TECHNOLOGICAL UNIVERSITY DUBLIN

MASTERS THESIS

Enhancing Static Malware Analysis with Large Language Models and Retrieval-Augmented Generation

Author:
Andre M M FARIA

Supervisor:
Dr Robert G SMITH

A thesis submitted in fulfillment of the requirements
for the degree of M.Sc
in Applied Cybersecurity

in the

School of Informatics and Cyber Security

May 2025

Declaration of Authorship

I, Andre M M FARIA, declare that this thesis titled, "Enhancing Static Malware Analysis with Large Language Models and Retrieval-Augmented Generation" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this Institute of Technology Blanchardstown.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: March 31, 2025

"The true masters are the friends we make along the way."

Anonymous

TECHNOLOGICAL UNIVERSITY DUBLIN

Abstract

School of Informatics and Cyber Security

M.Sc

Enhancing Static Malware Analysis with Large Language Models and Retrieval-Augmented Generation

by Andre M M FARIA

Malware's quick growth presents serious cybersecurity concerns, necessitating ongoing innovation in methods for detection, analysis, and mitigation. When it comes to handling complex and adaptable threats, traditional malware analysis techniques, which mostly rely on manual labour and rule-based systems, are progressively less effective. Large language models (LLMs) are a revolutionary way to automate malware analysis, even though Artificial Intelligence (AI) approaches have been effectively incorporated into cybersecurity. Current AI solutions, including machine learning classifiers and clustering algorithms, concentrate on aspects of malware but frequently lack the overall skills necessary to manage intricate datasets or fully understand virus behaviors.

This study suggests using LLMs to improve and automate static malware analysis. It seeks to automate crucial components of malware reverse engineering, such as code analysis, possible behavior interpretation and anomaly detection, by leveraging LLMs' capacity to handle and synthesize massive, heterogeneous information. In addition to addressing the drawbacks of manual and conventional AI techniques, this strategy adds scalability and adaptability to quickly changing malware threats.

In order to optimize the efficacy of malware analysis in cybersecurity, this study proposes a methodology that combines domain-specific datasets, such as malware sample databases, with prompt engineering techniques to evaluate samples and identify parameters such as malware classification categories and behavioral patterns on decompiled code. This approach is expected to provide a more comprehensive and accurate analysis of malware samples, as well as to improve the overall efficiency of malware analysis in cybersecurity.

Keywords: Malware Analysis, Large Language Models, Static Analysis, Cybersecurity, Artificial Intelligence, Generative Artificial Intelligence, Machine Learning, Reverse Engineering, Anomaly Detection, Code Analysis.

Acknowledgements

Thanks to my parents for supporting me through everything over the years…

Thanks to my supervisor, Dr Robert G Smith, for his guidance and support. :D

Contents

List of Figures

List of Tables

List of Abbreviations

AI	**A**rtificial **I**ntelligence
LLM	**L**arge **L**anguage **M**odel(s)
RAG	**R**etrieval-**A**ugmented **G**eneration

Chapter 1

Introduction

Ever since the creation of the first computer by Charles Babbage in the 19th century, computers have been evolving at a rapid pace performing more and more tasks as the time passes. With this evolution, malicious actors began to notice manners in which to subvert established systems. These individuals, by force of belief or personal gain, have caused losses in the trillions to organizations and individuals across the globe.

As a response to these threats, these organizations and individuals began to develop techniques and tools to counter malicious actors. It has then, started the cat-and-mouse race between criminals and agents of the law. One of these techniques revolves around the analysis of what software adversaries develop to understand how they operate and the vulnerabilities they exploit to weaken or outright topple established security systems.

This thesis presents a research about, and creation of, a method where RAG powered LLMs are introduced onto to the malware static analysis workflow in order to increase efficiency on the analysis.

1.1 Background and Motivation

TODO...

1.1.1 Static Malware Analysis in Cybersecurity

Static malware analysis allows analysts to scrutinize a file's bytecode, strings, imported functions, and file headers to uncover malicious intent. This tried-and-true method of inspecting a program's structure, instructions, and information without actually running it, is still one of the most popular and dependable tools in the cybersecurity toolbox. It is essential in settings where safety, speed, and scalability are critical since it eliminates the possibility of running potentially dangerous code. It is the keystone in early malware identification and categorization, due to its automation capabilities (i.e. antivirus engines or intrusion detection systems).

However, as threat actors evolve their tactics and adopt increasingly sophisticated evasion techniques, static analysis has begun to show its limitations. Many contemporary malware variants leverage obfuscation, packing, encryption, and code polymorphism to mask their functionality and frustrate traditional static inspection. Moreover, extracting actionable intelligence from static features often requires expert-level knowledge in assembly language, file formats, and API behavior—creating a steep barrier for automation. These challenges have shown the need for intelligent, context-aware tools capable of shortening the gap between low-level code patterns and high-level semantic understanding.

1.1.2 Emergence of LLMs in Code and Text Understanding

Over the past few years, Large Language Models (LLMs) have rapidly transitioned from experimental research tools to foundational components across a wide range of natural language processing and code analysis applications. Built on transformer-based architectures and trained on vast corpora comprising not only natural language text but also programming languages and technical documentation, these models have demonstrated an impressive capacity to generate, summarize, translate, and reason about code and human language alike. Their success in handling structured and semi-structured data has opened new avenues in fields such as automated code completion, bug detection, documentation generation, and even code synthesis. With models like GPT, CodeBERT, and StarCoder pushing the limits of scale and capability, LLMs are starting to play a major part in determining how we interact with intricate codebases and interpret syntactic patterns that once required domain-specific knowledge.

This increased capacity in both textual and programmatic reasoning has generated interest in bringing LLMs to cybersecurity, where duties such as code interpretation, vulnerability identification, and malware classification have historically been left for highly trained analysts. Unlike rule-based systems, which rely on predetermined signatures or static heuristics, LLMs may generalise from a variety of examples and adapt to novel or obfuscated code patterns, which is very valuable in malware research. Furthermore, their capacity for natural language explanation enables a level of interpretability that aligns well with security workflows that demand transparency and traceability. While LLMs have shown early promise in tasks such as vulnerability detection and reverse engineering assistance, their full potential in enhancing static malware analysis, especially through integration with contextual retrieval systems, remains a largely untapped area of research, presenting an opportunity to reimagine traditional security paradigms with the help of advanced, language-aware intelligence.

1.1.3 Motivation

The reasoning behind the focus of this research is that static analysis remains to be a critical component of malware detection and neutralization. Security analysts are often required to analyze and process huge amounts of obfuscated data. By integrating RAG powered LLMs to this process we can make the analysis easier by tapping onto contextual knowledge from the RAG and the analysis power of LLMs.

1.1.4 Key Concepts

Here are defined all of the concepts and/or tools that are mentioned.

1.1.4.1 Large Language Models

Large Language Models (LLMs) are a type of multi-parameter Machine Learning (ML) algorithm designed for natural language processing. These models that are trained on a large volume of text using a multitude of learning techniques such as supervised and self-supervised learning.

Generative pre-trained transformers (GPTs) are the biggest and most powerful LLMs. Current models can be adapted to certain tasks (via fine-tuning) or directed by prompt engineering [Brown et al., 2020]. These models learn to predict the syntax,

semantics and ontologies found in human language, but they also pick up biases and errors from the training data [Manning, 2022]. Popular examples of LLMs include:

- GPT models (like GPT-4 or ChatGPT)
- Google's PaLM and Gemini
- Meta Llama series models

1.1.4.2 Prompt Engineering

Interactions with LLMs are made through *prompts*. Those are natural language text segments that describe the task the AI should perform [Radford et al., 2019]. They can be queries, commands or statements including context, instructions and/or relevant information.

Prompt engineering is the action of manipulating these text segments in order to optimize the wanted output [Wahle et al., 2024]. For example, providing a prompt such as "What is an apple?" we have this output:

> An apple is a round, edible fruit produced by the apple tree (Malus domestica). It typically has smooth skin that can be red, green, or yellow, and a sweet to tart taste depending on the variety. Apples have a firm white flesh and contain a core with small black or brown seeds.
> They're commonly eaten raw, but also used in cooking and baking-like in apple pies, sauces, and juices. Apples are rich in fiber, vitamin C, and antioxidants, making them a popular choice for a healthy snack. The saying "an apple a day keeps the doctor away" hints at their reputation for promoting good health.
> Want to know about their origin, symbolism, or maybe how they're grown?

This output is acceptable, however it can be modified with some prompt engineering, such as an ask to change the output formatting. Then, providing the phrase "Provide only bullet points." in addition ot the original we have:

> * Edible fruit from the apple tree (Malus domestica)
> * Typically round with red, green, or yellow skin
> * White, crisp flesh with a sweet to tart flavor
> * Contains a core with small seeds
> * Commonly eaten raw or used in cooking and baking
> * Rich in fiber, vitamin C, and antioxidants
> * Grown in temperate climates worldwide
> * Symbol of health, knowledge, and temptation in various cultures

1.1.4.3 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) [Lewis et al., 2021] is a technique that allows for LLMs to query information from a specific type of datastore and use it as context for the response it provides. This allows for adaptive learning of the models without the need of retraining (fine-tuning). It improves the quality of LLM output as the model does not rely solely on pre-trained knowledge but also on relevant information from the datastore.

RAG works by having the LLM to query the datastore using vector search, embeddings or keyword matching (depending on the datastore type), feeding this

data onto the LLM input and then continuing with the normal LLM process for output generation.

1.1.4.4 Malware

Short for *Malicious Software*, Malware is any program or code that is created for malicious purposes like exploitation of computer systems networks or users. Often created with monetary gain in mind, these programs activities include stealing sensitive data, damage systems, disrupt operations or gain unauthorized access to systems or environments. Common types of malware include:

- Viruses: Attach themselves to legitimate software, spreading when the software is run.
- Worms: Autonomous malware capable of multiplying across networks without needing human intervention. Exploit network weaknesses to infiltrate other systems without permission.
- Trojans: Also known as Trojan Horses, these are disguised as genuine software to trick users into running them.
- Ransomware: Encrypts or locks files, demanding payment (ransom) for restoration.
- Spyware: Stealthily gathers sensitive data and user activities without the user's consent.
- Adware: Delivers unwanted advertisements, potentially slowing down or compromising systems.

1.1.4.5 Malware Analysis

As mentioned before, the software created by malicious actors needs to be evaluated and analyzed to understand how it works, what exploits it takes advantage of and how these exploits can be patched. As a basis, there are two types of analysis of software that can be made to understand any kind of software. These are as follows:

- **Static Analysis**: Focuses on looking at the sequences of individual instructions on the software bytecode to identify the its characteristics, such as identification of the parameters in which the analyzed software is built upon (e.g. Operating system it executes on, processor architecture, etc), what execution paths it employs, what system calls it performs, etc. This analysis is done without running the code and it requires a lot of time and effort from the analyst.
- **Dynamic Analysis**: Entails running the potential malicious software to gather runtime information such as network calls and system call execution data which are not visible through static analysis. This analysis is done by creating a special (sandbox) environment (i.e docker container and/or a virtual machine) and running the software.

1.1.4.6 Static Analysis vs Dynamic Analysis

Both static and dynamic analysis have their own benefits and are important on their own right for understanding how malware is designed and developed. While static analysis takes advantage of its speed and safety where files do not need to be executed to be analyzed, dynamic analysis enables a more extensive analysis which might not be visible with static analysis alone.

However, both have their limitations such as the increased computational cost and requirement of a secure environment of dynamic analysis versus the vulnerability of static analysis to obfuscation and encryption techniques.

1.2 Problem Statement

TODO...

1.3 Research Aim and Objectives

TODO...

1.3.1 Aim

Research the enhancement of static malware analysis using RAG-enhanced LLMs

1.3.2 Objectives

- Develop a method for integrating LLMs with RAG to enhance static malware analysis.
- Evaluate the effectiveness of RAG-enhanced LLMs in identifying, explaining, and comparing malware threats using static features (e.g., file structure, bytecode, API calls) against traditional static malware analysis techniques in terms of accuracy, efficiency, and interpretability, while identifying challenges and potential risks (e.g., adversarial manipulation and hallucination).

1.4 Research Questions and Hypothesis

1.4.1 Objectives

These are the research questions that will guide the development of this thesis.

- How LLMs can be leveraged to enhance static malware analysis?
- How does a RAG-enhanced LLM compare to traditional static analysis techniques in terms of accuracy, efficiency, and interpretability?
- What role does RAG play in improving LLM-driven malware classification, particularly in terms of contextual relevance and justifiability of results?

1.4.2 Hypothesis

- Hypothesis 1: LLMs can enhance static malware analysis by accurately interpreting and classifying malware samples based on static features such as bytecode, file structure, and API calls, offering greater adaptability and insight than rule-based static tools.
- Hypothesis 2: A RAG-enhanced LLM will outperform traditional static analysis techniques in terms of interpretability and contextual understanding, while maintaining comparable levels of accuracy and efficiency in malware classification.
- Hypothesis 3: RAG improves LLM-driven malware classification by providing relevant contextual information during inference, leading to more justifiable and semantically grounded explanations of classification results.

1.5 Methodology Overview

TODO...

1.6 Contributions of the Research

TODO...

1.7 Thesis Structure

TODO...

Chapter 2

Literature Review

TODO...

2.1 Static Malware Analysis

TODO...

2.1.1 Signature-Based and Heuristic Methods

TODO...

2.1.2 Machine Learning-Based Approaches

TODO...

2.2 The Role of Language Models in Cybersecurity

TODO...

2.3 Retrieval-Augmented Generation (RAG) in NLP

TODO...

2.4 Combining RAG and LLMs for Static Malware Analysis

TODO...

2.5 Gaps in Current Literature

TODO...

2.6 Summary

TODO...

Chapter 3

Methodology

TODO...

3.1 Research Design

TODO...

3.2 System Architecture

This section outlines the overview of the pipeline architecture. These steps have been designed to modularize the analysis, ensure reproducibility and allow for algorithmic changes whenever needed. Below, the diagram of the pipeline elements is displayed along with an overview on each of them.

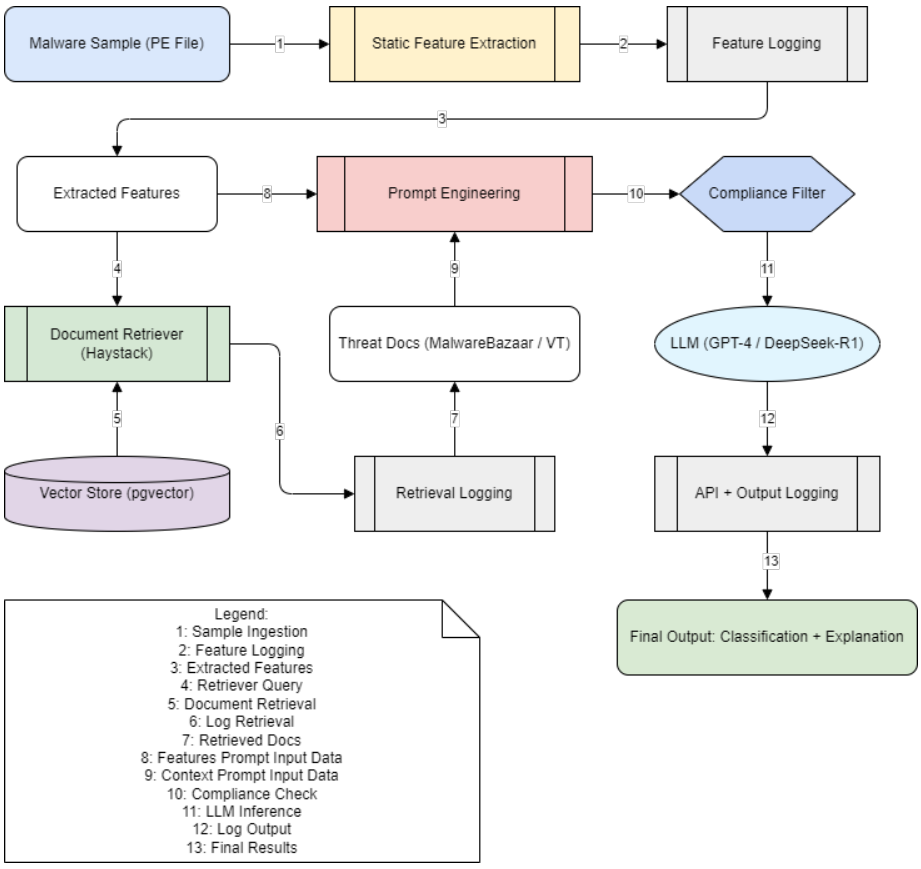

FIGURE 3.1: System Pipeline Diagram

1. **Malware Sample Ingestion:** Raw malware binaries (e.g., PE files) are sourced from public repositories such as MalwareBazaar or accessed via API from VirusTotal. Each file is stored with a unique identifier (e.g., SHA256 hash).
2. **Static Feature Extraction:** Using tools such as `pefile`, the system extracts static attributes from the malware, including:
    - PE header metadata
    - Imported API calls
    - Section structure and entropy
    - Printable strings
3. **Log Feature Extraction:** All extracted features are logged with timestamps, sample identifiers, and summary statistics.
4. **Feature-Based Retrieval Query:** The extracted static features are embedded into vector representations (if needed) and used as queries to a semantic search index.
5. **Document Retrieval:** Relevant documents (e.g., threat reports, malware family descriptions) are retrieved from an external datastore to be provided as context.
6. **Log Retrieval Step:** Retrieval activity is logged, including query terms, matching document IDs, retrieval time, and the data source used (e.g., VirusTotal vs. local data).
7. **Prompt Construction:** Retrieved documents and extracted features are formatted into structured prompts. Templates are used to ensure consistent formatting across LLM queries.
8. **Compliance Check Module:** Before sending prompts to the LLM, a filtering mechanism ensures that no disallowed content (e.g., malware payloads, sensitive data, proprietary code) is included.
9. **LLM Inference (GPT-4 / DeepSeek-R1):** The prompt is passed to a large language model (either GPT-4 or DeepSeek-R1) to generate:
    - Malware family classification
    - Explanation or justification of the classification
    - Comparative analysis against known malware types
10. **Log API Call:** Each API interaction is logged, including:
    - Model version and endpoint
    - Prompt metadata (not full text for privacy)
    - Token usage, latency, and return timestamp
11. **Output Generation:** The model output (classification + explanation) is presented to the user or stored for evaluation.
12. **Results Logging and Storage:** Final outputs, along with model metadata and hash references, are stored for downstream analysis or human review.

3.3 Data Sources

TODO...

3.4 Retrieval-Augmented Generation (RAG) Pipeline

TODO...

3.5 Prompt Engineering

TODO...

3.6 Toolchain

This section outlines the used tooling for the experiments and some discussion on why they were chosen.

3.6.1 Programming Language and Environment

TODO...

3.6.2 Language Models

TODO...

3.6.3 RAG Framework and Storage

TODO...

3.6.4 Malware Intelligence Sources

TODO...

3.6.5 Evaluation and Visualization

TODO...

3.7 Evaluation Framework

TODO...

3.8 Reliability, Validity, and Limitations

TODO...

3.9 Ethical Considerations

TODO...

3.10 Summary

TODO...

Chapter 4

Discussion of Results

TODO...

4.1 Summary of Key Findings

TODO...

4.2 Interpretation of Findings

TODO...

4.3 Implications of the Study

TODO...

4.4 Limitations

TODO...

4.5 Recommendations for Future Research

TODO...

4.6 Conclusion

TODO...

Chapter 5

Conclusion

TODO...

5.1 Summary of Research

TODO...

5.2 Answers to Research Questions

TODO...

5.3 Research Contributions

TODO...

5.4 Implications of the Study

TODO...

5.5 Limitations

TODO...

5.6 Recommendations for Future Research

TODO...

5.7 Final Reflections

TODO...

Appendix A

Frequently Asked Questions

A.1 Getting Feedback

1. Get feedback **often** and from different audiences – your family, friends, professors, colleagues, advisor, other graduate students. The more you talk about your research, the more comfortable you get with it.

2. Keep a positive attitude. Research is hard. If it were easy, everyone would be doing it.

3. Consider setting up or joining a thesis group to share your ideas and experiences.

Supervisor's feedback

Some supervisors will ask for you to send each chapter as you complete it, offering feedback at that point, and then again at the end when the thesis chapters are collated. Other supervisors may want to be more involved, and there are others who will not want to see your thesis until it is completed by your standards. Whichever approach your supervisor takes, be aware that they will need some time to read through your work and provide feedback. Your thesis review is likely not the only piece of work your supervisor is undertaking, so be patient and factor review time into your work schedule.

A couple of points to note about supervisor feedback:

- You will receive feedback on your approach to research (i.e., method, experiment design etc...) as well as your writing. It is your responsibility to take notes at meetings etc. in order to record this feedback. It is also up to you whether or not you act upon the feedback provided.

- Your supervisor's role is to guide your work. It is not their job to complete the research, suggest methods, design experiments, or to write/rewrite sections of your thesis.

- Your supervisor should be supportive but sometimes their feedback may be difficult to hear. Just remember, their goal is to guide you and to make you a better researcher. Learn to have your work criticised in a constructive manner, it is part of the learning process.

- It is not the role of your supervisor to proofread your thesis. Many supervisors will point out typos, grammatical errors or styling issues etc. when they see them, but this is not their role.

A.2 Proofreading/copyediting

It is important to have your work proofread[1]. If English is not your first language, this is even more important for you.

How?

A good approach is to proofread yourself as you write and then again when you are finished writing a section or chapter. When you have a near final draft, have it proofread by a friend, family member, colleague, or a classmate etc... (not your supervisor). Choose your proofreader wisely. Make sure that they have good written English skills and are able to spot grammatical errors. A native English speaker can be good for this but not all native English speakers have the skills needed to be a good proofreader.

There are many things to look out for when reviewing your own work, everything from text alignment and section numbering, to figures and tables, to spelling and grammar. It's best to identify and fix any of these errors immediately. Don't wait until the end because these will build up and it often takes longer than you think to fix them.

If you find that you make the same mistake regularly, e.g., you misspell the same word regularly, or you use a colon where you shouldn't, then make a list of these to check back when you are finished each section (the search feature is good for this).

[1] Two types of editing that are commonly used interchangeably are copy editing and proofreading. Both types of editing clean up writing, but each has its distinct contribution to the process. https://thesiswhisperer.com/2016/11/30/doing-a-copy-edit-of-your-thesis/

A.3 Writing Assistants

In the past, students may have used tools such as Grammarly or Quetext, but this has become more problematic because such editing tools now come with AI assisted writing (see more here: https://tudublin.libguides.com/c.php?g=720901&p=5233062).

Using tools such as a spell checker, a grammar checker, and a punctuation checker are generally acceptable. Using more advanced tools to rewrite sentences, check tone, offer alternative word choices, offer citations etc... is not acceptable.

If in doubt don't use any such software. In general, it appears that, as of 2025, the free version of Grammarly is fine to use, but the pro version is not.

A.4 Example of Longtable

Ticket Type ID	Description
300	Feeder Ticket - Child
301	Feeder Ticket - Adult
310	10-Journey Feeder - Adult
317	Airlink Adult Airport-Busarus
318	Airlink Child Airport-Busarus
319	Airlink Child Airport-Heuston
320	Airlink Adult Airport-Heuston
333	Adult Single Feeder
365	Child Bus/Rail Short Hop - Day
366	Adult Bus/Rail Short Hop - Day
367	Family Bus/Rail Short Hop - Day
369	4 Day Explorer
410	Weekly Adult Short Hop Bus/Rail
430	Weekly Adult Medium Hop Bus/Rail
431	Weekly Adult Long Hop Bus/Rail
432	Weekly Adult Giant Hop Bus/Rail
433	Monthly Adult Short Hop Bus/Rail
455	Monthly Adult Long Hop Bus/Rail
456	Monthly Adult Giant Hop Bus/Rail
457	Monthly Student Short Hop Bus/Rail
458	Annual Bus/Rail
478	Annual All CIE Services
479	Annual CIE Pensioner Bus/Rail
480	Monthly CIE Pensioner Bus/Rail
493	Foreign Student - 1 Week
494	Foreign Student - 2 Week
495	Foreign Student - 3 Week
496	Foreign Student - 4 Week
497	CYC Group
600	Adult Cash Fare
608	Nitelink (Maynouth/Celbridge)
609	Nitelink (Maynouth/Celbridge)
610	Child Cash Fare
620	Schoolchild Cash Fare
625	Adult (formerly Shopper)
630	Adult 10-Journey (3 Stages)
631	Adult 10-Journey (7 Stages)
632	Adult 10-Journey (12 Stages)
633	Adult 10-Journey (23 Stages)
634	Adult 10-Journey (23+ Stages)
640	Adult 2-Journey (3 Stages)
641	Adult 2-Journey (7 Stages)
642	Adult 2-Journey (12 Stages)
643	Adult 2-Journey (23 Stages)
644	Adult 2-Journey (23+ Stages)
650	Schoolchild 10-Journey
651	Scholar 10-Journey
652	Schoolchild 2-Journey
653	Scholar 2-Journey
657	Transfer 90 (or Passenger Change)
658	Adult Single Heuston-CC
660	Adult One Day Travelwide
661	Child One Day Travelwide
662	Family One Day Travelwide
665	Rambler (3 Day Bus only)
670	Weekly Adult Bus

Ticket Type ID	Description
671	Weekly Adult Cityzone
690	Weekly Student Travelwide
691	Weekly Student Cityzone
705	Monthly Adult Citizone (AerLingus.)
710	Monthly Adult Travelwide
730	Annual Adult Travelwide
760	Annual Staff Bus
790	School Pass
791	OAP Pass
800	City Tour - Adult
801	City Tour - Family
802	City Tour - Child
898	10 - Journey Test Ticket

Bibliography

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

Lewis, Patrick et al. (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

Manning, Christopher D. (May 2022). "Human Language Understanding & Reasoning". In: *Daedalus* 151.2, pp. 127–138. ISSN: 0011-5266. DOI: 10.1162/daed_a_01905. eprint: https://direct.mit.edu/daed/article-pdf/151/2/127/2060607/daed_a_01905.pdf. URL: https://doi.org/10.1162/daed_a_01905.

Radford, Alec et al. (2019). "Language Models are Unsupervised Multitask Learners". In: *OpenAI*. Accessed: 2024-11-15. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Wahle, Jan Philip et al. (Nov. 2024). "Paraphrase Types Elicit Prompt Engineering Capabilities". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, pp. 11004–11033. DOI: 10.18653/v1/2024.emnlp-main.617. URL: https://aclanthology.org/2024.emnlp-main.617/.