SHAP (SHapley Additive exPlanations)**. SHAP values are based on cooperative game theory and provide a unified measure of feature importance. They explain the output of any ML model by assigning each feature an importance value, representing its contribution to the prediction. SHAP values are consistent and locally accurate, making them suitable for detailed analysis of model behaviour [7].
2. **LIME (Local Interpretable Model-agnostic Explanations)**. LIME approximates the decision boundary of a black-box model with an interpretable model in the vicinity of a prediction. It helps understand the impact of each feature on individual predictions. However, LIME may not always be consistent with the global behaviour of the model, which is why SHAP is often preferred for a more comprehensive explanation [8].

### Application in Malware Detection

Explainable AI methods like SHAP can significantly enhance our understanding of how ML models make decisions in malware detection. By providing feature importance scores and visualizations, these methods help in:

* **Identifying Redundant Features**: Features that do not contribute significantly to the model’s decisions can be identified and removed, reducing the risk of overfitting, and improving model performance. This pruning process not only simplifies the model but also enhances its generalization capabilities. For instance, in malware detection, removing irrelevant API calls that do not influence the outcome can streamline the model without compromising accuracy [9].
* **Improving Model Transparency**: Understanding which features influence model predictions builds trust in the system, making it easier to deploy in real-world scenarios. Transparent models allow security analysts to validate and interpret decisions, ensuring that the model's logic aligns with domain knowledge and expectations. This is particularly important in high-stakes environments where false positives and negatives can have significant consequences [10].
* **Enhancing Feature Selection**: By highlighting the most important features, SHAP aids in selecting the most relevant features for model training, leading to more accurate and robust classifiers. Feature selection is a critical step in building effective ML models as it directly impacts the model's performance and interpretability. In the context of malware detection, focusing on key indicators such as specific API calls or permission requests can enhance the detection accuracy and robustness of the classifiers [11].

## DREBIN Feature Space

The dataset used in this project is based on the DREBIN feature space, as described in the "Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift" paper [14]. It is important to clarify that this dataset is not the same as the original DREBIN dataset introduced by Arp et al. in their NDSS 2014 paper [15]. Instead, it employs a similar feature extraction methodology applied to a different collection of malware samples spanning five years (2014-2018). This approach ensures a rich and diverse dataset for training and evaluating machine learning models for malware detection.

### Feature Categories

The DREBIN feature space comprises a variety of static features extracted from Android applications. These features are grouped into several categories, each representing different aspects of the application’s behaviour and characteristics. The primary feature categories used in the dataset are as follows:

1. **Hardware Components**. This category includes features indicating the use of specific hardware components such as the camera, GPS, and sensors. The presence of these features can signal attempts by the application to access hardware functionalities that may be exploited for malicious purposes.
2. **Requested Permissions**. Permissions requested by the application during installation are captured in this category. Permissions such as access to contacts, SMS, and the internet are scrutinized, as excessive or unnecessary permission requests can indicate malicious intent.
3. **App Components**. Declared components in the application's manifest file, including activities, services, and broadcast receivers, are considered in this category. These components define the application's core functionalities and its interaction with the Android operating system.
4. **Filtered Intents**. Intents listed in the manifest file that the application is interested in are part of this category. Intents are messaging objects used for communication between different components of an application and other applications. Monitoring intents can reveal potentially suspicious communication patterns.
5. **Restricted API Calls**. This category includes API calls that require special permissions, indicating access to sensitive data or system functionalities. Restricted API calls can be indicative of actions such as reading SMS, accessing user data, or making network connections.
6. **Used Permissions**. Unlike requested permissions, used permissions track the permissions that are actively utilized by the application during runtime. This distinction helps identify discrepancies between requested and actual usage of permissions.
7. **Suspicious API Calls**. API calls that are commonly associated with malicious activities are included in this category. Examples include calls to access personal data, send SMS, or initiate network connections. These calls are flagged based on their historical association with known malware behavior.
8. **Network Addresses**. This category captures IP addresses, hostnames, and URLs found within the application's code. Network addresses are crucial for detecting potential communication with command-and-control servers or exfiltration of data to remote locations.

The DREBIN feature space is instrumental in capturing a comprehensive view of an application’s behaviour. By examining these static features, it is possible to construct a detailed profile of the application, which can be used to distinguish between benign and malicious behaviour. The richness of the feature set allows for robust machine learning models that can generalize well across different types of malware.

References

[7] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in \*Advances in Neural Information Processing Systems\* (NIPS), 2017.

[8] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," in \*Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining\* (KDD), 2016.

[9] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in \*Advances in Neural Information Processing Systems\* (NIPS), 2017.

[10] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," in \*Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining\* (KDD), 2016.

[11] D. Arp et al., "DREBIN: Effective and explainable detection of Android malware in your pocket," in \*NDSS\*, 2014.

[12] C. Cortes and V. Vapnik, "Support-vector networks," \*Machine Learning\*, vol. 20, no. 3, pp. 273-297, 1995.

[13] L. Breiman, "Random forests," \*Machine Learning\*, vol. 45, no. 1, pp. 5-32, 2001.

[14] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, “Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift,” \*IEEE Security & Privacy\*, 2022.

[15] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket,” in \*NDSS\*, 2014.

# Objectives, Specifications and Design

## Objectives

The primary objective of this project is to enhance our understanding of what machine learning models learn from data to achieve high performance in detecting malware and its variants. This objective is twofold:

1. **Feature Set Evaluation**. Verify which feature sets are more suitable for malware classification. Specifically, the project will focus on the subset of features assigned to the project variant, which in this case are the API features. This involves determining the most relevant features that contribute to the accuracy of malware detection.
2. **Explanation Methods Analysis**. Understand whether explanation methods, such as SHAP (SHapley Additive exPlanations), can help us improve the design of detection algorithms. This includes using SHAP to evaluate the importance of different features and understand the learning process within classifiers.

## Specification and Design

The design of this project is structured to achieve the stated objectives by developing and analysing machine learning models, focusing on the assigned API features, and employing explanation methods to provide insights into the model's decision-making process.

### Tools Used

The DREBIN feature space comprises a variety of static features extracted from Android applications. These features are grouped into several categories, each representing different aspects of the application’s behaviour and characteristics. The primary feature categories used in the dataset are as follows:

To achieve the project goals, several tools and technologies will be utilized:

* **Python**: The primary programming language for developing machine learning models and implementing SHAP explanations.
* **Scikit-learn**: A library for implementing machine learning algorithms, including Linear SVM and Random Forest.
* **SHAP Library**: For generating SHAP values to explain model predictions.

### Machine Learning Models

The project will focus on three types of machine learning models:

* **Linear SVM (Support Vector Machine)**: Suitable for high-dimensional data, providing a clear margin of separation between classes. Linear SVM will be used to identify patterns in the API features that distinguish malware from benign software.
* **Random Forest**: An ensemble learning method that builds multiple decision trees and merges their results to improve the predictive performance and control overfitting. It is particularly useful for handling the complexities and non-linearities in the dataset.

### Feature Space and Dataset

The dataset used in this project is derived from the DREBIN feature space, as described in the "Transcending Transcend" paper [1]. It includes data collected from 2014 to 2018 with malware-family labels. The dataset consists of several feature subsets, with this project focusing specifically on API features. These features include the API calls made by Android applications, which are critical for detecting malicious behaviour. The analysis will involve identifying which API calls are most indicative of malware.

### Explanation Methods

The SHAP method will be employed to provide insights into the learning process of the classifiers. SHAP values will be used to interpret the output of machine learning models by attributing the contribution of each feature to the final prediction. This helps in understanding the importance of different API features in detecting malware.

##### 3.2.5 Design Approach

The design approach involves the following steps:

1. \*\*Data Preprocessing:\*\*

- Clean and preprocess the dataset to ensure that the API features are in a suitable format for machine learning algorithms.

2. \*\*Model Training:\*\*

- Train the Linear SVM, Random Forest, and Deep Learning models using the preprocessed dataset. Evaluate their performance using metrics such as precision, recall, F1-score, and ROC-AUC to handle the class imbalance problem inherent in malware detection datasets.

3. \*\*Feature Importance Analysis:\*\*

- Use SHAP values to analyze the trained models and identify the most important API features contributing to the detection of malware. This involves generating visualizations to explain the model's decisions.

4. \*\*Evaluation and Validation:\*\*

- Validate the models and the SHAP analysis through cross-validation techniques and test the models on a separate validation set to ensure their robustness and generalizability.

5. \*\*Iteration and Optimization:\*\*

- Based on the insights from the SHAP analysis, iterate on feature selection and model tuning to improve detection performance. This includes eliminating redundant features and optimizing hyperparameters for better accuracy and efficiency.

By following this detailed design approach, the project aims to provide a comprehensive understanding of how machine learning models detect malware, identify the most critical features, and improve the transparency and effectiveness of the detection algorithms.

### References

[1] A. Dan, S. Michael, H. Malte, G. Hugo, and R. Konrad, “DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), 2014.

# Methodology and Implementation

It presents and justifies the methodology used to deal with the problem and describes in detail the implementation procedures. The background theory presented in the previous chapter can be recalled to support the proposed implementation. The originality, novelty and contribution are to be demonstrated with the discussion of the strengths and limitations.

# Results, Analysis and Evaluation

It summarises the results obtained from the proposed design and methodology. The way to obtain the results should be described in detail. Analysis and evaluation have to be performed. Comparisons should be made. It should justify if the project aims, objectives, requirements and specifications have been achieved.

# Legal, Social, Ethical and Professional Issues

A chapter gives a reasoned discussion about legal, social ethical and professional issues within the context of your project problem. You should also demonstrate that you are aware of the Code of Conduct \& Code of Good Practice issued by the British Computer Society (BSC) (https://www.bcs.org/membership/become-a-member/bcs-code-of-conduct) for computer science project and Rule of Conduct issued by The Institution of Engineering and Technology (IET) (https://www.theiet.org/about/governance/rules-of-conduct) for engineering project. You should have applied their principles, where appropriate, as you carried out your project. You could consider aspects like: the effects of your project on the public well-being, security, software trustworthiness and risks, Intellectual Property and related issues, etc.

# Conclusion

It is a chapter to sum up the main points and findings of the work; how you achieve the project aims and address the research questions; the contributions and results you have achieved. Future plan and development can be mentioned in this section as well. It is normally in one or two pages.

# References

Refer to the citing reference information on KEATS

# Appendices

## Appendix A: Heading

Supplementary materials (such as source code, user menu, etc) could be included. Each appendix must be labelled (for example, Appendix A, Appendix A.1, Appendix A.2, Appendix B, Appendix B.1, etc.) and with heading. All Appendices must be referred in the text.

## Appendix B: Points to Note

* Please note the following points when you write your report:
* Consider the outline of the report. It is a good idea to start with the table of contents, which gives you an overall structure of the report.
* Show understanding of the topic and demonstrate the contribution of the work. 70\% of the content of the report should be your own contributions and achievements.
* Always use your own words.
* The main report and any appendices must constitute one document.
* Pages must be numbered consecutively.
* Captions must be provided for all figures and tables.
* Equations (or important equations), figures and tables must be numbered.
* All figures and tables must be referred to in the text.
* Units of all variables must be provided.
* Numerical values (floating-point number) should be in 4 decimal places.
* Contractions should not be used.
* Check the punctuation of sentences. In particular, those sentences with equation. For example, if an equation is at the end of a sentence, a full stop should be used.
* All variables must be defined.
* Font face of variables throughout the report (in the text, equation, figures and table) must be consistent.
* Use proper headings for chapters, sections, subsections.
* Chapters, sections, subsections should be numbered and with the same numbering system throughout the report.
* It is suggested that vector and matrix variables should be in bold, scalar variables should be in italic.
* References must be used for materials used in the report that are not yours.
* A standard reference format must be adopted and be consistently applied through the report. General guidelines for reference format can be found on KEATS.
* Always backup your files.