APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 1 (2025) 521-538
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

Feature Extraction Tool for Email Header using Bidirectional Encoder Representation from Transformers (BERT) Model

Chan Ning Kang[1], Isredza Rahmi A Hamid[1*]

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: rahmi@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.01.029

Article Info
Received: 27 July 2024
Accepted: 19 June 2025
Available online: 30 June 2025

Keywords
Email Header Extraction, Deep Learning,
Natural Language Processing, BERT,
Information Retrieval

Abstract
In the ever-evolving landscape of communication and information retrieval, the effective extraction of key features from emails has become a concern, especially in the field of email forensics. The current email extraction tools lack standardization in the techniques or algorithms in the extraction. Further into the privacy issue is the fear that personal information may leak if improperly handled. In this paper, an advanced email feature extraction tool using BERT model is proposed. The primary objectives include designing an email header extraction tool using an object-oriented approach, developing an email header extraction tool to extract metadata through comprehensive analysis, implementing and testing the proposed tool in terms of system functionality and user acceptance. A Natural Language Processing algorithm using the BERT model constructs the backbone of the analysis approach. The findings underscore the tool's potential to enhance information retrieval and analysis processes in various domains. The average accuracy obtained in this paper is 0.98, illustrating the proportion of correctly identified labels over the total number of labels. In conclusion, the Feature Email Header Extraction Tool offers a valuable contribution to the field, providing a robust algorithm for extracting features from emails, navigating to the use of email forensics and cybersecurity.

1. Introduction

Email serves as a primary communication platform across personal and professional domains in modern-day interaction. While the management of email systems holds substantial importance, particularly in the domain of cybersecurity and information retrieval, email headers could be helpful to email forensics in various ways. In cases where the prevalence of social engineering attacks such as phishing and spoofing involves both technical and social-based techniques, email headers alone may not be sufficient for forensics purposes. There is more to explore in the email body content such as the Uniform Resource Locator (URL) links, attachments, and abnormal activities. Even so, users' privacy is limiting the advanced analysis of email body content. It can be very risky when performing extraction on an organization email, as there's a chance of information leakage. By following the security compliances, this project will use a comprehensive analysis approach to support the overall email header metadata. The current study aims to investigate the many subtleties of email headers, examining the abundance of metadata

they hold and how this information is useful for forensic investigations. Designed for email users, this application stands out as a vital resource for cybersecurity. These email extraction tools give users the ability to proactively protect themselves from prospective risks and strengthen their defenses against many attack vectors.

The existing email feature extraction and analysis encounter various critical issues, including the lack of standardized tools and procedures, limited header modification detection, and challenges pertaining to legality and consent. There is no uniformity and standardized protocols that are verified by the authorities. These inconsistencies result in incomplete and often inadequate analysis, leaving users more vulnerable to potential cyber threats. Moreover, the accuracy and legitimacy of incoming emails are compromised by the current method's inability to identify changes made to email headers. In addition, the extraction and analysis procedure must navigate the limits of data protection laws and privacy laws, which present ethical and legal challenges. Effective threat mitigation strategies and thorough email analyses are hampered by the lack of explicit consent and legal framework compliance, which limits the scope of investigations and makes it difficult to follow ethical guidelines.

To address these challenges, a proposed solution will be developed in the email feature extraction tool using an analytical approach. This tool will extract the metadata information synchronizing with the existing system format with a specialized focus on overcoming the identified issue. Furthermore, the tool will incorporate mechanisms to ensure legal compliance and ethical guidelines. User's privacy should be kept private, regardless of the information available. The objectives of this project are:

i. To design an email header extraction tool using an object-oriented approach.
ii. To develop an email header extraction tool to extract metadata through a deep learning approach using BERT model.
iii. To test the functionality and user acceptance of the proposed tool.

The rest of the paper is organized as follows: Section 2 discusses a literature review on the related work of email extraction tools. It uncovers the fundamentals of project-based knowledge, providing insight into the details of email extraction techniques and algorithms. Section 3 describes the methodology used in the project. Technically, the project uses Object-Oriented Approach (OOP) to ensure continuous improvement by aligning with users' needs. Each phase explains the detailed activities that are to be achieved. Section 4 explains the system analysis and design of the proposed system. Finally, the last section concludes the current work and highlights the future contribution.

2. Related Work

The related work on the project explores the existing literature, research, and methodologies related to the email extraction topic. It serves as a comprehensive examination of prior work conducted by other researchers within similar domains or environments. This helps in understanding the way to approach the metadata extracted, with each tool may behave differently. The primary goals of the related work section are to provide context to the current project, highlight the gaps in existing knowledge, and position the new research within the broader scholarly conversation.

2.1 Email Forensics

Email forensics is the process of scrutinizing the origin and content of an email message with the objective of obtaining evidential information. This includes identifying key details like the true sender, recipient, and the precise date and time of transmission. The investigation employs information-gathering techniques, leveraging the digital trails left on the internet within the networking community. In cases of cybercrime, investigators follow a meticulous study of evidence, integrating a chain of custody to maintain control over the investigation process. Technically, email forensics encompasses two investigative approaches: static and dynamic analysis. Static analysis involves a thorough examination of the email header, which contains rich metadata. This foundational approach provides insights into authentication signatures, email routing, and sender addresses. On the other hand, dynamic analysis delves deeper into the email's content and attachments. The project's focus on feature extraction aligns with the broader goals of email forensics, contributing to the advancement of techniques and methodologies for effective analysis of digital evidence within email communication.

2.2 Static Analysis

Static analysis, as applied in email forensics, involves examining the properties and content of emails without executing them [1]. This approach focuses on studying the metadata appended to the program structure and testing the investigator's skills in utilizing this information. The rapid development in machine learning has introduced algorithms such as Random Forest, Support Vector Machine (SVM), and Naïve Bayes, enhancing the accuracy and efficiency of static analysis. These algorithms have simplified the tasks in the email extraction process. It is widely used in many industries including the email extraction process which greatly reduces human error. The upscale of static analysis is its speed in accessing email headers. When handling a large dataset, static analysis is relatively faster than dynamic analysis, allowing investigators to process and review a substantial volume of emails in a timely manner. Aside from the advantages, static analysis has limited depth of analysis into the email content, potentially missing important information revealed only during the execution phase.

2.3 Deep Learning Approach

Deep Learning is a subset of Machine Learning (ML) that uses neural networks to process and analyze information. Neural networks are machine learning programs or models that make decisions like the human brain. Each neural network contains an input layer, an output layer, and a hidden layer [2]. A neural network that has more than three layers including inputs and outputs can be considered as a deep learning algorithm. These algorithms are fed with a large set of training data which further increases the accuracy of the neural network model. Through the training process, the model will continue to learn the tasks from the processed data, while learning to provide meaningful information that resolves the complex correlation in between. Deep learning has contributed to many tasks such as image and speech recognition, object recognition as well as natural language processing.

2.4 Natural Language Processing (NLP)

Natural Language Processing (NLP) [3] is a subfield of machine learning focused on enabling computers to understand and analyze human language input in text or speech. In the context of information extraction (IE), NLP demonstrates the ability to classify text using keywords and extract meaningful information from unstructured data. This involves breaking down language into components such as sentences, words, and phrases, utilizing techniques like tokenization, part-of-speech tagging, and named entity recognition. By understanding the context and relationships between words, NLP systems can identify key entities, events, and sentiments within a given text. In email information extraction, NLP plays a crucial role in identifying key entities and events mentioned in the text. It enables computers to understand, interpret, and generate human language in a meaningful way.

2.5 Named Entity Recognition (NER)

Named Entity Recognition (NER) is a Natural Language Processing (NLP) technique used to identify and extract entities from text. These entities can include names of people, organizations, locations, dates, quantities, monetary values, percentages, and more. By recognizing these entities, NER helps transform unstructured text into structured data, which can be used for various applications such as information retrieval, question answering, and text summarization. NLP is a broad field that encompasses various processes and analyses of human language, and NER is a specific task within this field. NER focuses on locating and classifying entities such as names, dates, locations, and other significant terms within a given text, facilitating the extraction of meaningful information for further analysis and application. With the advent of deep learning, models such as BERT (Bidirectional Encoder Representations from Transformers) have further advanced the state-of-the-art in NER by leveraging large pre-trained language models that understand context in a sophisticated manner [4].

2.6 Supervised Learning Algorithm

Supervised learning algorithms rely on labeled data, where input data is paired with corresponding output labels or target variables. This technique is used to learn the mapping between input and output. Common supervised learning algorithms include Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), and Neural Networks (NN). Hiszpanski et al. [5] discussed a supervised machine learning approach to analyze sentences and extract solution-based synthesis protocols, facilitating easier management of output due to controlled data variables. Human intervention is required to manually label the data, ensuring the accuracy and relevance of the training dataset.

2.7 Bidirectional Encoder Representation from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained model introduced by Google in 2018, designed to enhance Natural Language Processing (NLP) by understanding the context of words within sentences. Inspired by the Cloze task, BERT employs the Masked Language Model (MLM), which randomly masks tokens in the input and predicts the original vocabulary ID of the masked words based on their context. Google AI Language [6] emphasized the importance of bidirectional pre-training for language representations, reducing the need for heavily engineered, task-specific architectures. Fig. 1 illustrates how an input sequence is transformed into embeddings that combine token, segment, and positional information. For example, the phrase "my dog is cute" is tokenized into individual tokens. Each token is then converted into a token embedding (e.g., E_my, E_dog, E_is, E_cute). Segment embeddings (E_A, E_B) indicate which part of the input the tokens belong to, and position embeddings encode the position of each token in the sequence.

BERT's bidirectional nature allows it to read the input sequence both before and after each token, making it highly effective for numerous NLP tasks. In Named Entity Recognition (NER), for example, BERT can identify and classify entities such as people, organizations, dates, and locations within unstructured text by learning from annotated data. By fine-tuning BERT on labeled datasets specific to information extraction, it can effectively extract structured information from unstructured text.

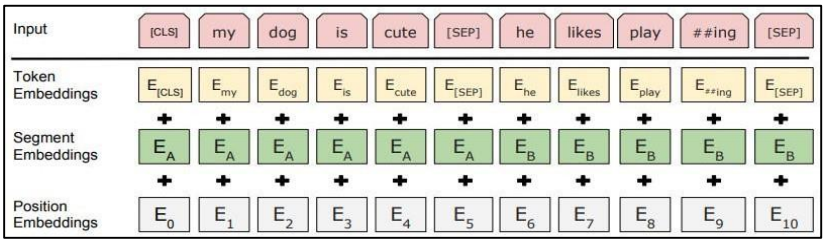

Fig. 1 *BERT model implication*

2.8 Related Work in Email Header Extraction Tools

This section discusses related works in email extraction tools such as Message Header Toolbox by Google, MxToolBox, and Email Header Analysis (EHA).

2.8.1 Message Header Toolbox

Fig. 2 shows the email analysis tool developed by Google [7] and intended to investigate the delivery status of emails sent to or from Gmail accounts. While the tool can receive hop delay information from servers, it falls short as not all hops are displayed. Moreover, Message Header Toolbox does not provide further information for X-headers. As a result, the effectiveness of email extraction is greatly reduced. Additionally, this tool performs static analysis. It parses the email header into human-readable text and extracts relevant information. It also checks DomainKeys Identified Mail (DKIM), Sender Policy Framework (SPF), and Domain-based Message Authentication Reporting and Conformance (DMARC) authentication, validating the reliability and integrity of the email.

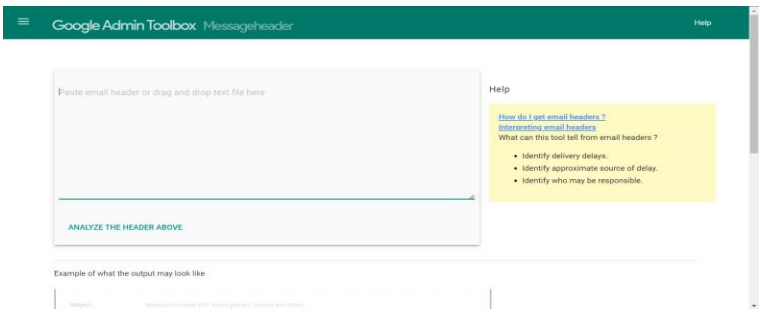

Fig. 2 *Message Header Toolbox*

2.8.2 MxToolBox

It is a robust tool with comprehensive features for managing and extracting email headers. MxToolBox [8] offers many additional features such as blacklist monitoring, mail server diagnostics, and DNS-related services as shown in Fig. 3. However, due to the technical nature of the features offered, MxToolBox may seem a bit complex for beginners, requiring a foundational understanding of email architecture.

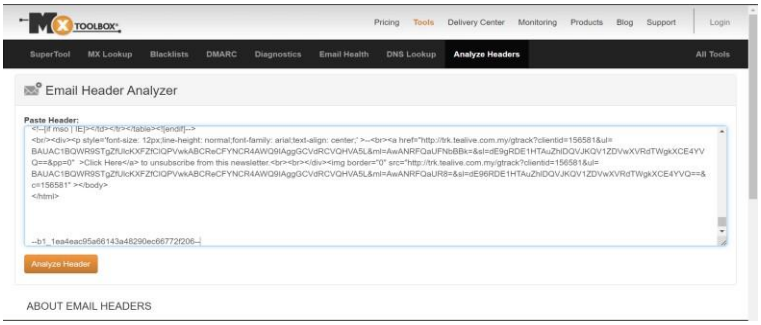

Fig. 3 *MxToolBox*

2.8.3 Email Header Analysis (EHA)

Email Header Analyzer (EHA) [9] is an open-source tool available in Github written in Python and several languages for front-end development like HTML, JavaScript, and CSS. It takes the input of the email headers, analyzes and extracts information as in Fig. 4. The key features of this tool are the identification of hop delays, source of the email, checking on the SPF record, DKIM signature, MIME-version, and Message ID. This tool is intended to promote the authenticity of email. It compares the DKIM signature with its associated private key stored on the actual sending server 103.52.18.228/pepipost.net server. If the DKIM signature is not passing correctly, there may be an attempt of email tampering during the process.

Fig. 4 *Email Header Analysis (EHA)*

2.9 Comparison of Proposed Tool with the Current Tools

Many tools are developed in order to handle the email analysis process. The evolution of technology from regular expression (Regex) to a deep learning approach showcases the importance of email studies. Although regex is effective for pattern recognition, which makes it faster and easier to extract email text field information, regex has limitations when dealing with variations or dynamic patterns that may occur in email headers. Moving into deep learning algorithms, in natural language processing (NLP) context, provides a more adaptive and flexible approach to feature email extraction. These algorithms are trained using a dataset containing thousands of data. It can capture complex patterns and contextual information to detect the behavior of the text. This ability makes them robust in handling different patterns in email headers/bodies. As a result, many existing tools are implementing machine learning in email analysis and incorporating deep learning approaches to refine the methodology. It enhances the overall performance, accuracy, and effectiveness of email header extraction.

Table 1 illustrates the comparison between the Feature Email Extraction Tool with other existing tools. The hop delay refers to the time an email takes to travel from one server to another in the email delivery path. Message Header Toolbox by Google has its limits in capturing the hop delays as not all hops are displayed. MxToolBox and EHA on the other hand display comprehensive hop delay information. Besides, the authentication of the proposed tool is not developed. The SPF, DMARC, and DKIM are the authentication keys needed to prove an email's validity.

Table 1 *Comparison between existing systems and the proposed tool*

No	Feature/Tools	Hop delay	Headers	Body	Authentication	URL links	X-headers
1	Message Header Toolbox by Google	✘	✓	✘	✓	✘	✘
2	MxToolBox	✓	✓	✘	✓	✘	✓
3	Email Header Analysis (EHA)	✓	✓	✘	✓	✘	✘
4	Proposed Tool	✘	✓	✘	✘	✘	✓

3. Methodology

In the context of software development, the focus lies on the methodology used for guideline the overall development progress which ensures a high-quality tool with comprehensive procedures. The methodology resolves into two categories, structured and object-oriented (OO) approaches, while the implementation varies depending on the project environment. The structured approach, characterized by a step-by-step, linear progression through various development phases, provides a foundation for organized and systematic software development. Simultaneously, the project will heavily leverage the object-oriented paradigm to enhance modularity, maintainability, and extensibility.

The proposed project is centered around Python as the programming language further facilitates the integration of object-oriented principles. Python's support for object-oriented programming, coupled with its

emphasis on readability and simplicity, makes it well-suited for object-oriented projects. The object-oriented approach makes use of the Unified Modeling Language (UML) and diagrams to visualize the code. In this life cycle, the focus on object-oriented principles extends to the determination of system objects, successfully distinguishing each essential object associated with the project. Fig. 5 depicts the phases in Object-Oriented Software Development.

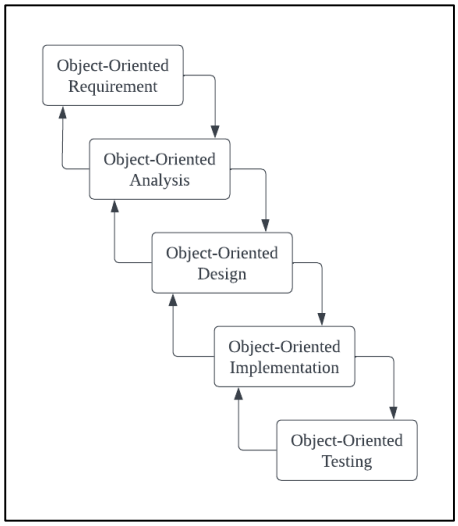

Fig. 5 *Object-Oriented Software Development*

3.1 Object-Oriented Requirement

The focus of this phase is to gather requirements and prepare information for later analysis. This phase serves as the foundation for subsequent phases by providing problem statements, scope, and goals of the project. Key questions addressed include: "What services are available?", "What are the necessary data?", and "What are the expected inputs and outputs?". Answering these questions offers a comprehensive understanding of the system's functionality and user requirements.

The gathered requirements involve both software and hardware specifications, detailing the minimum requirements to develop the email extraction tool. This ensures a smooth workflow during development and enhances performance in executing tasks. By defining software and hardware specifications upfront, the project aligns with the defined requirements, ensuring the development process progresses efficiently.

3.2 Object-Oriented Analysis

The object-oriented analysis (OOA) is crucial in software development, aiming to gather important information needed to draft the overall system. The information gathering process will involve the end-users as well as the stakeholders in identifying the requirements for the software system. During the analysis phase, it is crucial to identify potential use cases, user interactions, and system requirements. This includes understanding the specific requirements of email forensics, such as handling different email formats, accommodating variations in header structures, and ensuring the tool's adaptability to diverse email sources. Refining and organizing the information pieces provide insight into the system structure, acknowledging the user requirements are achieved. To manage the project effectively, a detailed project timeline is established and presented through a Gantt chart. The Gantt chart serves as a valuable tool for planning, scheduling, and tracking progress, ensuring adherence to the predetermined timeline by the development team. Further in the analysis process, UML is used to construct the case diagram and other object model diagrams, assisting in a deeper understanding of the system.

3.3 Object-Oriented Design

This phase shifts from defining the system's requirements to creating a detailed blueprint of the software solution based on the identified objects, their attributes, and their interactions. This involves conceptualizing the email extraction process, outlining system flow, modules, and relationships between email metadata. The design phase contributes to UML diagrams and wireframes developed using tools like Lucidchart, draw.io, and Figma. Additionally, it plays a crucial role in understanding how the BERT algorithm, known for its natural language processing capabilities, integrates into the tool, enhancing its intelligence and information extraction proficiency. The design ensures user-friendliness, especially for entry-level users, with a visually appealing and intuitive interface for better user experience.

3.4 Object-Oriented Implementation/Coding

The implementation phase prioritizes the development of the email extraction tool, focusing on practical coding rather than extensive documentation. The key activities include creating classes, implementing methods, defining attributes, and establishing the relationships outlined in the design phase. Recognizing the complexity of the project, extensive research is conducted to explore the implementation of algorithms into actual code. The project is coded in Python due to its rich libraries and frameworks, particularly suitable for machine learning and natural language processing. Visual Studio Code is chosen as the coding platform, seamlessly supporting Python and enhancing development with features like syntax highlighting, debugging tools, and version control integration. Coding in this phase would also consider best practices in machine learning, such as integrating the BERT model effectively, managing dependencies, and optimizing performance. Throughout the implementation, hardware and software requirements are crucial, considering the project's long-term execution demands high power, memory, and processing unit usage. Using outdated software or low-spec hardware may damage the system during execution.

3.5 Object-Oriented Testing

The testing phase follows the development of the email extraction tool and focuses on ensuring its reliability, functionality, and performance. Two types of tests, alpha testing, and beta testing are conducted. Alpha testing is an initial internal evaluation conducted in a controlled development environment to ensure the stability and functionality of the tool, addressing any identified issues. Beta testing takes the software into a real-world environment, evaluating its ability to precisely extract and process information from emails, including metadata like addresses, timestamps, and subjects. System testing would involve comprehensive evaluations of the entire email extraction tool, verifying its ability to handle various email formats, addressing potential errors, and ensuring the accuracy of extracted metadata.

4. Results and Discussion

This section explains the analysis and design involved in the topic. The analysis of the project dives into an understanding of functional and non-functional requirements that are incorporated to lay the foundation for the design of the feature email extraction tool. Besides, the user requirement analysis discussed in this section aims to capture the needs and expectations of end-users. Further in this section is the design of the feature email extraction tool. The design is the process of translating gathered requirements into a blueprint that outlines the structure and components of the system. It is “how” the system will deliver the information to users.

4.1 General System Architecture

The system architecture is designed to provide a high-level overview of the components and workflow in the email header extraction tool. It illustrates the relationship between the system, users, and the admin, involving the flow of tasks to demonstrate how the system behaves during the activities. Fig. 6 showcases the general system architecture of the email header extraction tool. Users can operate in anonymous mode or register for more personalized features. Once logged in, users can paste raw email headers or upload files in .eml or .txt format. The system then processes these inputs, by running the BERT model to identify the information needed and display the result on the user interface. The administrative functions include managing user accounts, updating tools, and monitoring the system for performance and security. Admins have access to a login interface where they can perform CRUD operations, manage system updates, and oversee overall system functionality through monitoring tools.

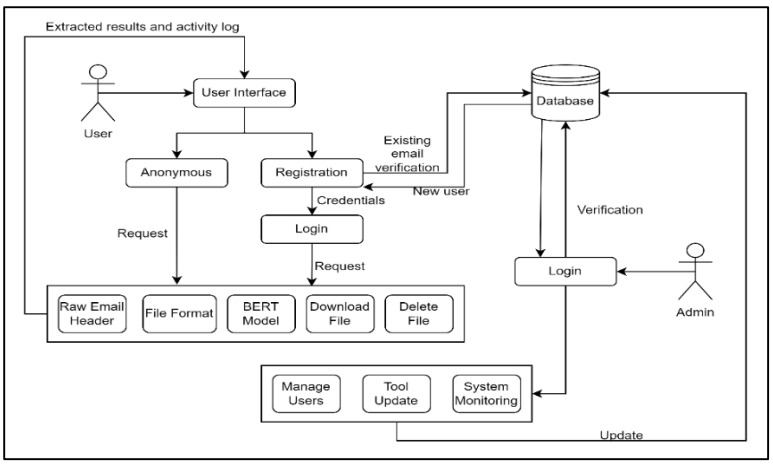

Fig. 6 *General System Architecture for Email Extraction Tool*

4.2 Unified Modelling Diagram (UML)

In the Object-Oriented approach, UML is frequently used to visualize the software diagram utilizing a collection of diagrams. UML is divided into two distinct groups, structural and behavioral diagrams. Structural diagrams implement the key objects of a system and the relationships between them. Examples of structural diagrams are class diagrams, package diagrams, object diagrams, and Component diagrams. The behavioral diagrams are responsible for defining the history of objects over time and the communications among them to accomplish goals. This includes activity diagrams, use case diagrams, and sequential diagrams.

4.2.1 Use Case Diagram

Within the behavioral diagrams, the use case diagram illustrates the interactions between users (actors) and a system to achieve specific goals (use cases). The use case is represented using oval shapes, while the actor is represented using a stickman. Use case diagrams are particularly easy to understand, even for non-technical stakeholders, as they do not involve detailed system functionality. It is useful in capturing and communicating the functional requirements clearly and concisely. Fig. 7 and Fig. 8 represent the user and admin use case diagrams respectively. The user use case diagram illustrates the functionality he can perform, including login and registration, extracting email headers, viewing activity logs, downloading and deleting files, and logout the account. Similarly, the admin use case diagram outlines the following functionality, login, viewing registered user statistics, creating, updating, deleting users, viewing reports, and logging out.

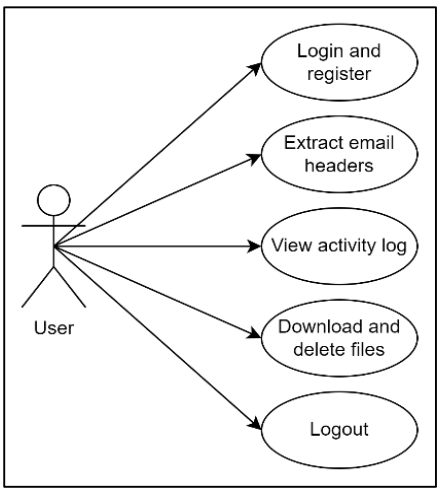

Fig. 7 *Use case diagram (User)*

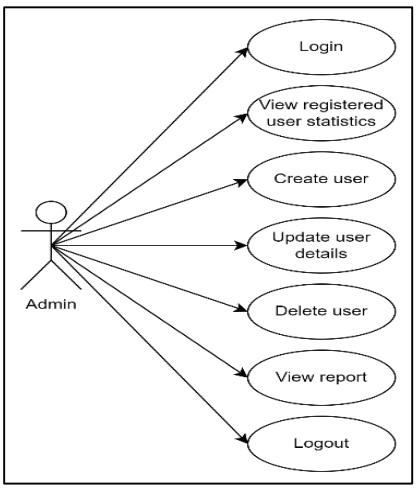

Fig. 8 *Use case diagram (Admin)*

4.2.2 Sequential Diagram

Sequential Diagram is a diagram that depicts how objects interact with each other in a particular sequence of time. It provides a detailed visualization of the flow of messages between objects in a system, showing the order in which these interactions occur. The sequential diagram is unique as it has additional lifelines that show how objects collaborate over time to achieve a particular functionality. Fig. 9 and 10 demonstrate the flow of the system with user and admin interactions.

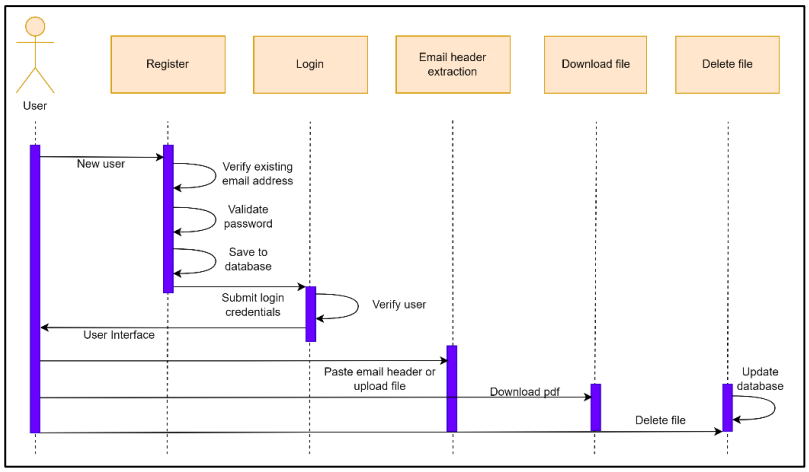

Fig. 9 *Sequential Diagram (User)*

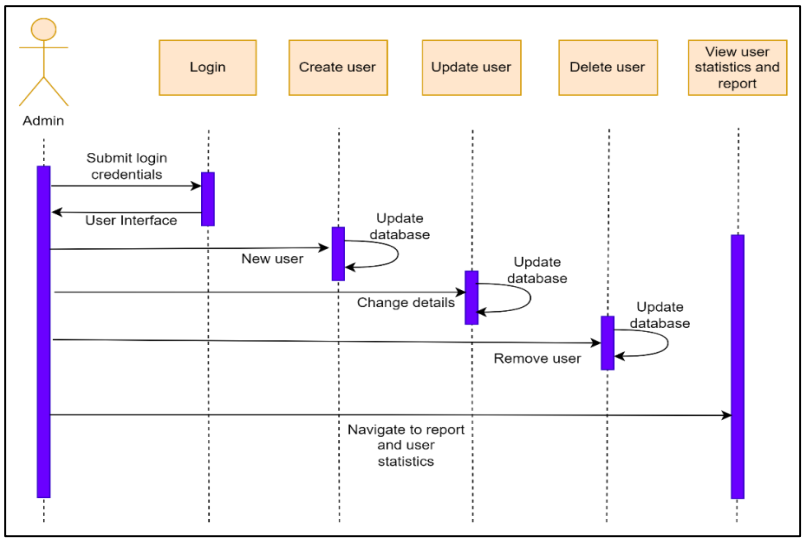

Fig. 10 *Sequential Diagram (Admin)*

4.2.3 Activity Diagram

An activity diagram is a behavioral diagram in a Unified Modelling Language (UML) diagram that describes the dynamic aspects of a system. Fig. 11 illustrates the dynamic aspects of the system, depicting optional login steps for users who want to save their analysis logs. Users can choose to remain anonymous and proceed without logging in. The user interface includes a section for pasting email header text, and upon clicking the "Analyze" button, the system utilizes the BERT model for email extraction. A report is generated based on the analysis, allowing users to view or download it for future reference. Next, Fig. 12 is the activity diagram for the Admin. The admin is responsible for handling user creation, update, and deletion in the email extraction tool. Being the top-level management, the admin is provided the privilege to view the reports but is limited to the report file name. This is because there may be sensitive information contained within the emails.

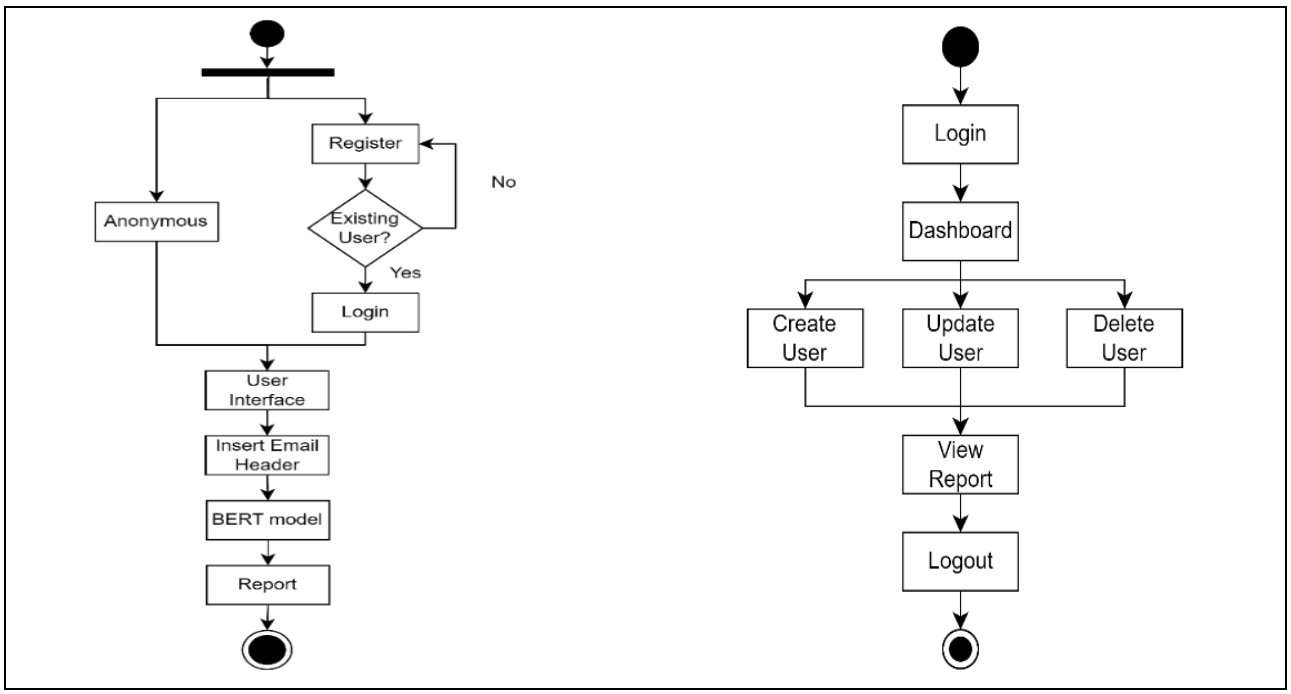

Fig. 11 *Activity Diagram (User)*

Fig. 12 *Activity Diagram (Admin)*

4.2.4 Class Diagram

A use case diagram is a type of structural UML diagram. It describes the structure of a system by showing its classes, attributes, operations (or methods), and the relationships among objects. In Fig. 13, the user will manage the files by performing the download and deletion. Moreover, the user will be able to use the BERT model for extraction that runs behind the interface. It will then generate a report, enabling the user to view it. Admin's operation is to login and logout, and he can manage users by performing create, update, delete on users.

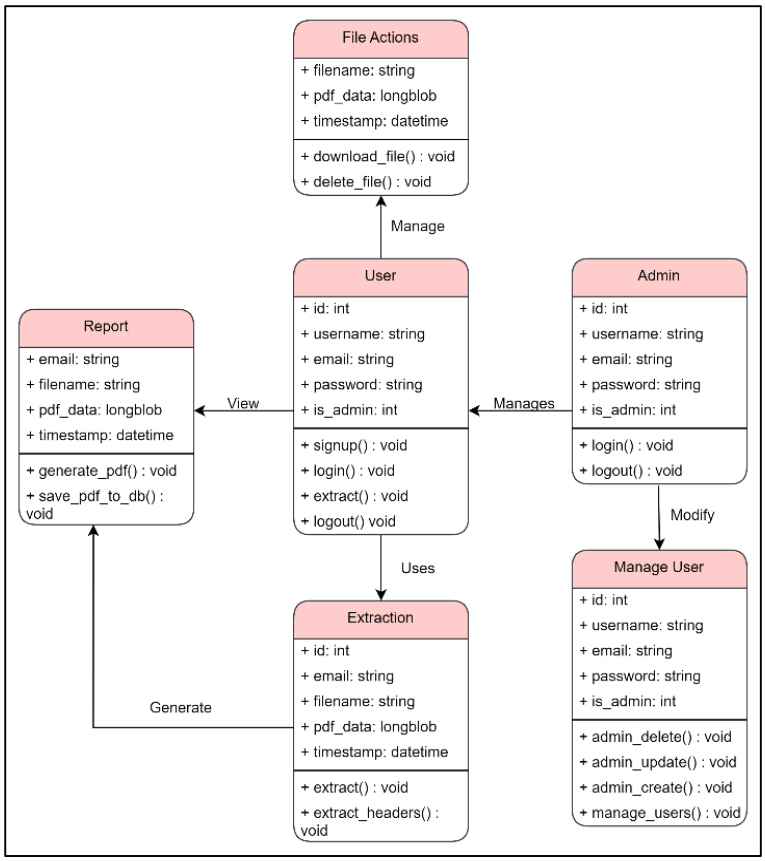

Fig. 13 *Class diagram*

4.3 Functional Requirements

Functional requirements refer to what a program needs to do. The need for functional requirements provides insight into the functionality aspects. The system must perform with specific features that meet the needs of its users as shown in Table 2. The tool must proficiently parse incoming emails to extract critical information like timestamps, sender details, and authentication signatures, ensuring accuracy. A robust algorithm and analysis techniques, including static and dynamic analyses, are employed to extract metadata from both the email header and body content. On the other hand, the BERT algorithm is utilized to process and understand this information. The results are then presented to users in a clear and comprehensive format, rearranging scattered information for easy interpretation. Additionally, users have the option to download the final output for future reference.

Table 2 *Functional Requirements for Email Extraction Tool*

No	Module	Functionality
1	Email Parsing	Extract relevant information from the incoming emails
2	Feature Extraction	BERT algorithm and techniques to analyze email content and extract information
3	Result Presentation	Extracted features should be presented to users in a clear and comprehensible format, such as a detailed report or diagrams
4	Register and Login	The system should provide the optional register and login interface to users.

4.4 Non-functional Requirements

The non-functional requirement analysis is the study of the overall system behavior, performance, and constraints for the system as depicted in Table 3. These requirements, though not directly tied to specific functionalities, are crucial for the system's success and usability. They address aspects such as performance, reliability, and security. The system is expected to efficiently process a substantial volume of emails within an acceptable timeframe, establishing performance benchmarks for responsive feature extraction. High reliability, minimal downtime, and robust error recovery mechanisms are essential, ensuring users consistently experience accurate extraction without disruptions. Given the confidentiality of email content, the tool must prioritize strong

security measures, employing encryption to secure student privacy and implement well-structured access controls to prevent unauthorized access.

Table 3 *Non-Functional Requirements for Email Extraction Tool*

No	Requirement	Description
1	Performance	• Able to process a specified volume of emails within an acceptable time frame. • Responsive and timely despite the workload increase.
2	Reliability	• High reliability with minimal downtime. • Ensures consistency rely on the tool for accurate feature extraction.
3	Security	• Keep user data hidden in the database. • Strong encryption mechanism implementation. • Prevent unauthorized access.

4.5 Implementation of the BERT model

To adjust the BERT model to fit the email extraction task, the steps are divided into three main topics, dataset preprocessing, fine-tuning, as well as model predictions and evaluations which can be further categorized into smaller tasks. The development of the BERT model is conducted using Google Colab, which requires a foundational knowledge of Python.

4.5.1 Dataset Preprocessing

The dataset contains the data needed to be fed into a model to help it understand the area of study. Typically, a dataset involves thousands of data points, and having more data generally leads to higher accuracy as more references are provided for the model. A dataset, commonly in CSV file format, has several columns called 'features.' These features are the source of information used to train the model to evaluate and understand the data. This project uses a random sample of 2,000 entries from the Enron Email Dataset [10], further processed to fit the model's needs. The keywords 'data processing' and 'data cleaning' refer to the adjustment of the dataset for model training.

Data annotating is the process of tagging tokens in chunks. The IOB format, sometimes called BIO format, is popular for annotating tokens. The BIO (B: Beginning, I: Inside, O: Outside) format is useful in Named Entity Recognition (NER) tasks as it labels tags on each subword. For example, ['To', ':', 'james', '@', 'mail', '.', 'com'] becomes ['O', 'O', 'B-To', 'I-To', 'I-To', 'I-To', 'I-To']. Here, the focus is on 'james@mail.com,' with 'james' annotated as 'B-To' to represent the beginning of the 'To' entity, while 'I-To' represents the tokens inside the entity. Fig. 14 shows the handling of the BIO tagging by capturing the entities like Message-ID, Date, From, etc. The BIO tags were paired with each token, labeling ‘B-‘ and ‘I-‘ for the information required and ‘O’ for the information outside the entities. In this project, the email content will be labeled as ‘O’ since the aim is to focus on the email header. In other words, given that any entities that appear in the body content (which is unlikely to happen) will always be ignored as it is not part of the email headers.

```
# Add columns for ner tags
for key in allkeys:
    ner_tags_col = []
    for index, row in df_parsed.iterrows():
        tokens = row[key]
        ner_tags = ['O'] * len(tokens)
        if key != 'Body':
            for i, token in enumerate(tokens):
                if token in [key, ':'] and i < 2:
                    ner_tags[i] = 'O'
                elif i == 2:
                    ner_tags[i] = 'B-' + key
                else:
                    ner_tags[i] = 'I-' + key
        ner_tags_col.append(ner_tags)
    df_parsed['ner_tags (' + key + ')'] = ner_tags_col
```

Fig. 14 *BIO tagging*

4.5.2 Fine-tuning BERT

Fine-tuning is a process to train a pre-trained model on a specific task to improve its performance for that task. Fig. 15 illustrates the process of fine-tuning BERT on email extraction. Hyperparameter is a word that is often used in machine learning, that helps to govern the overfitting and underfitting behavior of the model. Epochs, gradient accumulation steps, and learning rate belong to the hyperparameter.

- **Epochs** – the number of times the model runs. Running epochs of 3 is preferable as more epochs would cause overfitting.
- **Gradient accumulation steps** – the number of steps to accumulate gradient before performing a parameter update. This is useful when the batch size is limited by memory constraints. If the parameter is updated every step, it might cause unstable training, particularly when the batch size is small.
- **Learning rate** – the step size at each iteration while moving towards a minimum of the loss function. Taking an example, when playing badminton, swinging the racket too early or too late will miss the shuttlecock. But if you start with a slow tempo to a point where you feel comfortable with the speed, you'll perform the best. More epochs are needed when the learning rate is small due to the changes made to the weights will be small. In contrast, large learning rates attract few epochs due to high speed. When the learning rate is too high, model convergence is very fast while on the other hand when the learning rate is too small, the process might be stuck at some point [11].

```
num_epochs = 3
gradient_accumulation_steps = 2

checkpoint_dir = './bert_ner_model'

# Create the directory if it doesn't exist
os.makedirs(checkpoint_dir, exist_ok=True)

def save_checkpoint(epoch, model, optimizer, loss, checkpoint_dir):
    checkpoint_path = f"{checkpoint_dir}/checkpoint_epoch_{epoch}.pt"
    torch.save({
                'epoch': epoch,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
                'loss': loss,
                }, checkpoint_path)
    print(f"Checkpoint saved at {checkpoint_path}")

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    optimizer.zero_grad()

    for step, batch in enumerate(dataloader):
        batch = tuple(t.to(device) for t in batch)
        input_ids_batch, attention_masks_batch, labels_batch = batch

        outputs = model(input_ids=input_ids_batch, attention_mask=attention_masks_batch, labels=labels_batch)
        loss = outputs.loss
        loss = loss / gradient_accumulation_steps
        total_loss += loss.item()

        loss.backward()

        if (step + 1) % gradient_accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

    avg_train_loss = total_loss / len(dataloader)
    print(f"Epoch {epoch+1}/{num_epochs}, Average Training Loss: {avg_train_loss:.4f}")

    save_checkpoint(epoch, model, optimizer, avg_train_loss, checkpoint_dir)
```

Fig. 15 *Fine-tuning BERT*

4.5.3 Model Predictions and Evaluations

To ensure the model's performance, the model is put to make predictions and evaluations to observe the overall accuracy level. Fig. 16 is the code implementation to perform the predictions by iterating through the test dataset using a DataLoader, moving each batch to the appropriate device. Fig. 17 is the code to generate and print the classification report for evaluation as shown in Fig. 18. Either in machine learning or deep learning, the model predictions and evaluations involve precision, recall, f1-score, and support [12]

- **Precision –** is the positive value gained from the prediction. Equation 1 shows the formula of precision. It answers the question: "Of all the entities that were identified as a particular type, how many were actually correct?" In Fig. 18, the precision for 'B-To' is 0.96. This means that out of all instances predicted as 'B-From', 96% were correct.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (1)$$

- **Recall –** is also known as sensitivity. It is the relevance of gained occurrence. Equation 2 is the recall formula. It answers the question: "Of all the entities that actually are a particular type, how many did the model correctly identify?". In Fig. 18, the recall for 'B-To' is 0.99. This means that out of all actual 'B-To' instances, the model correctly identified 99%.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (2)$$

- **F1-score –** is the weighted average of precision and recall. It is a more comprehensive metric that balances both precision and recall, especially useful when the class distribution is imbalanced. The formula is shown in Equation 3. In Fig. 18, the f1-score for 'B-To' is 0.97. This combines the precision and recall for 'B-To' into a single metric.

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

Support is the number of actual occurrences of each class in the dataset. In Fig 16, the support for 'B-From' is 1944. There were 1944 instances of 'B-To' in the dataset.

```
[ ]  model.eval()

     # Lists to store true labels and predictions
     true_labels = []
     pred_labels = []

     # Iterate through the test dataset
     for batch in test_dataloader:
         batch = tuple(t.to(device) for t in batch)
         input_ids_batch, attention_masks_batch, labels_batch = batch

         with torch.no_grad():
             outputs = model(input_ids=input_ids_batch, attention_mask=attention_masks_batch)

         logits = outputs.logits
         predictions = torch.argmax(logits, dim=2)

         # Move tensors to CPU and convert to numpy arrays
         predictions = predictions.detach().cpu().numpy()
         labels = labels_batch.detach().cpu().numpy()

         for i in range(len(labels)):
             true_labels.extend(labels[i])
             pred_labels.extend(predictions[i])
```

Fig. 16 *Prediction*

```
from sklearn.metrics import classification_report

# Generate the classification report
report = classification_report(true_labels_flat, pred_labels_flat, digits=4)
print(report)
```

Fig. 17 *Evaluation*

	precision	recall	f1-score	support
B-Bcc	0.0000	0.0000	0.0000	121
B-Cc	1.0000	0.3468	0.5150	124
B-Content-Transfer-Encoding	0.9845	0.9909	0.9877	1984
B-Content-Type	0.9909	0.9919	0.9914	1985
B-Date	1.0000	0.9975	0.9987	1997
B-From	0.9861	0.9920	0.9890	1997
B-Message-ID	0.9512	0.9955	0.9728	1997
B-Mime-Version	0.9904	0.9919	0.9912	1985
B-Subject	0.3733	0.9874	0.5418	1424
B-To	0.9603	0.9949	0.9773	1944
B-X-FileName	0.9814	0.9850	0.9832	1933
B-X-Folder	0.9540	0.9850	0.9693	1938
B-X-From	0.8918	0.9647	0.9268	1982
B-X-Origin	0.9897	0.9891	0.9894	1935
B-X-To	0.8724	0.9732	0.9201	1975
B-X-cc	0.0000	0.0000	0.0000	110
I-Bcc	0.8309	0.7679	0.7982	2490
I-Cc	0.7797	0.7525	0.7658	3099
I-Content-Transfer-Encoding	0.9920	0.9819	0.9869	2152
I-Content-Type	0.9974	0.9952	0.9963	24333
I-Date	0.9998	0.9997	0.9997	33464
I-From	0.9986	0.9885	0.9935	19811
I-Message-ID	0.9991	1.0000	0.9995	51669
I-Mime-Version	0.9904	0.9919	0.9912	3970
I-Subject	0.9177	0.9459	0.9316	11272
I-To	0.9591	0.9908	0.9747	37556
I-X-FileName	0.9884	0.9937	0.9911	14283
I-X-Folder	0.9906	0.9977	0.9941	32889
I-X-From	0.9268	0.9603	0.9432	13393
I-X-Origin	0.9856	0.9825	0.9841	3896
I-X-To	0.8671	0.9627	0.9124	25751
I-X-cc	0.9612	0.0716	0.1332	3465
O	0.9986	0.9928	0.9957	713540
accuracy			0.9868	1022464
macro avg	0.8821	0.8655	0.8529	1022464
weighted avg	0.9884	0.9868	0.9860	1022464

Fig. 18 *Prediction and Evaluation (result)*

4.6 Implementation of the User Module

Fig. 19 shows the user interface after a user has logged into the system. The technology primarily relies on Visual Studio Code, which integrates HTML, CSS, JavaScript, and Python. On this page, users can paste the email header directly into the provided text area. Alternatively, users can upload a file with a .eml or .txt extension that contains the email header. They can then select multiple entities to extract, such as Message-ID, From, Date, and Subject, from a dropdown menu. By clicking the "Extract Headers" button, the tool processes the input, extracts the selected entities, and generates a detailed report in a new tab. On the right side, the Activity Log displays the filename and associated timestamp, allowing users to view the date and time of the respective files. It also provides functions to download and delete files. Fig. 20 shows the example output from the email extraction tool.

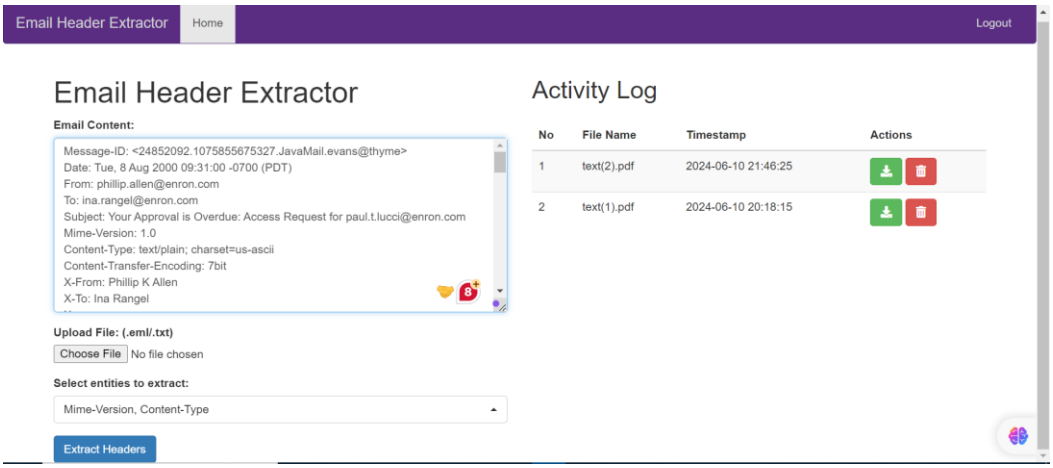

Fig. 19 *User Interface*

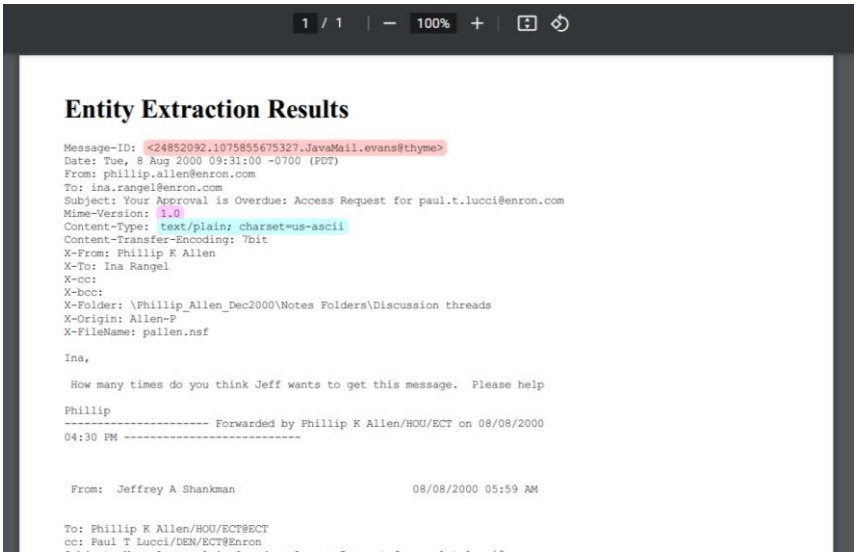

Fig. 20 *Example of Extracted Results*

4.7 Implementation of the Admin Module

The admin module consists of a Dashboard, Manage Users, and Report page. Each page carries out different tasks, but together they serve to perform CRUD (Create, Read, Update, Delete) actions for managing users and the system. Visual Studio Code is the development platform that integrates these pages. The functionality and appearance are maintained using HTML, CSS, JavaScript, and Python.

4.7.1 Admin Dashboard

Fig. 21 shows the admin dashboard that displays the total number of users registered in this email extraction system. This includes the admins and users registered. The green box and yellow box are the active users and admins respectively to distinguish them better. The Recent Activity table will display the ten most recent user activities.

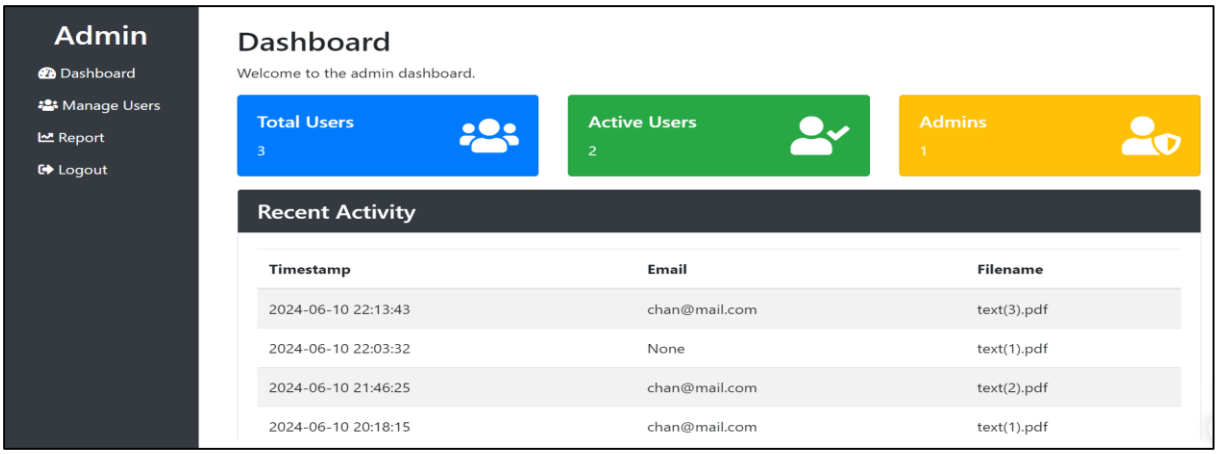

Fig 21 *Admin Dashboard*

4.7.2 Manage Users

Within the Manage Users page as shown in Fig. 22, the admin has the privilege to create, update, and delete users or other admins. Whether for user creation, updating, or deletion, these actions share a common process, fetching requests from and to the database. These modifications involve collecting user information and updating the database accordingly.

Fig. 22 *Manage Users Page*

4.7.3 Report

The Report page is where the admin can view the list of activities completed by users. Each entry in the table shows the user's email, the filename of the generated report, and the timestamp indicating when the file was created. This allows the admin to monitor overall activity while maintaining confidentiality. The design ensures that sensitive information within emails is protected, as the admin can track activities without accessing the content of the emails themselves. The detailed logs help the admin govern user interactions and ensure compliance with security policies. The illustration of admin's report page is shown in Fig. 23.

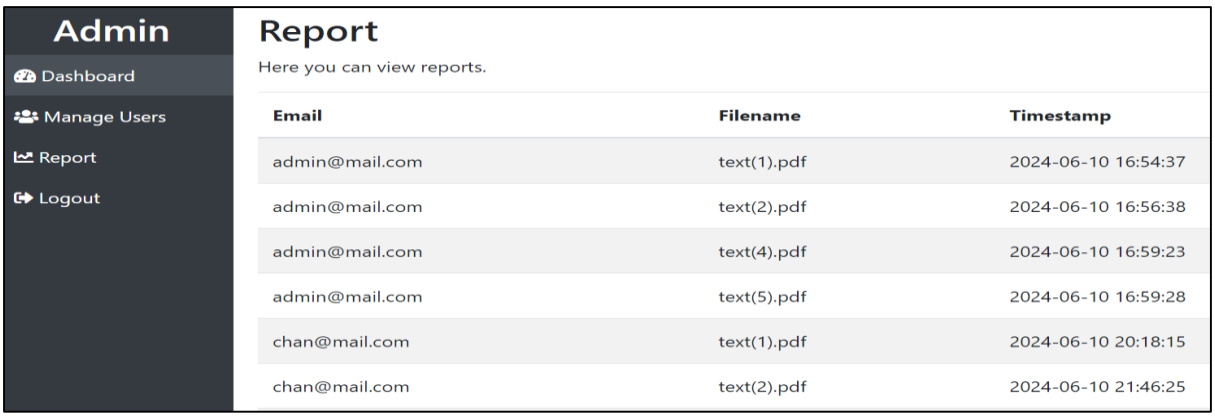

Fig. 23 *Admin Report Page*

4.8 Test Plan

The testing phase is a crucial step in software development to ensure the product's reliability before market release. The test plan outlines the strategy, approach, schedule, and scope for comprehensive testing, identifying potential threats and validating system functionality. Table 4 consists of two test categories, user and admin. Each is associated with the test setup, describing the navigation to perform the specific activity. The expected result is the output that must be conducted by the system. The actual result will be labeled "Pass" or "Fail" based on the output obtained.

Table 4 *Test Plan for Email Extraction Tool*

Test Category	Test case	Test Setup	Expected Result	Actual Result
User	Registration	• Register with a new email, username and password. • Click on the "Signup" button	The system should direct the user to login page	Pass
	Login	• Enter user email and password Click on the "Login" button	The system will direct the user interface	Pass
	Extract information	• Paste the raw email header or upload the file. • Select the entities for extraction. • Click on the "Extract Headers" button	The report will be displayed in a new tab with highlighted information representing the corresponding entities and the information should be accurate.	Pass
	Download files	• Click on the download icon near the Activity Log box	The file will be downloaded to the user's local computer.	Pass
	Delete files	• Click on the delete icon near the Activity Log box	The file will be deleted from the database and removed from the user interface.	Pass
	Login	• Enter admin email and password. • Click on the "Login" button	The system will direct the admin to the admin interface and not the user interface	Pass
Admin	View report	• Navigate to the "Report" section	Admin can view the timestamp and the filename created in the table	Pass
	Create user	• Go to the "Manage User" section. • Enter a new email, username, and password	A new user is added to the registered user list	Pass
	Update user	• Go to the "Manage User" section. • Enter a different email, username, and password	The updated information should be displayed in the table	Pass
	Delete user	• Go to the "Manage User" section. • Click on the delete icon	The system should remove the record of the selected user	Pass

4.9 User Acceptance Testing Form

User Acceptance Testing (UAT) is a critical phase involving end-users or stakeholders validating a software tool against predefined criteria and procedures. The UAT form is instrumental in facilitating user engagement for the final validation of the feature email extraction tool. Users participate in testing, providing insights into the tool's usability, functionality, and alignment with their needs, ensuring it meets their expectations. Table 6 shows the user acceptance testing form for email extraction tool. The form includes a rating scale from 1 (dissatisfy) to 5 (strongly satisfy), allowing users to express their level of satisfaction with various aspects of the tool. This rating system provides a quantitative measure of user satisfaction and helps in identifying areas that may require improvement or further refinement.

From Table 6,all 10 users rated 5 (strongly satisfy) for which the user can delete and download files,the user is not allowed to access the admin interface, and the admin can create, update,and delete user. Some may think that the result extracted is not truly informative and useful in digital forensics and the system is not quite user-friendly with clear instructions, resulting in 3 ratings each from 2 people.

Table 6 *User Acceptance Testing Form for Email Extraction Tool (Likert scale form 1-strongly dissatisfy to 5-strongly satisfy)*

No	Acceptance Requirement	Test					
		Scale from 1 to 5					Total
		1	2	3	4	5	
1.	User can register and login to the system	0	0	0	4	6	10
2.	User can obtain desired information from selected options	0	0	1	5	4	10
3.	User can delete and download file	0	0	0	0	10	10
4.	User can view activity logs with filenames and timestamp	0	0	0	1	9	10
5.	User is not allowed to access the admin interface	0	0	0	0	10	10
6.	Admin can create, update, and delete user	0	0	0	0	10	10
7.	Admin can view the report generation time with the associated user	0	0	0	5	5	10
8.	Admin can view the most recent activities of users	0	0	0	2	8	10
9.	The result is informative and useful in digital forensics	0	0	2	4	4	10
10.	The system is user-friendly with clear instructions	0	0	2	3	5	10

5. Conclusion

In conclusion, this project has successfully developed a sophisticated Email Header Extraction Tool using an Object-Oriented and deep learning approach, focusing on extracting key information from email headers. The findings highlight the significance of accurate feature extraction in the field of email forensics and cybersecurity. By integrating the BERT algorithm, the project demonstrates an advanced method for analyzing, understanding, and extracting important metadata from email headers. This ensures both accuracy and efficiency in the process.

The tool has its strength in terms of its automation, which allows for the rapid and accurate extraction of metadata without manual intervention. This automation enhances efficiency and ensures consistency in the results. The tool's capability to understand the context within the email makes the extraction process more dynamic and accurate. Additionally, it provides a user-friendly interface that allows users to either paste email headers or upload files for processing. The system's ability to generate detailed reports and maintain an activity log further enhances its usability and effectiveness. Recognizing the sensitivity of email content, it is crucial to implement strong security measures and privacy policies to safeguard user information. Future work could focus on enhancing these security features, as well as expanding the tool's capabilities to handle a wider range of email formats and additional metadata types.

In the future works, the tools could be further developed by improving the maximum length sequence of the BERT algorithm which is commonly 512 tokens [13]. BERT is inherited from the transformers' architecture, which uses self-attention, feed-forward layers, residual connections, and layer normalization as their foundational components. The fact that transformers themselves are aggressive and there is a significant decrease in performance when the input sequence length is longer than 512 tokens, it is to prevent low quality output. Furthermore, BERT's self-attention model complexity is $O(n^2)$, meaning that a longer input requires more resources to fine-tune the model.

Additionally, expanding the tool to analyze the email body content can significantly improve BERT's ability to understand and leverage the contextual relationships within entire emails, providing a more comprehensive analysis. However, ensuring transparency about how email body content is analyzed and securing user trust are critical concerns for the adoption of these new features. Providing clear instructions about privacy measures and demonstrating compliance with data protection standards will be essential for advancing security while maintaining user confidence.

Expanding the tool's capabilities in handling various email formats helps to support the advancement of the email header extraction process, making the tool more applicable in diverse email environments. Each email format has a different email structure, and the entities may differ from others. By feeding sufficient data to the model, BERT could learn more about the correlation between the sentences given that the dataset consists of different file structures.

Overall, the project has achieved its primary objectives, providing a robust tool for email header extraction and demonstrating the potential of deep learning techniques in enhancing email forensics and cybersecurity practices.

Acknowledgment

The author would like to thank the Faculty of Computer Science and Information Technology, University of Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *N.K. Chan, I.R. A Hamid;* ***data collection:*** *N.K. Chan, I.R. A Hamid;* ***draft manuscript preparation:*** *N.K. Chan, I.R. A Hamid. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Fedák and J. Štulrajter, "Fundamentals of Static Malware Analysis: Principles, Methods, and Tools," *Science & Military*, vol. 15, no. 1, pp. 45–53, Nov 2020.

[2] N. Buduma, N. Buduma, and J. Papa, "Fundamentals of Deep Learning", 2nd ed. O'Reilly Media, Inc, 2022. Accessed: Jun. 06, 2024. [Online]. Available: https://books.google.com.my/books?hl=en&lr=&id=2e5vEAAAQBAJ&oi=fnd&pg=PT5&dq=deep+learning&ots=3ZHaBIgTCs&sig=x8PscFg0A9lj_00BgXJaidPBVfs&redir_esc=y#v=onepage&q=deep%20learning&f=false

[3] E. Kochmar, "Getting_Started_with_Natural_Language_Pr," *Simon and Schuster*, Aug. 2022, Accessed: Dec. 11, 2023. [Online]. Available: https://books.google.com.my/books?id=I4yKEAAAQBAJ&lpg=PA1&ots=HFw3C3DwfO&dq=nlp%20information%20extraction%20email&lr&pg=PA1#v=onepage&q&f=false

[4] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2019. [Online]. Available: https://github.com/tensorflow/tensor2tensor

[5] A. M. Hiszpanski *et al.*, "Nanomaterial Synthesis Insights from Machine Learning of Scientific Articles by Extracting, Structuring, and Visualizing Knowledge," *J Chem Inf Model*, vol. 60, no. 6, pp. 2876–2887, Jun. 2020, doi: 10.1021/acs.jcim.0c00199.

[6] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." [Online]. Available: https://github.com/tensorflow/tensor2tensor

[7] Google Admin Toolbox: Messageheader." Accessed: Dec. 28, 2023. [Online]. Available: https://toolbox.googleapps.com/apps/messageheader/

[8] "MxToolBox: Email Header Analyzer." Accessed: Dec. 28, 2023. [Online]. Available: https://mxtoolbox.com/EmailHeaders.aspx

[9] Shubhamkhapra, "Email_header_analysis_forensics." Accessed: Dec. 28, 2023. [Online]. Available: https://github.com/Shubhamkhapra/Email_header_analysis_forensic.

[10] C. Will, "The Enron Email Dataset." Accessed: Jun. 08, 2024. [Online]. Available: https://www.kaggle.com/datasets/wcukierski/enron-email-dataset/data

[11] J. Jepkoech, D. M. Mugo, B. K. Kenduiywo, and E. C. Too, "The Effect of Adaptive Learning Rate on the Accuracy of Neural Networks." Accessed: Jun 20, 2024. [Online]. Available: www.ijacsa.thesai.org

[12] R. Qasim, W. H. Bangyal, M. A. Alqarni, and A. Ali Almazroi, "A Fine-Tuned BERT-Based Transfer Learning Approach for Text Classification," *J Healthc Eng*, vol. 2022, 2022, doi: 10.1155/2022/3498123

[13] L. Mcquillan, "BERT: How to Handle Long Documents," Salt Data Labs. Accessed: Jul. 27, 2024. [Online]. Available: https://www.saltdatalabs.com/blog/bert-how-to-handle-long-documents.