National College of Ireland

An Enhanced Deep Learning Approach to Detect Cyberattacks and Malware in Cybertext

MSc Research Project
MSc in Cybersecurity

Aayush Sharma
Student ID: 23422211

School of Computing
National College of Ireland

Supervisor: Arghir Nicolae Moldovan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aayush Sharma
Student ID:	23422211
Programme:	MSc. in Cybersecurity **Year:** 2025
Module:	Research Practicum
Lecturer:	Arghir Nicolae Moldovan
Submission Due Date:	28-01-2026
Project Title:	Configuration Manual
Word Count:	809 **Page Count: 5**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aayush Sharma

Date: 28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

An Enhanced Deep Learning Approach to Detect Cyberattacks and Malware in Cybertext

Aayush Sharma
Student ID: x23422211

1. Introduction

The presented project will construct an end-to-end research pipeline to identify cyberattacks and malware intent using short, noisy, context-dependent cybertext and categorize it into threat-relevant classes to facilitate prioritization and save on the workload of analysts. This is tricky since cybertext is heterogeneous, antagonistic, and likely to be class-imbalanced: the same indicators can be innocent or malicious in different contexts, attackers obfuscate and distract, and significant cues may entail subtle semantics and long-range relationships. In order to address these problems, the system uses text normalization and label consolidation followed by classical machine-learning baselines and then deep sequence models and finally a mixed ALBERT+LSTM architecture which uses contextual embeddings with sequential learning to achieve robust and reproducible performance.

2. System Specifications

The configuration described below reflects the environment used to preprocess the dataset, extract features, train models, generate embeddings, and evaluate performance.

Hardware Requirements

- Minimum **4-core CPU** (recommended for pre-processing and baseline training)
- **RAM:** at least **8 GB** (stable execution), **16 GB** preferred (faster experimentation)
- **Storage: 5–10 GB** free disk space (datasets, cached transformer weights, logs, model artefacts)
- **GPU (optional but recommended):** CUDA-capable **NVIDIA GPU** for faster embedding extraction and neural training
- **VRAM: 6 GB+** suggested for comfortable ALBERT-based runs

Software Requirements

- **Operating system:** Windows / Linux / macOS
- **Python: 3.9+** (recommended), **3.10** preferred
- **Environment:** use **venv** or **conda** to freeze dependency versions
- **GPU stack (if used): CUDA + cuDNN** versions must match the installed **TensorFlow/PyTorch** build
- Ensure **pip**, required build tools, and **GPU runtime libraries** are available in the same active environment used for training.

3. Required Libraries

The pipeline uses core libraries for end-to-end text ML: **NumPy/Pandas** for data handling, **scikit-learn** for splits, metrics, and classical baselines, **XGBoost** for gradient-boosted tree models, **SpaCy** for text pre-processing and lemmatization, **Transformers** for ALBERT tokenization and contextual embeddings, and **TensorFlow/Keras or PyTorch** for building and training LSTM-based deep and hybrid models with callbacks and checkpointing.

a) **pip install numpy pandas scikit-learn xgboost matplotlib spacy transformers torch tensorflow**
b) **python -m spacy download en_core_web_sm**

4. Dataset description

The dataset used in this project consists of cyber intelligence text entries representing short, noisy, and context-rich cybertext such as incident notes, threat reports, indicators-of-compromise narratives, and operational security messages, each paired with a target label indicating the relevant threat category. At minimum, the data is organized with a **text** field containing the message content and a **label** field containing the corresponding class (e.g., attack/malware/benign or related threat-intent categories), with optional metadata fields such as IDs, entity/indicator tags, offsets, or auxiliary annotations depending on the source format.

5. Models Implemented

The implementation steps include classical baselines, deep learning, and lastly a hybrid architecture. It starts with Random Forest and XGBoost as highly fast and stable reference models where the performance of these two models relies on the engineered text features like TF-IDF or n-grams. It next proceeds to repetitive deep architectures such as BiLSTM and GRU which are trained on text to acquire sequential dependencies and context patterns. The last and most successful stage is a hybrid ALBERT+LSTM model, in which ALBERT is used to give contextual embeddings and the LSTM is trained to learn task-specific sequence patterns on top of these, enhancing robustness to noisy and adversarial cybertext.

6. Code snippets and plots:

1) Dataset Description

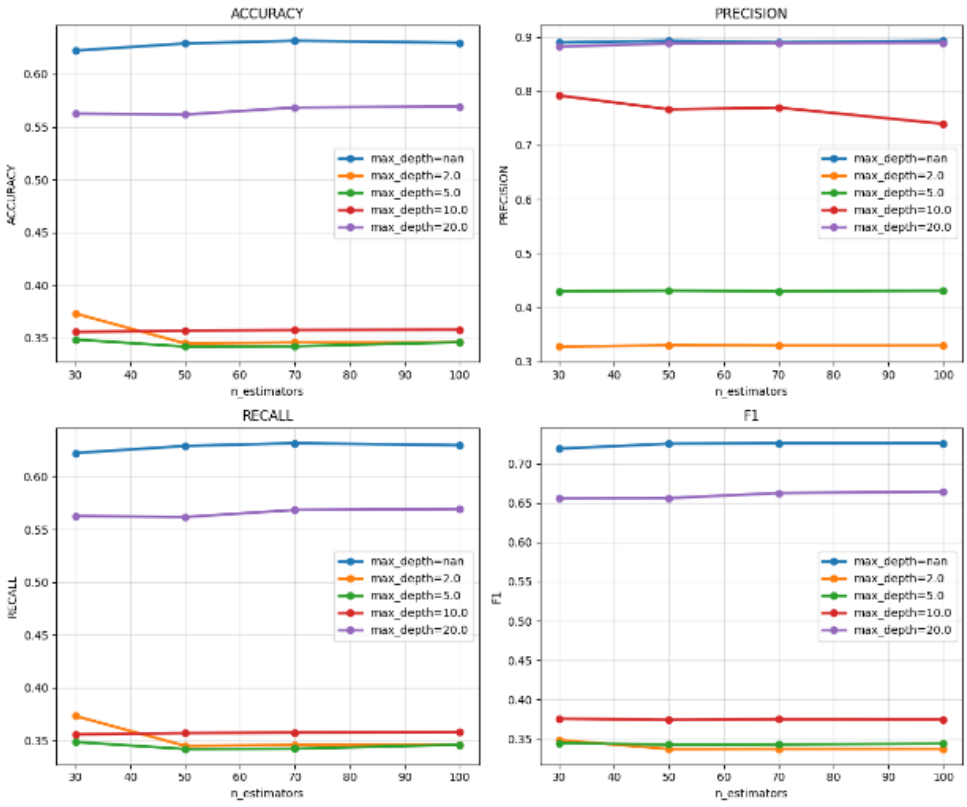
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19940 entries, 0 to 19939
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   index         19940 non-null  float64
 1   text          19940 non-null  object
 2   entities      19940 non-null  object
 3   relations     19940 non-null  object
 4   Comments      19940 non-null  object
 5   id            19940 non-null  float64
 6   label         19940 non-null  object
 7   start_offset  19940 non-null  float64
 8   end_offset    19940 non-null  float64
dtypes: float64(4), object(5)
memory usage: 1.4+ MB
```

2) Splitting Dataset

```
from sklearn.model_selection import train_test_split
import pandas as pd

# Split
x_train, x_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.20, random_state=42
)
```

3) Random Forest plot of hyper parameters



4) Albert embeddings

```python
import tensorflow as tf
import numpy as np
from tqdm import tqdm

def get_albert_embeddings_safe(texts, tokenizer, model, max_len=64, batch_size=32, device='CPU'):
    """
    Generate ALBERT embeddings safely on Colab without crashing.
    Saves memory by processing in small batches.
    """
    all_embeddings = []

    device = '/GPU:0' if (tf.config.list_physical_devices('GPU') and device.upper() == 'GPU') else '/CPU:0'
    print(f"Using device: {device}")

    with tf.device(device):
        for i in tqdm(range(0, len(texts), batch_size), desc="Embedding batches"):
            batch_texts = texts[i:i+batch_size]

            encoded = tokenizer(
                batch_texts,
                max_length=max_len,
                truncation=True,
                padding='max_length',
                return_tensors='tf'
            )
```

5) LSTM on albert embeddings

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input, BatchNormalization
import tensorflow as tf

model = Sequential([
    Input(shape=(1, 768)),

    LSTM(256, return_sequences=True),
    Dropout(0.4),
    BatchNormalization(),

    LSTM(128),
    Dropout(0.3),

    Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(1e-4)),
    Dropout(0.4),

    Dense(num_classes, activation='softmax')
])
```

6) Accuracy and Loss plots of ALBERT + LSTM

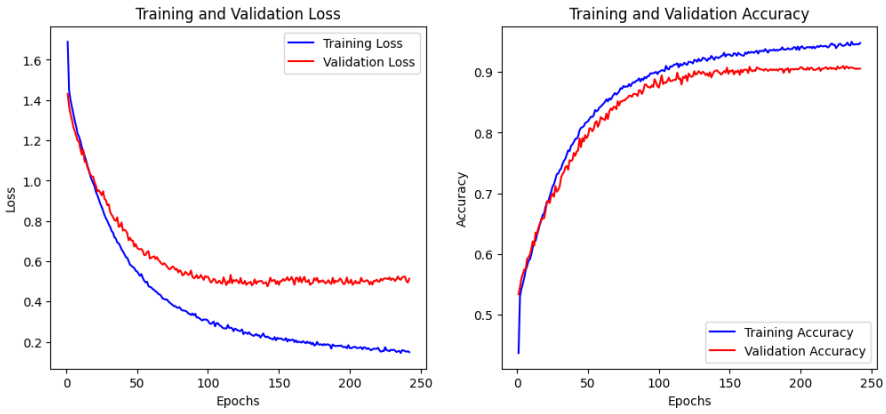


Table 1. Performance Comparison of Machine Learning, Deep Learning, and Hybrid Models

Model	Accuracy	Precision	Recall	F1 Score
Random Forest	0.864	0.900	0.864	0.873
XGBoost	0.861	0.891	0.861	0.868
Bi-LSTM	0.852	0.854	0.852	0.852
GRU	0.858	0.859	0.858	0.858
ALBERT + LSTM	0.900	0.900	0.900	0.900

7. Conclusion:

This report covers a reproducible workflow starting with validation and pre-processing of datasets, followed by baseline machine learning models, deep recurrent baselines, and finally a hybrid ALBERT+LSTM architecture. The design makes replication possible as requirements of the environment, libraries, and core training steps are well defined and the progression of models is transparent and encourages the reasons behind the introduction of the hybrid approach.

The most successful model is the ALBERT + LSTM hybrid that presents the highest reported results and its combination of contextual transformer representations with sequential modelling is conceptually justified. This method is especially suitable to adversarial and unclear cybertext, in which the semantics cannot be faithfully represented with sparse features only. The system is also extensible: more classes may be added with label mapping, the embedding backbone may be updated to newer transformer variants, and the sequence head may be substituted with a lightweight classifier based on the latency limits.

References:

1. https://huggingface.co/docs/transformers/en/index
2. https://spacy.io/usage
3. https://xgboost.readthedocs.io/
4. https://pytorch.org/docs/stable/index.html
5. https://www.nltk.org/
6. https://www.tensorflow.org/
7. https://scikit-learn.org/stable/
8. https://numpy.org/
9. https://pandas.pydata.org/