Master's thesis

Master's Programme in Computer Science

Neural Router: A System for Structure-Agnostic and Configuration-Free Content Delivery

Alexander Engelhardt

April 16, 2025

Faculty of Science

University of Helsinki

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki, Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Alexander Engelhardt

Työn nimi — Arbetets titel — Title

Neural Router:
A System for Structure-Agnostic and Configuration-Free Content Delivery

Ohjaajat — Handledare — Supervisors

Sasu Tarkoma, Abhishek Kumar

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	April 16, 2025	67 pages

Tiivistelmä — Referat — Abstract

As data volumes and complexity continue to grow, the development of effective data management solutions that work on a scale becomes increasingly challenging. Recent advances in artificial intelligence provide new tools to address this challenge by enabling software capable of handling diverse data in an intelligent and intent-driven manner.

This thesis presents such a software solution for the publish-subscribe paradigm, the *de facto* communication model for asynchronous and decoupled communication. The two main contributions are: (1) the **Agentic Pub-Sub Architecture**, which establishes a system capable of processing *highly diverse data with immediate effect while continuously self-improving over time*; and (2) the implementation of **Neural Router**, which provides the diverse data processing and immediate functionality within the Agentic Pub-Sub Architecture by handling data in a structure-agnostic and configuration-free manner.

To validate the effectiveness of the Neural Router, it was tested on two real-world datasets that represent opposites in the structuredness spectrum. The first, *CardiffNlp*, consists of social media posts and evaluates the Neural Router's ability to handle ambiguous natural language. The second dataset, *SmartCampus*, consists of sensor readings and evaluates the performance of the Neural Router on structured numerical data. With identical configurations, the Neural Router achieves an F1-score of 0.619 on CardiffNlp and 0.900 on SmartCampus. However, manual inspection of the CardiffNlp results suggests a significantly higher practical accuracy.

By introducing the Neural Router and establishing the foundations of the Agentic Pub-Sub Architecture, this thesis demonstrates the potential for an intelligent publish-subscribe system capable of addressing challenges that traditional, application-specific solutions cannot resolve.

ACM Computing Classification System (CCS) Computer systems organization → Distributed architectures → Publish-subscribe Information systems → Information retrieval → Content-based filtering Computing methodologies → Natural language processing → Applications

Avainsanat — Nyckelord — Keywords

AI-agent, Publish-Subscribe, Self-improving, LLM

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Software study track

Acknowledgements

The thesis work was carried out in the Business Finland Neural Pub/Sub research project. The Neural Router presented in this thesis is based on the early design of the Neural Router by Professor Sasu Tarkoma - my supervisor. I would like to thank him for the opportunity to work on this and for trusting me to incorporate my own ideas. I also want to thank my second supervisor Dr. Abhishek Kumar for being there on a weekly basis, guiding me through the practices of the academic world and for validating the feasibility of many of my proposed solutions. I greatly appreciate your support throughout the process.

Wells, 1883 ©

Artificial intelligence may soon understand us better than we do ourselves, yet for now, it was hopeless at interpreting this symbolic head of cognition.

A sentiment of the times in which this thesis was written

Contents

List of Abbreviations

Abbreviation	**Full Form**
Pub-Sub	Publish-Subscribe
APSA	Agentic Pub-Sub Architecture
LLM	Large Language Model
API	Application Programming Interface
IoT	Internet of Things

1 Introduction

1.1 Context and Motivation

The world is experiencing an unprecedented surge in data generation, surpassing current capabilities for efficient processing and management (Ataei and Litchfield, 2022). This information explosion is evident not only in the sheer volume of data but also in its increasing complexity. The advent of Generative Artificial Intelligence (GenAI) has further amplified this challenge, leading to an influx of largely unstructured content that traditional data management systems struggle to handle (Wang et al., 2023). As a result, there is increasing uncaptured value, network congestion, and lost opportunities.

Although GenAI is part of the problem, it might also be part of the solution. Especially the perceived *intelligence* of gpt-based Large Language Models has sparked hope of *autonomous intelligent* systems capable of interpreting large amounts of complex and unstructured data (Yan et al., 2024).

The publish-subscribe (Pub-Sub) pattern is a fundamental communication paradigm for asynchronous and scalable data routing (Eugster et al., 2003). However, traditional Pub-Sub architectures rely on rule-based matching and strict protocols, making them increasingly inadequate in the face of data heterogeneity and volume expansion (Saleh et al., 2024a). To address these limitations, a next-generation Pub-Sub system infused with native intelligence and autonomous decision-making is required.

Integrating intelligence into the Pub-Sub paradigm is a multifaceted challenge. First, there exist multiple possible approaches. More importantly, different fundamental problems must be addressed. This thesis focusses on one of the core challenges in the Pub-Sub paradigm: effective data routing. Given the evolving data landscape, this problem can be reframed as follows.

> *How can data be routed intelligently to address the challenges posed by the information explosion?*

1.2 Problem Statement

The challenges posed by increasing the complexity and volume of data for Pub-Sub systems can be viewed as a challenge of *flexibility*. In this thesis, *flexibility* is defined as *the ability of a Pub-Sub system to dynamically handle diverse data formats without human intervention*. This definition is based on the core design of decoupling in Pub-Sub systems. While decoupling has traditionally been understood as *decoupling in space and time* (Eugster et al., 2003), this thesis argues that *true decoupling* must extend beyond these dimensions to include *decoupling from strict protocol adherence*. That is, this is a problem of *structure-agnostic and configuration-free data routing*.

Most existing Pub-Sub systems rely on *rule-based data routing*, where publishers and subscribers must conform to predefined schemas, formats, or taxonomies. This approach leads to *application-specific solutions* that struggle to accommodate *heterogeneous, unstructured, and evolving data* (Saleh et al., 2024b). To address this, an intelligent Pub-Sub system should transition toward *intent-based data routing*, allowing publishers and subscribers to interact without strict protocol adherence. The problem of protocol interoperability is particularly evident in large-scale systems such as *smart cities* (Weil et al., 2023), *meta-verse* (Lei et al., 2023), and *6G networks* (Tarkoma et al., 2023).

Attempts to introduce greater flexibility into Pub-Sub systems have relied primarily on *content-based routing*, where the *data content itself* dictates routing decisions (Eugster et al., 2003). Although this method improves flexibility, it still depends on predefined filtering rules and domain-specific heuristics, making it unsuitable for handling highly dynamic, unstructured, or unanticipated data (Saleh et al., 2024b).

To overcome these limitations, the utilisation of Large Language Models (LLMs) and semantic reasoning in Pub-Sub systems presents a promising direction (Saleh et al., 2024a). However, exactly *how to integrate* these capabilities has not been thoroughly explored, and no existing research has presented well-defined architectures or solutions focused on the delivery of *structure-agnostic and configuration-free content*.

1.3 Research Questions and Objectives

The objective of this thesis is to gather recent research on intelligent Pub-Sub systems and present the foundations to an architecture focused on *structure-agnostic and configuration-free content-delivery*. In addition, the core matching logic of this architecture is to be implemented and evaluated. To capture these objectives, a set of three research questions is defined.

- **RQ1**: *What features and requirements are presented in recent literature on natively intelligent Pub-Sub systems?*
- **RQ2**: *How can the foundational structure of a natively intelligent Pub-Sub system be designed to achieve structure-agnostic and configuration-free content-delivery while self-improving over time?*
- **RQ3**: *How can a data routing logic be implemented and evaluated to demonstrate the feasibility of structure-agnostic and configuration-free content delivery?*

Research Question 1 aims to establish a solid foundation of knowledge and understanding of recent research on intelligent Pub-Sub systems. Although the focus of this thesis is primarily on flexible data routing, this research question aims to build a good understanding of the big picture to ensure that any proposed solutions do not overprioritize flexibility to the detriment of other critical requirements. It also ensures that the proposed solutions learn from the challenges faced by others.

Research Question 2 investigates the architectural design required to support natively intelligent Pub-Sub systems. It focusses on defining the fundamental structures needed to support *structure-agnostic and configuration-free content delivery* and the ability to *auto-optimise* performance. While this is the core focus, the broader context of a Pub-Sub system is also taken into consideration.

Research Question 3 focusses on the design, implementation and evaluation of the *core matching logic* within a broader intelligent Pub-Sub architecture. It examines how *Large Language Models (LLMs)* and *semantic reasoning* can be used to enable *structure-agnostic* and *configuration-free* content delivery. Additionally, this research question assesses the *accuracy, efficiency, and feasibility* of the core matching logic.

1.4 Contributions

The first major contribution of this thesis is the **Agentic Pub-Sub Architecture** (APSA). This includes identifying critical quality requirements and defining the most basic architectural structure. This structure consists of three main components: **1.)** *The Overseer*, expressing the intelligence of the system, **2.)** *The Neural Router* enabling immediate and structure-agnostic and configuration-free routing, and **3.)** *Expert Routers*, constituting routing optimisation. By defining the foundations of this architecture, this thesis aims to lay the groundwork to solve the long-term challenges of *true decoupling* and *intent-based routing*.

The second major contribution of this thesis is the design, implementation and evaluation of the **Neural Router**. Designed to enable structure-agnostic and configuration-free data routing, its role within APSA is to function as a universal router capable of handling any data with sufficient accuracy and efficiency until an optimised solution is found. By implementing and evaluating its accuracy and efficiency, the Neural Router proves the feasibility of *structure-agnostic* and *configuration-free* routing.

The third major contribution of this thesis are the **two datasets** used to evaluate the Neural Router. More precisely, what has been created are a set of subscriptions, including ground truth, for two existing datasets. The first dataset is *CardifNlp*, consisting of real-world social media posts with subscriptions that express interest according to a set of themes. The second dataset *SmartCampus*, consists of readings from real world IoT sensors, with subscriptions consisting of SQL data queries. Natural language representations of SQL queries are also available. Both the CardiffNlp and SmartCampus datasets are publicly available on GitHub:

- CardiffNlp Repository: https://github.com/alwengel/CardiffNlp-Pub-Sub
- SmartCampus Repository: https://github.com/alwengel/SmartCampus-Pub-Sub

1.5 Organization of the Thesis

The remainder of this thesis is structured as follows: **Chapter 2** provides an overview of the publishing subscription paradigm, covering both fundamental concepts and recent research on infusing intelligence into Pub-Sub systems. **Chapter 3** introduces the foundation of the Agentic Pub-Sub Architecture. **Chapter 4** delves into the Neural Router, explaining its design and functionality. **Chapter 5** describes the experimental methodology and the datasets used to evaluate the Neural Router. **Chapter 6** presents the experimental results, followed by **Chapter 7**, which analyses the results, provides insight and explores opportunities for further improvements. Finally, **Chapter 8** summarises the key contributions of this work and outlines potential directions for future research.

2 Background

2.1 Overview of Publish-Subscribe Systems

2.1.1 Fundamentals of Pub-Sub

"The many Faces of Publish/Subscribe" (Eugster et al., 2003) gives an in-depth analysis of the fundamentals of the Pub-Sub communication paradigm. The basic idea of Pub-Sub is to decouple *publishers*, the producers of data, *subscribers*, the consumers of data. This is done by replacing any direct communication between publishers and subscribers, with a *router* (broker/bus), responsible of matching the interests of subscribers referred to as *subscriptions*, with data produced by publishers referred to as *publications*. For a visual representation of the basic structure of the Pub-Sub paradigm, see Figure 2.1.

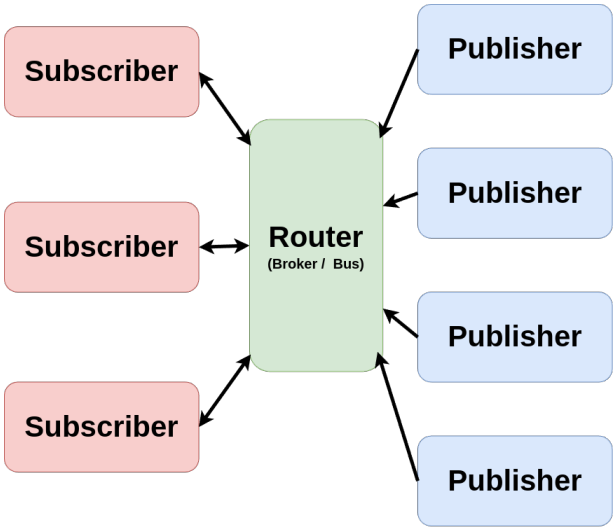


Figure 2.1: The basic structure of the Pub-Sub communication paradigm.

The Pub-Sub paradigm allows for decoupling in dimensions of *time* and *space*. Decoupling by time, or *asynchronous* communication, in the context of Pub-Sub, means that publishers and subscribers are not required to be online simultaneously. For example, the subscriber registers interest with the router and then disconnects. Later, a publisher joins, sends data to the router, and leaves. When the subscriber reconnects, the router provides the relevant data based on the prior subscription.

The decoupling of publishers and subscribers in the dimension of *space*, means that neither entity needs knowledge of the other's location or identity. That is, both subscribers and publishers are unaware of each other and communicate strictly through the router. It is the sole responsibility of the router to keep track of the locations and identities of subscribers and publishers.

The decoupling of publishers and subscribers allows the Pub-Sub systems to disseminate information both efficiently and scalably (Donta et al., 2022), (Chafi et al., 2020), (Maniezzo et al., 2022). Efficiency is largely determined by the specific routing approaches (see Subsection 2.1.2). Scalability, defined as a system's ability to handle increasing computational loads (Red Hat, 2024), is achieved through horizontal or vertical scaling of the router (Chapuis et al., 2017). Vertical scaling simply implies adding more compute to the router in the form of stronger hardware, while horizontal scaling involves adding more compute by replicating the router through a distributed system. Although vertical scaling is simpler and requires less overhead, there is a cap on how far the system can be scaled this way. Horizontal scaling, on the other hand, allows for potentially infinite scaling, but requires a more complex system of load balancing between multiple instances of the Pub-Sub router.

In addition to efficiency and scalability, the router can also provide *reliability* through ensuring data is not lost (Oracle, 2024a). This is usually done through acknowledgment, retries, or temporary storage until delivery is successful. *Fault tolerance* against system failures or router reboots can be achieved through persistent storage, which means that data are kept in some kind of database or file system (Oracle, 2024b). The aspect related to persistent storage is the aspect of the router *in the internal state*. Although this can have many features, one that is key is the routers ability to reflect the history of processed data (Lazidis et al., 2022). More precisely, functions such as aggregation, min, max, and mean values are a way to save the context into the state of the router. Although relatively simple if only implemented on a single instance of a router, this becomes increasingly complex, demanding *distributed snapshots*, as replications are spread across non-local nodes. Additionally, routers can provide authentication support, by enforcing access rights, encryption, etc. (Oracle, 2024b).

2.1.2 Routing Approaches

Topic-Based Routing: The most traditional Pub-Sub routing approach is based on the so-called *topics* (Eugster et al., 2003). Simply put, it is a way to divide publications and subscriptions into a set of categories. Subscribers choose which topics to *subscribe* to, while *publishers* define which topics to publish data. For a simple visualized example of a topic-based routing system, see Figure 2.2. Here, we see how a set of three subscribers have each chosen to subscribe to one topic, while a set of four publishers publish data according to topics.

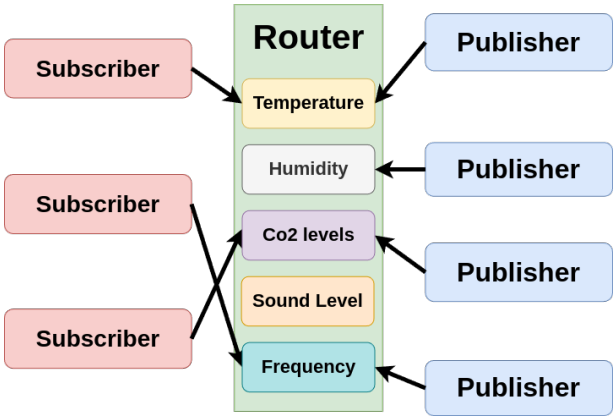


Figure 2.2: A basic example of topic based routing.

However, the downsides emerge in maintaining such systems (Tarkoma, 2012). To optimize topic-based routing, it is possible to structure topics *hierarchically* (Eugster et al., 2003). This allows for the usage of *containment relationships* where a subscription or publication belonging to a parent topic implies that it also belongs to all child topics of that parent. For example, (Milo et al., 2007) presents a distributed clustering algorithm that uses correlations in user subscriptions to create a set of *virtual topics* that are used to group subscriptions together.

Content-Based Routing: While topic-based Pub-Sub systems route data based on specific topics, content-based Pub-Sub routes data based on the contents of subscriptions and publications themselves (Eugster et al., 2003). This allows for a system where subscribers can specify the desired information on a much more granular scale than with topic-based routing. For example, Figure 2.3 illustrates subscriptions defining specific conditions for a category, while the router is responsible for enforcing these conditions as publications enter the system. While this is a simple example, queries can be more complex.

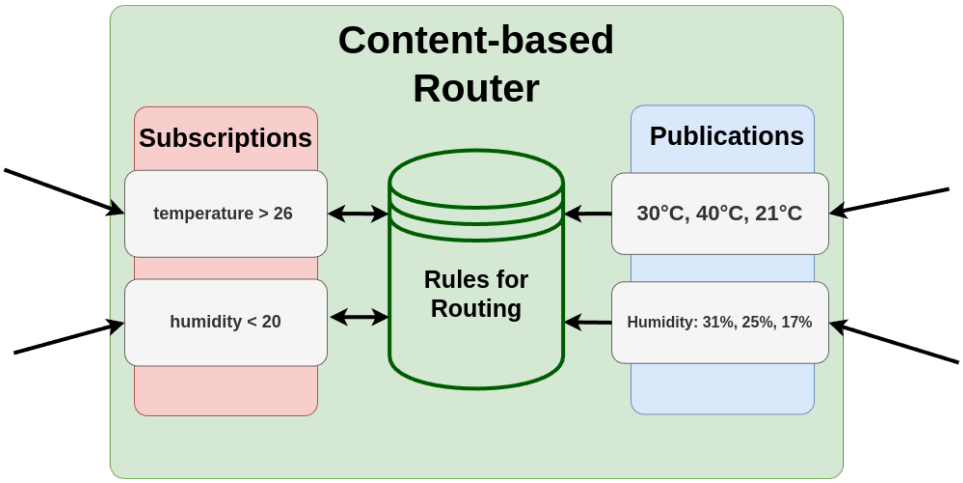


Figure 2.3: A basic example of content-based routing.

With the content-based routing approach, it is possible to define *complex* and *composite* subscription queries to extract specific information or patterns from the publication data, which is impossible with strict topic-based routing (Eugster et al., 2003). The design of a content-based pub-sub system that allows expressive matching semantics has long been a challenge, and there is a direct relationship between expressivity and computational cost, leading to a trade-off with latency and scalability (Tarkoma, 2012).

2.2 Large Language Models and Agents

2.2.1 Large Language Models

Large Language Models (LLMs) are sophisticated deep learning neural networks that possess the ability to interpret and generate natural language, code, and other textual data. Recent examples, such as the gpt series (OpenAI et al., 2024) are built on the *transformer* architecture (Vaswani et al., 2017) which allows the capture of long-range dependencies in sequential data. Such LLMs consist of several neural network layers, each consisting of a number of parameters. The total number of parameters decides the size of *the model*. The largest LLMs today have hundreds of billions, if not more than a trillion parameters (Li, 2025). With the launch of ChatGpt in 2022, LLMs have been thrown to the forefront of artificial intelligence due to their ability to *understand complex phenomena*, produce *human-level* language and code, and due to their ability to *reason* to some extent. Next, the key steps in creating LLMs are described.

First, a large amount of raw text-data consisting of natural language, code, csv, JSON etc., is collected. Then these data are *tokenised*, which constitutes splitting text into individual *tokens* that constitutes typically sub-word units (Vaswani et al., 2017).

Using tokenized data, the LLM is trained on it in an unsupervised learning process called *pre-training* (H. Li, 2022). Here, the model learns the general patterns found in its training data by guessing the next token or filling in gaps. Following pre-training, the LLM is *fine-tuned* in a supervised learning process to achieve the desired behaviour.

Once trained and fine-tuned, an LLM is interacted with by giving it input, called *prompts*. The specific formulations used in the prompts can have a great effect on the output. For example, *Chain-of-Thought* prompts the LLM to "think step-by-step" before producing its answer (Wei et al., 2022).

Chain-of-Thought has been paramount in creating so-called *reasoning models* such as O1 (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025). These models have been fine-tuned to spend tokens in a Chain-of-Thought fashion to achieve higher levels of *reasoning* and increase performance.

The two most important factors for the performance of an LLM are the *amount of training data* and the *number of parameters* (Kaplan et al., 2020). While revolutionising, LLMs exhibit the following challenging characteristics (Karpathy, 2023):

- Their output is **non-deterministic**.
- They may produce illogical or false statements, often referred to as **hallucinations**.
- While reasoning capabilities exist, they are **not strictly logical**.

An example of this can be seen in responses generated by ChatGPT. When asked who Tom Cruise's mother is, it knows the correct answer: Mary Lee Pfeiffer. However, when asked about the son of Mary Lee Pfeiffer, it does not know or hallucinates something (Karpathy, 2023).

2.2.2 Agents

The concept of an *agent* has long been a central part in the creation of intelligent systems and can be defined as an *autonomous system located in an environment and capable of acting within this environment* (Franklin and Graesser, 1999). With recent advances in LLMs, the discussion around *LLM-based Agents* has gathered a lot of interest (Wang et al., 2024). With the ability to reason and handle various forms of data along with the capability to generate code, LLMs can function as the central brain of an agent unlocking new levels of autonomous and intelligent interaction with their environment.

With a multitude of approaches for building LLM-based agents, the fundamentals are based on keeping the LLM up to date with data from the environment and informing it about what action it can decide to take (Wang et al., 2023). On a technical level, these actions would consist of function-calling to interact with its environment. Central challenges consists of 1.) making sure the agent reflects its role and stays on-track with its mission, 2.) safety concerns around agents being misused for malicious intent, and 3.) hallucinations. Especially, with multi-agent frameworks, where many agents work together to achieve a common goal, these challenges are amplified further.

With a plethora of use cases and multiple technical approaches to achieving effective LLM-based agents, it is one of the most central paradigms shaping the world today. Integrating LLMs and LLM-based agents into the Pub-Sub communication model poses one of the key ways of infusing intelligence into the Pub-Sub paradigm and stands as a possible solution to tackling the crucial challenges of ever-increasing data volumes and data complexity.

2.3 Infusing Intelligence into Pub-Sub

2.3.1 Overview of Intelligence in Pub-Sub

Saleh et al., 2024b provide a systematic review of the recent literature and future prospects related to Pub-Sub for Edge Intelligence. Here, research gaps and potential enhancements to existing solutions are presented. There, Chiu, 2023, highlights how Generative AI can improve the matching accuracy on heterogeneous data in Pub-Sub systems by leveraging *text understanding and generation*. Similarly, Jiang et al., 2024 emphasise the importance of semantic communication techniques in Pub-Sub systems for effective data-rich routing.

Beyond synthesising previous research, Saleh et al., 2024b underscore the need for future Pub-Sub systems to be autonomous, focused on smart data, dynamic and resilient to adapt to a rapidly evolving landscape. Notably, they stress the absence of a universal "one-size-fits-all" solution, highlighting the necessity to develop application nonspecific approaches.

The work of Saleh et al., 2024a stands as the most comprehensive study on the relationship between Generative AI (GenAI) and Pub-Sub communication. It lists AI models such as the GPT series as having the potential for real-time personalised content delivery (Mann et al., 2020). These potentials are highlighted as topics for further research. Additionally, language-based GenAI is identified as having potential for subscription handling and feedback using natural language. Challenges related to GenAI operations, along with the data produced by such systems, are also discussed, leading to a call for robust architectural solutions. This architecture is described as using the unique advantages of GenAI alongside traditional criteria in Pub-Sub use cases.

More precisely, this architecture is based on a *GenAI agent model* that can perceive and interact with its environment. It consists of three main components: *perception*, *decision-making*, and *actuation*. The perception component is responsible for gathering and presenting information to the decision-making component using tools such as Multimodal-GPT (Gong et al., 2023), Flamingo (Alayrac et al., 2022), HuggingGPT (Shen et al., 2024), AudioGPT (Huang et al., 2024), and VisualChatGPT (Weil et al., 2023). The decision making component, also called the *brain*, handles *analysis*, *short- and long-term memory*, *knowledge*, and *planning* (Xi et al., 2023). It can employ self-criticism and reflection to learn from errors and iteratively improve its performance (Alayrac et al., 2022; Shinn et al., 2024). The actuation component is responsible for taking action based on decisions made by the decision-making component. In particular, the concept of generating tools, executable programmes, is introduced, using frameworks such as CREATOR (Qian et al., 2023).

Dhoni, 2023, analyse GenAi's capability to provide a number of benefits. For example, improved data filtering, proactive work distribution, anomaly detection, data analysis, energy efficiency, automatic corrective measures, and finding optimal times for scaling,

2.3.2 Intelligence in Pub-Sub: Implementations

Isahagian et al., 2023 present a proof-of-concept Pub-Sub system using LLM to improve expressiveness. They match subscriptions to publications using static embedding distance and leverage LLMs to generate personalised natural language notifications for the end user. They identify statefulness and subscription optimisation as topics for future research.

Lee et al., 2024, present a paper-alert system, *PaperWeaver*, that builds on a pre-existing Pub-Sub system for matching new scientific publications to interests expressed by users (researchers). Their unique contribution is the use of LLMs to explain how new articles relate to articles that users have collected previously.

Zaarour et al., 2022 address the challenge of structureless data in content-based peer-to-peer pub-sub systems by proposing OpenPubSub, a content-based approximate semantic system. The system maps a large shared corpus of text into a vector space using distributional semantics. Hierarchical clustering based on dendrograms organises the vector space into semantically related clusters, each assigned to nodes in a peer-to-peer overlay. Publications and subscriptions are mapped to the space using vector distances and an iterative lookup operation. The system includes a semantic event matcher and membership management protocol to group nodes with similar interests into semantic sub-overlays.

Zeeshan et al., 2025 presents a multi-agent complex event processing pipeline integrating multimodal LLMs into a Pub-Sub system to monitor video streams. It combines the Autogen LLM-agent framework with Kafka message brokers. Video streams at different complexity levels are monitored, measuring performance trade-off between latency and narrative coherence. The study finds a correlation between *number of agents + video complexity* and higher *latency*. Narrative coherence maintains high consistency regardless of video complexity. Identified as a key challenge were loss of information and hallucination, as the number of agents increased. For example, agents might call non-existing functions. Another challenge is the choice of the next active agent, currently decided through a hard-coded state machine, becoming increasingly complex as the number of agents increases. This limits the system to using a smaller number of agents.

3 Agentic Pub-Sub Architecture

This chapter engages in an architectural exploration process focused at gathering important requirements and defining the most foundational architectural structures of the Agentic Pub-Sub Architecture. While many quality requirements are taken into consideration, the main focus is on *matching flexibility*.

The Agentic Pub-Sub Architecture draws inspiration from the intelligence components (*perception*, *decision-making*, *actuation*) described in (Saleh et al., 2024a). It also integrates concepts described as central to native intelligence by Tarkoma et al., 2023, such as the MAPE-K model (Monitor-Analyse-Plan-Execute over shared Knowledge) (Gheibi et al., 2021) in combination with LLMs and ReAct (Reason+Act) (Yao et al., 2022). While the perception-decision-actuation dynamic captures the *agentic* part of the architecture, MAPE-K and LLMs provide concrete tools for achieving information flow and reasoning over this information.

3.1 Requirements

Structure Agnostic and Configuration-Free: APSA should allow for autonomous and intelligent content delivery without requiring manual configuration. It should handle *structured*, *semi-structured*, and *unstructured data* from diverse domains. Users provide minimal input, such as an operational context and sample data, while APSA autonomously determines routing strategies.

Efficiency and Scalability: Efficiency and scalability are key quality requirements for Pub-Sub systems (Donta et al., 2022), (Chafi et al., 2020), (Maniezzo et al., 2022). APSA must dynamically balance computational demands, utilising lightweight techniques when possible, and heavier AI-models when needed. The system should adapt based on historical patterns, deploying resource-intensive methods only when necessary. To maximize horizontal scalability and deployability, APSA ought to follow a microservice architecture. An overseer component, leveraging distributed snapshots, maintains a macro-level view, optimizing deployment strategies.

Internal Communication: Internally, APSA should communicate using its own internal Pub-Sub system between components deployed in a distributed system. This could be reminiscent of the natively intelligent Pub-Sub used within the AI-interconnect to handle *resource selection*, *service placement*, and *task coordination* (Tarkoma et al., 2023). In a sense, it could use a simplified version of itself as its communication model leading to a somewhat recursive architecture. In addition, it should also use the *request-response* paradigm for direct messaging.

Context Management: Context management is critical, encompassing both *historical data tracking* and *system environment awareness*. APSA must retain past data flows for analytics and subscription matching, employing relational-, document-based, graph-based, and vector databases as needed. This is needed to handle complex subscriptions asking for information dependant on previous information such as averages. Context can be spread across the system needs using APSA's internal Pub-Sub. LLM context-sharing across distributed nodes can be achieved using shared memory models (Packer et al., 2023), (Rasmussen et al., 2025). APSA also needs to reason about the environmental context. For example, an IoT network that depends on solar energy should adjust its operations based on weather forecasts and predefined user requirements.

Personalised Outputs: Subscribers should define their preferred data format, enabling APSA to tailor responses. Personalisation can range from simple textual summaries (e.g., "List devices with battery <10%") to complex visual representations (e.g., "Generate a map of IoT sensor battery levels"). Techniques from (Zhang et al., 2024), (Feng et al., 2023), and utilization of OpenStreetMap (OpenStreetMap, 2025) can be employed for map-based outputs. In machine-to-machine based communication personalised outputs could consist of a specific data format to enable easy interaction and collaboration between two separate systems. Here, APSA would function as a sort of protocol translator.

Human-in-the-Loop Integration: While APSA needs to be designed for autonomous operation, it must facilitate seamless human intervention when necessary. The system should *minimize the need* for human input but *maximize usability* when interaction is required. Thus, APSA should allow users to interact with it through natural language in combination with an intuitive graphical user-interface. Examples where human input might be needed are *providing context* and *integrating existing routing solutions*.

3.2 Architectural Foundation

The focus of the Agentic Pub-Sub Architecture's architectural foundation presented in this thesis is flexible matching of publications and subscriptions. The underlining idea is to enable *immediate routing of any data with sufficient accuracy and efficiency until an optimised solution has been integrated or generated.*

3.2.1 Core Components

The core structure revolves around a trinity of components: the *Overseer*, *Neural Router*, and *Expert Routers*. Below is a short description of each component and its responsibilities.

- **Overseer**: The brain of the system that analyses information flows to manage system-wide optimizations, routing-strategies and the generating of code.
- **Neural Router**: Able to handle data in a structure-agnostic and configuration-free way, enabling immediate routing without preparation.
- **Expert Routers**: A library of specialized routers optimized for specific data types or domains. The Overseer can utilize these for finding an optimized routing strategy or for storing new custom solutions that have been generated.

In Figure 3.1 we can see how subscriptions and publications enter the APSA system through the Overseer which responsibility it is to *decide the routing option*, *optimize routers*, and *generate new routers*. The idea is that whenever data from a new source enter the system, the Overseer immediately routes it using the Neural Router, which is possible due to its *structure-agnostic and configuration-free* routing capability.

This allows for immediate routing of any data with sufficient accuracy and efficiency. That is, the Neural Router's most important feature is its flexibility, while accuracy and efficiency only need to be acceptable. The Neural Router then continues routing data whilst the Overseer allocates an appropriate amount of resources for finding an optimised solution. This is a balance between immediate content delivery and progressive optimisation.

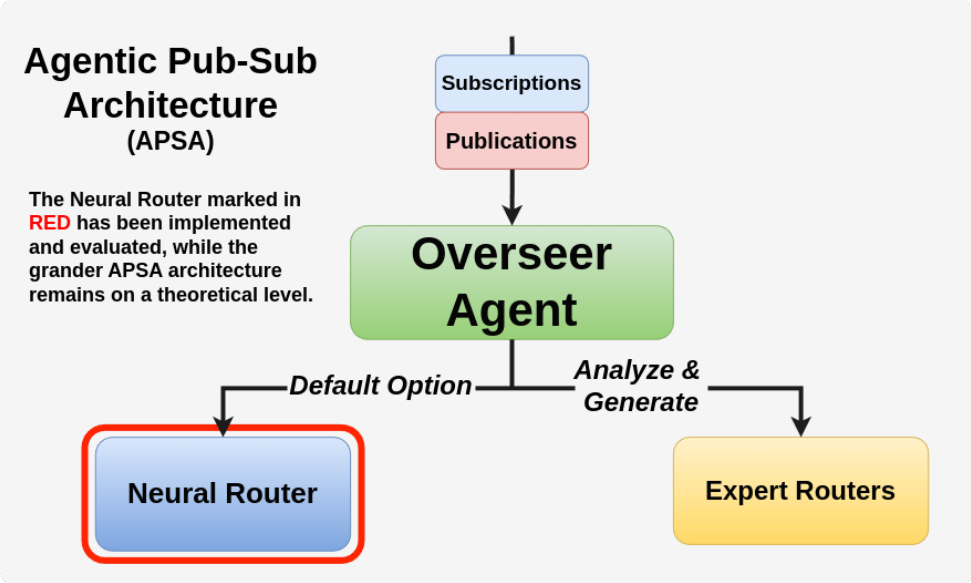


Figure 3.1: The core trinity of APSA

3.2.2 Finding an Optimised Solution

Finding an optimized solution can be done on three main levels. Depending on the specific use-case data the Overseer would choose which level of optimization is needed and whether to move incrementally from partial optimization towards absolute optimization.

The first level of optimization would be to configure the Neural Router to optimize its processes according to the specific data being routed. For an in-depth description of the Neural Router and how it can be configured see Chapter 4.

The second level of optimisation would be for the Overseer to find an existing Expert Router which can be used or modified slightly to function with the current data being routed. Slight modifications could be writing a script to format the data according to the needs of the Expert Router or implementing minor changes to the Expert Router itself.

The third level of optimisation consists of making major changes to an existing Expert Router or generating complete custom solutions based on the use-case data. This would rely on a robust agentic framework for writing, testing, and deploying code.

3.2.3 Context-aware Optimisation

It should be noted that optimisation is a continuous process. That is, the Overseer should continue to look for aspects to improve upon. For example, making routing more accurate, faster, more resource effective, but also making the code more reliable and maintainable. This means that the line between different levels of optimisation might become blurred. If the Overseer initially uses one of the available Routers and begins optimising it, at some point, it may evolve into a custom solution of its own.

What should really decide when to stop optimising is resource usage. For example, if routing is already very accurate, blazing fast, and very resource efficient while also being highly reliable and written in easy-to-understand and maintainable code, there is no point in editing the code. This only increases the risk of errors.

While the previous example, showcases a scenario where optimization stops due to finding a more or less *truly optimized* solution, the Overseer also needs to be able to decide when to stop optimizing even if the solution is still *sub-optimal*. That is, the Overseer needs to take the larger context into play when deciding whether to use available resources for optimization or active operations.

For example, if routing accuracy and efficiency is sufficient, if not optimal, but resources are needed for scaling the routing system or giving personalised responses, etc., optimisation should potentially be de-prioritised. Overarching priorities should be defined by human developers and operators, while the principle of Human-in-the-Loop becomes very important in cases of dynamically changing environments.

4 Neural Router

4.1 System Design

4.1.1 Requirements

The Neural Router is the default router within the greater Agentic Pub-Sub Architecture (APSA). It's purpose is to be able to "*route any data without preparation and with sufficient enough accuracy and efficiency, until an optimised solution has been constructed*". That is, as soon as subscriptions and publications enter APSA it can activate the Neural Router to start the routing of any data. This allows the system to be functional immediately, while autonomous processes can work on an optimised solution.

The Neural Router seeks to strike a practical balance between these competing goals of *flexibility*, *accuracy* and *efficiency*. To achieve a system that is so flexible that it is able to handle diverse domain data in a structure-agnostic and configuration-free manner, the Neural Router employs LLMs in key roles. However, the Neural Router aims to minimize the utilization of LLMs due to high computational resource use. That is, the design aims to use an *overhead of lighter techniques* to limit the usage of the heavier techniques.

The following is a summary of core requirements and why they are important.

- **Flexibility:** The ability of the system to route data from diverse domains in a structure-agnostic and configuration-free manner. This enables immediate routing making quick adoption possible.
- **Efficiency:** The ability of a system to handle input and output quickly using fewer resources. This is critical for scalability and usability.
- **Accuracy:** The ability to create correct matches between subscribers' interests and data from publications. The better the accuracy, the better the system.

4.1.2 Main Processes

The Neural Router consists of three main processes. These are: 1. Subscription-Handling, 2. Publication-Handling, 3. Matching-Handling and can be viewed in Figure 4.1. Here we see the main processes as graded coloured boxes with rounded corners, while the input and output consist of squared boxes.

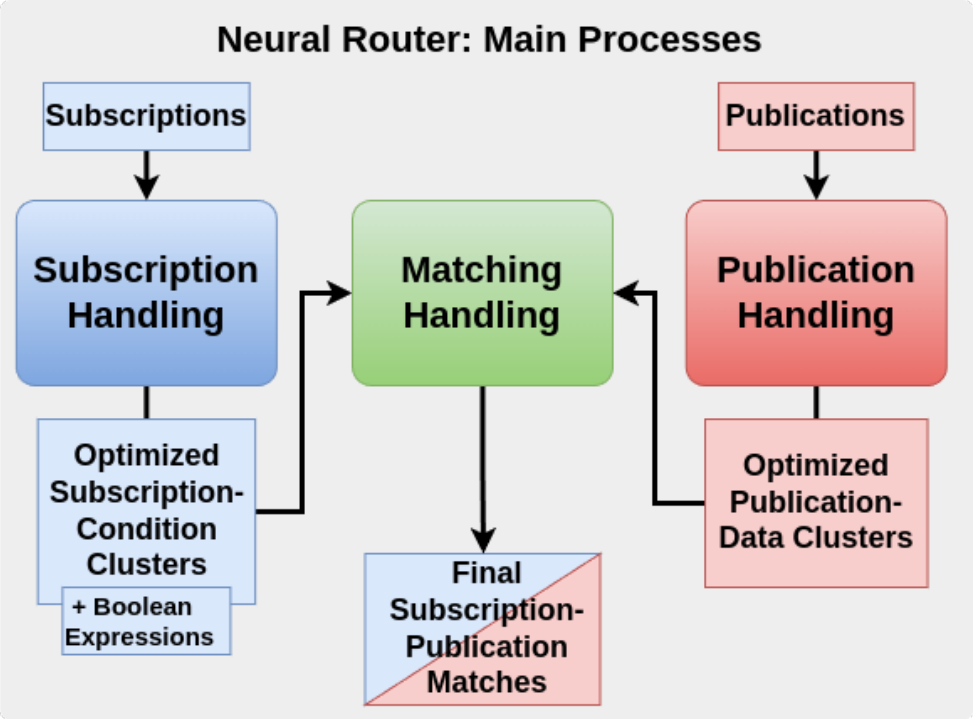


Figure 4.1: An overarching look at the main processes of Neural Router.

Subscription-Handling takes a set of subscriptions as input and outputs a set of *optimized subscription-condition-clusters*, where overlapping conditions have been removed. Clusters consists of semantically similar conditions. Additionally, how the set of individual conditions depend on each is noted in the format of a boolean-expression.

Publication-Handling follows a similar process as Subscription-Handling. As input a set of publications is given and the output consists of a set of *publication-data clusters*. These clusters consists of *optimized publication-data points* where overlapping data has been removed. Each cluster contains semantically similar data.

Matching-Handling takes the subscription-condition clusters and publication-data clusters and outputs final publication-subscription matches. This is done by first matching

individual publication-data against subscription-conditions and then applying the boolean-expressions to make sure *all conditions* are met.

The effect of Subscription-, Publication-, and Matching-Handling is to narrow down the search space of potential matches, before the final matching done in program stage Pub-Sub Matching. The main goal of this is increasing efficiency, but there is also a potential upside to accuracy since eventual matching is simpler with more targeted condition-data matching. Whereas optimisation removes redundancy, clustering limits the direct number of combinations that are considered in the final matching stage.

4.2 Implementation

4.2.1 Programming Environment

The Neural Router is implemented in Python 3.10.12, using a set of third-party libraries which can be seen in Table 4.1 along with an explanation of how the library have been used. The most consequential of these choices are *transformers*, *scikit* and *openai*. Transformers was used to load the model *all-MiniLM-L6-v2* for creating word embeddings. It was chosen for its compactness and efficiency while still having good performance. The scikit library was used for *clustering*. Finding the most suiting clustering framework was a trial and error process. The openai library was used for AzureOpenAi API calls to OpenAi's gpt-models. More details of how these are used are found in the next subsection.

Table 4.1: Libraries and their Usage

Library	**Usage**
numpy 1.26.4	Used to handle vector embeddings.
torch 2.5.1	handle model input/output.
transformers 4.47.0	Loads pre-trained transformer model and tokenizer.
litellm 1.53.7	Simplifies LLM inference.
scikit 2.5.1	Clustering was done using sklearn.
openai 1.57.0	OpenAi API is used for LLM inference.
tiktoken 0.8.0	Counts number of tokens some string contains.

4.2.2 Components

The Neural Router consists of six major components that together are responsible for matching subscriptions to publications. The major components are 1. the *Parser* which extracts key information from input, 2. the *Clusterer* which groups items that are similar, 3. the *Optimizer* which removes overlapping data, 4. the *Cluster Matcher* which matches groups of publication data to groups of subscription conditions, 5. the *Pub-Sub Matcher*, responsible for individual data-to-condition matching, and 6. the *Boolean Enforcer* which applies logical consistency and the final publication-subscription matches. In Figure 4.2, an overview of the execution flow of the Neural Router can be seen, with data being expressed as squared boxes and components as colour-graded boxes with rounded corners. Black arrows indicate the flow of information.

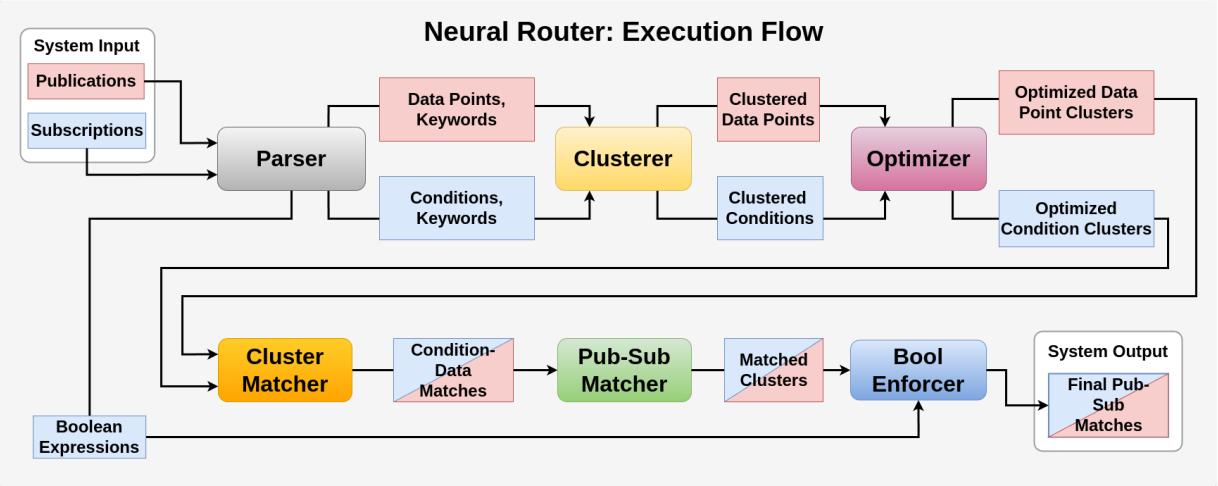


Figure 4.2: An view of Publications and Subscriptions moving through the Neural Router.

The Parser takes either subscriptions and publications as input and parses them into a sensible format. From subscriptions individual conditions, condition-keywords and boolean expressions are extracted. The boolean expression captures how conditions relate to each other. In Figure 4.3 we see an example of a subscription and how it has been parsed into conditions, condition-keywords, and a boolean expression. On the publication side, the Parser extracts data points and keywords. An example of a parsed publication can be seen in Figure 4.4.

```
{
    "subscription_id": 0,
    "subscription": "Inform about devices with temperature less
                     than 20°C or humidity greater than 40%.",
    "conditions": [
        {
            "condition_id": 1,
            "condition": "temperature less than 20°C",
            "keywords": ["temperature"]
        },
        {
            "condition_id": 2,
            "condition": "humidity greater than 40%",
            "keywords": ["humidity"]
        }
    ],
    "boolean_expression": {
        "operator": "OR",
        "operands": [1, 2]
    }
}
```


Figure 4.3: Example of a parsed subscription with an OR-based boolean expression.


```
{
    "publication_id": 1,
    "publication": "Floor: 1,
                    Temperature: 25.0,
                    Channel: 4.0"
    "data": [
        {
            "data_id": 1,
            "data": "Floor 1",
            "keywords": ["Floor"]
        },
        {
            "data_id": 2,
            "data": "Temperature: 25.0",
            "keywords": ["Temperature"]
        },
        {
            "data_id": 3,
            "data": "Channel: 4.0",
            "keywords": ["Channel"]
        }
    ]
}
```


Figure 4.4: Example of a parsed publication with structured data.

The Clustering component is responsible for clustering either subscriptions or publications using the same procedure. The process consists of five main steps. **1.)** First, condition/data keyword vector embeddings are calculated using *all-MiniLM-L6-v2* sentence transformer. **2.)** Determining an optimal number of clusters, k, is done using *K-Means* and *Silhouette Score*. Using the embedding matrix, k is varied from 2 to a Kmax, where Kmax is at most half of the number of keywords (N /2). This is done in max 1000 iterations. **3.)** Once k is identified Gaussian Mixture Model (GMM) is used to produce a set of potentially overlapping soft clusters. **4.)** After creating the initial clusters, a *cluster embedding* is calculated for each cluster, representing the central point in each cluster. **5.)** By using the cluster embedding, very similar clusters are merged using *cosine similarity*, after which the cluster embedding is updated as the average between the merged cluster embeddings. At the end of the clustering process, the clusters are cleaned of duplicate conditions/data within each cluster. This clustering framework has an effect of double-fuzzy clustering where each condition/data point can potentially belong to multiple clusters.

Optimizing the clusters entails eliminating redundancies by removing semantically identical conditions/data points. For example, *temperature above 20°C* and *temp. > 20°C* would lead to one of them being removed. The semantic optimization of publication data and subscription conditions, is done through LLM inference.

The Cluster Matcher takes all publication-data cluster and maps them to the closest matching subscription-condition clusters in a many-to-one fashion to form the eventual search-space for data-condition matching. To form these cluster matches the Cluster Matcher employs *cosine similarity* between *cluster mean embeddings*, which were calculated at the Clustering.

The Pub-Sub matcher takes cluster-matches and uses an LLM to do fine-grained matching of individual publication data-points against subscription-conditions. That is, only data and conditions within cluster matches are matched against each other.

The Boolean Enforcer takes all condition-data matches and makes sure that *all conditions* for a subscription have been met according to their interdependence. This is achieved by utilizing the *boolean expressions* that were extracted by the Parser. In the JSON seen in Figure 4.5 we first see an example of a *parsed subscription*, and how a *publication* has been matched against it using boolean-expression.

```
# PARSED SUBSCRIPTION
{
    "subscription_id": 57,
    "subscription": "Inform about devices with low signal strength
                    (rssi < -90), high temperature (> 18°C), and low
                    humidity (< 40% RH).",
    "conditions": [
        {
            "condition_id": 1,
            "condition": "low signal strength (rssi < -90)",
            "keywords": ["Received Signal Strength Indicator"]
        },
        {
            "condition_id": 2,
            "condition": "high temperature (> 18°C)",
            "keywords": ["temperature"]
        },
        {
            "condition_id": 3,
            "condition": "low humidity (< 40% RH)",
            "keywords": ["humidity"]
        }
    ],
    "boolean_expression": {
        "operator": "AND",
        "operands": [1, 2, 3]
    }
}

# PUBLICATION MATCHED TO SUBSCRIPTION ON INDIVIDUAL CONDITIONS AND DATA.
# Using "boolean_expression" TO VALIDATE THAT ALL CONDITIONS ARE MET.
{
    "publication": {
        "publication_id": 12,
        "publication": "Floor: 2, Readings - Temperature: 23.4,
                        Humidity: 31.0% RH, Network Metrics - RSSI: -107.0"
    },
    "subscription_matches": [
        {
            "subscription_id": 57,
            "subscription": "Inform about devices with low signal strength
                            (rssi < -90), high temperature (> 18°C),
                            and low humidity (< 40% RH)."
        }
    ]
}
```

Figure 4.5: Example of a parsed subscription and its matched publication.

4.2.3 Configuring the Neural Router

While designed to work in a configuration free manner, it is still possible to configure the Neural Router. The current Proof-of-Concept implementation can be configured in the following ways. First, all program stages where an LLM is used are able to use a separate specified model. Second, the *Token Limit Per Call* can be set to different values. This controls how much data is used within a single prompt.

4.2.4 LLM Inference Heuristics

The LLM inference done throughout the Neural Router follows the same structure with some additional steps in certain components. The first commonality is that prompts are always generated using the same procedure. Data (*subscriptions/publications* etc.) is given as a list which is passed to the prompt generator which divides the data up into batches with max size *Total Tokens Per Call*. Each batch is then combined with predefined instructions specific to the component. Here, the prompt is also combined with any meta-data about the dataset which would be provided by the user. Finally, all batches are sent in parallel asynchronous calls to the LLM API using formatted JSON defined using pydantic models.

Another heuristic used whenever using LLM inference in the Neural Router is validating that all data in the input is also included in the output. Any missing data is gathered and resent to the LLM. This validation step makes the Neural Router more robust against LLM hallucinations.

The final heuristic used is *pseudo-identifiers*. This is done partly due to necessity when passing subscription conditions or publication data points by themselves to the LLM without their parent subscription/publication. That is, if not substituting these ids, there would exist multiple ids with the same numeric value. An alternative would be to pass conditions/data along with their parent subscription/publication. However, using pseudo-identifiers has the positive effect of minimising token usage while also simplifying the data, potentially giving more reliable LLM inference. Pseudo-identifiers are used in *subscription-optimisation*, *publication-optimization*, and *pub-sub matching*.

5 Neural Router Evaluation

This chapter describes how the Neural Router has been evaluated. The evaluation considers the Neural Router's core requirements *flexibility*, *efficiency*, and *accuracy*. First, a description of how each requirement is measured is given. Then the datasets used for evaluating the Neural Router are explained. After this, the evaluated configurations of the Neural Router are described, and key aspects of the execution environment are listed.

5.1 Parameters for Evaluation

5.1.1 Accuracy

For evaluating the accuracy of the Neural Router, a confusion matrix and derived metrics are used. A **confusion matrix** summarizes the performance of a classification model by comparing the predicted labels with the true labels. It consists of four core metrics:

- **True Positive**: When the system has identified a match as True and the system is correct. For example, when the system matches subscription *temperature > 20°C* and publication *24°C*, would count as a True Positive.
- **False Positive**: When the system has identified a match as True, but the system is incorrect. For example, when the system matches subscription *temperature > 20* and publication *18°C* it would count as a False Positive.
- **True Negative**: When the system has identified a match as False and the system is correct. For example, the system recognises subscription *temperature > 20°C* and publication *18°C* as *not matching* would count as a True Negative.
- **False Negative**: When the system has identified a match as False, but the system is incorrect. For example, the system *doesn't match* the subscription *temperature > 20°C* and publication *24°C* would count as a False Negative.

That is, True Positives and True Negatives are cases where the system is functioning accurately, while False Positives and False Negatives are cases where the system is not.

From the confusion matrix, it is possible to calculate a set of derived metrics. For evaluating the Neural Router we use the following: *Precision*, *Recall*, *F1-Score*. The following bullet list explains the metrics briefly.

- **Precision**: Precision quantifies *how many of the predicted positive matches are actually correct*:
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}.}$$
A high precision indicates that the system is not producing a high number of false positives. That is, the system doesn't over-match.

- **Recall**: Recall quantifies *how many of the actual positives were correctly predicted*. It is given by:
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}.}$$
A high recall demonstrates that the router does not produce many false negatives. That is, the system doesn't under-route.

- **F1-Score**: The F1-Score is the *harmonic mean of precision and recall*, providing a balanced measure when there is a trade-off between over-routing and under-routing. It is defined as:
$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}.}$$
A high F1-Score indicates a good balance between precision and recall.

5.1.2 Efficiency

To evaluate the execution efficiency of the Neural Router, we measure *Execution Time*, *LLM Inference Time*, *Latency*, *Total Token Usage*, and *Token Distribution*. Below is a quick explanation of what each metric entails.

- **Execution Time**: The absolute time taken to run a programme or part of a programmeme. The execution time of each stage of the programme is recorded separately for granular analysis.

- **LLM Inference Time**: The round trip execution time of remote LLM API calls. The time from parallel calls are aggregated.

- **Latency**: The time taken to handle a single subscription or publication. In this thesis, subscription-handling latency and publication-handling+matching-handling latency *combined*, henceforth referred to as *real-time latency* are measured separately.
- **Total Tokens**: Counting the number of tokens used can be used to estimate the computational load Bharath et al., 2025, and is the primary metric to estimate the usage of LLM resources within this thesis.
- **Input- & Output Tokens**: Input Tokens consist of the input prompt to the LLM while output tokens stem from the LLM response. The weight of input and output tokens is not equal. Cutting input tokens by 50% might improve latency by 1-5%, cutting output tokens by the same amount can lead to improvements of around 50% (OpenAI, 2025). In addition, the monetary cost of input tokens is significantly cheaper than that of output tokens.
- **Data- & Instruction Tokens**: Input Tokens can be split into Data- and Instruction Tokens. Data tokens are the result of data being passed to the LLM, while instruction tokens are the result of *instructions on how to handle data*. Separation of these can give an indication of how effective the instructions are.

In a Pub-Sub system, each metric plays a critical role in ensuring efficient operation. Notable is that common metrics such as *cpu-usage*, and *memory-usage* has been left out. This is due to a number of reasons. LLM inference is performed in the proprietary cloud, making it impossible to measure these metrics. While it would be possible to measure cpu and memory usage for the local operations, this would not capture the computationally heaviest technique of LLMs, blurring the picture. Additionally, measuring local cpu-usage, and memory-usage would clearly be tied to the specific hardware. While the same could be said about execution time and latency, they do capture the LLM inference time, making an analysis of trends and patterns possible. Thus, in the interest of time, the multifaceted analysis that would be needed to include cpu-usage and memory-usage was decided to be left out.

Another aspect which has only been taken into account to a degree is the network latency from LLM API calls. More precisely, no exact method has been employed to capture how much of the metric LLM Inference Time is *actual LLM inference* vs. network latency. However, when doing the experiments, iterations where LLM Inference Time was radically

higher than typical values for that programme stage, were run again. No part of the programme had to be rerun more than once.

5.1.3 Flexibility

Flexibility of a Pub-Sub system has within this thesis been defined as being able to handle diverse domain data in a structure-agnostic and configuration-free way. To achieve this we test the Neural Router using two datasets (see Section 5.2), representing opposites on the structuredness spectrum. Subscriptions and publications enter the Neural Router in their raw form. In Figure 5.1, we see an example of the format of a subscription entering the system for the first time. Here, all subscription data is included as a single string.

```
{
    "subscription_id": 0,
    "subscription": "Inform about devices with temperature less
                     than 20°C or humidity greater than 40%."
}
```

Figure 5.1: Format of subscription when entering the Neural Router. Publications follow an identical structure.

Additionally, the Neural Router is evaluated on these two datasets using the exact same configuration. The only aspect of the Neural Router that could be considered to be differently configured between the datasets is that of *user-input* consisting of meta-data about each dataset that is used in LLM prompts.

5.2 Description of Datasets

In this section the two datasets used to evaluate the Neural Router are described. The common feature between the usage of these datasets is that a total of *1000* unique publications were randomly selected from each dataset to be used for the experiments.

5.2.1 CardiffNlp: Ambiguous Natural Language

The first dataset used to evaluate the Neural Router consists of real-world social media posts originally collected and labelled according to topics by CardiffNlp (Antypas et

al., 2022), henceforth referred to as *CardiffNlp*. This dataset was originally meant for testing and training natural language classification models, but has been repurposed to function as publications and subscriptions. More precisely, social media posts are used as publications. On the subscription side, the original 19 unique topic-labels have been expanded to constitute a set of 114 subscriptions. The 114 subscriptions are based on the original 19 unique topic labels, garnering six subscriptions per original topic label. More precisely, the subscriptions consist of four *synonyms* or *rephrasings*, and two copies of the original topic label. Using these groups of semantically similar or identical subscriptions guarantees the ability of the Neural Router to cluster, optimise, and reason semantically about data. An example of two original topic labels and their expanded subscriptions can be seen in Table 5.1. In addition, a table with all original topic labels and their subscriptions can be found in Appendix A.

Table 5.1: Topic Labels and Subscriptions

Topic-label	**Subscriptions**
arts_&_culture	art and culture, art, culture, culture and art, cultural art
entrepreneurship	entrepreneurship, entrepreneurial, entrepreneurialism, entrepreneurialism, entrepreneurs

When evaluating the Neural Router using CardiffNlp, the ground truth consists of the original topic-labels. The relation between original topic-labels and the expanded subscription set is kept track of by having both unique identifiers for all subscriptions and group identifiers and a mapping function to flip back and forth between these.

Using the CardiffNlp dataset evaluates the Neural Router's ability to route ambiguous natural language. While natural language itself can already be ambiguous, the casual language commonly used on social media might often be even less clear. This means that the accuracy results must be considered approximate, while manual analysis of the results gives a better understanding of the practical accuracy . In Figure 5.2, an example of a CardiffNlp publication can be seen along with ground-truth subscription matches. This is an example where the ground truth is not black and white. For example, one could easily imagine switching out *fitness & health* to *science & technology* which is another of the original topic labels that was not chosen by the original creators of the dataset.

```
{
        "publication_id": 4,
        "publication": "Coronavirus: social distancing puts the future of
                        drone delivery to the test {{URL}} via
                        {{USERNAME}}",
        "subscription_matches": [
            {
                "subscription": "news_&_social_concern",
                "subscription_id": 64
            },
            {
                "subscription": "fitness_&_health",
                "subscription_id": 2048
            }
        ]
}
```


Figure 5.2: CardiffNlp publication with ground truth subscription matches.

5.2.2 SmartCampus: Structured Numerical Data

The second dataset used to evaluate the Neural Router, called SmartCampus, consists of readings from 429 IoT sensors deployed at the University of Oulu (Eldeeb, 2023). It evaluates the Neural Router's ability to route structured numerical data. In Table 5.2 the different readings present in the dataset can be seen. Readings are gathered at gateway devices with eventual publications consisting of these combined set of readings. At gateways, *timestamps* and *unique_ids* are also added to publications. Column name was used in both publications and subscriptions. For the experiments, the original unique_ids were substituted with *psuedo_ids* before entering them into the system. While a heuristic commonly used within the Neural Router this was prepared for easier validation of results.

The SmartCampus dataset lacks subscriptions all together. To create subscriptions, the following process was used to generate a set of 64 subscriptions: **1.)** All SmartCampus publications were inserted into an SQL database. **2.)** Summary statistics (max, min, mean) were calculated and stored for the columns. **3.)** LLM inference was used to generate SQL queries to retrieve publications. The prompt included meta-data and summary statistics to generate valid queries. **4.)** The SQL query was used to fetch publications. If this did not return a sensible number of matches (1-5 000 000), the process went back to Step 3. **5.)** If the query returned a sensible number of matches, the query was inserted into Table Subscriptions. **6.)** For each publication match, the subscription ID was inserted into the column *subscription_matches*. This tracks the ground truth.

Table 5.2: Sensor Data Descriptions

Description	**Column name**	**Unit**
Air temperature	temperature	[°C]
Relative humidity	humidity	[% RH]
Light	light	(linear index)
Passive infrared	motion	(linear index, pir)
Co2	co2	[ppm]
Battery voltage	battery	[V]
Average SPL	sound_avg	[dB]
Peak SPL	sound_peak	[dB]
Soil moisture	moisture	[%]
Atmospheric pressure	pressure	[bar]
Acceleration (X)	acceleration_x	float
Acceleration (Y)	acceleration_y	float
Acceleration (Z)	acceleration_z	float

Additionally, the above process for generating subscriptions takes parameter *query_complexity*. For the experiments of this thesis, 16 *simple* and 48 *composite* subscriptions were generated. A simple subscription constitutes a single condition, while a composite subscription includes multiple conditions.

In Figure 5.3 a SmartCampus publication can be seen along with its ground truth subscription matches. Unlike CardiffNlp, the SmartCampus ground truth is unambiguous. However, an unintended detail introduced an additional layer of evaluation. While all system prompts refer to *publication_id*, the subscriptions in SmartCampus instead use *advertisement_id*. Although this terminology mismatch was unintentional, such inconsistencies can occur in real-world scenarios and actually strengthen the evaluation of the Neural Router's ability to do *intent-driven* data routing in an intelligent way.

```
{
        "publication_id": 3,
        "publication": "Timestamp: 2021-04-24T12:56:02.554000+00:00,
                        Floor: 2
                        Readings -
                        Temperature: 21.3°C,
                        Humidity: 25.0% RH,
                        Light: 603.0,
                        Motion: 0.0,
                        Co2: 469.0 ppm,
                        Battery: 3.693V",
        "subscription_matches": [
            {
                "subscription": "SELECT advertisement_id FROM
                                 advertisements WHERE floor = 2",
                "subscription_id": 17
            },
            {
                "subscription": "SELECT advertisement_id FROM
                                 advertisements WHERE light > 100
                                 AND humidity < 40 AND
                                 temperature > 20",
                "subscription_id": 63
            }
        ]
}
```


Figure 5.3: CardiffNlp publication with ground truth subscription matches.

While the generated SQL queries are used as subscriptions in the experiments for this thesis, a set of secondary subscriptions was also produced. These consist of natural language versions of the SQL queries and can be used in future experiments.

5.3 Evaluated Configurations

To prove the flexibility and preparation-free nature of the Neural Router, experiments were performed on the two datasets using identical configurations. That said, the Neural Router was evaluated with different configurations to see how it affects the system.

The first category of parameters is *which LLM* is used. Here the Neural Router was evaluated using the smaller *gpt-4o-mini* vs. the bigger *gpt-4o*. The number of LLM parameters has a direct positive relationship with performance (Kaplan et al., 2020). While proprietary to OpenAI, the number of parameters for gpt-4o is approximated to be in the hundreds of billions, while gpt-4o-mini is about eight billion (Li, 2025). Evaluation

using different sized models makes sure to evaluate the performance of the *Neural Router system*, apart from LLM model power.

The second category where configuration changes have been made is a *Token-Limit-Per-Call* to the LLM. That is, we set different limits for how many tokens are passed to the LLM in a single prompt. More precisely, we are controlling the maximum amount of data tokens that are combined with instruction tokens in each prompt. The token limits used for the experiments in this thesis are *100*, *500*, and *1000* tokens. This measures how the size of the prompt affects accuracy and efficiency.

The six unique configurations are:

1: *gpt-4o-100* **2:** *gpt-4o-500* **3:** *gpt-4o-1000*
4: *gpt-4o-mini-100* **5:** *gpt-4o-mini-500* **6:** *gpt-4o-mini-1000*

5.4 Execution Environment

The experiments were run on a desktop with the specifications found in Table 5.3.

Table 5.3: Experiments were run on a Desktop with these specifications.

Component	**Specification**
Processor	AMD Ryzen 5 5600G
Cores/Threads	6 cores, 12 threads
GPU	NVIDIA GTX 1660 SUPER, AMD Radeon Graphics
Memory	16 GB DDR4 (2×8 GB)
Operating System	Ubuntu 20.04.3 LTS

6 Results

The evaluation of the Neural Router includes measuring its capabilities according to *routing accuracy*, *execution efficiency*, and *flexibility*. Routing accuracy relies on a *confusion matrix*, efficiency uses *Execution-Time*, *LLM Inference Time*, *Latency*, and *Token Usage*. To evaluate the flexibility of the Neural Router, it was evaluated using identical configurations on two datasets representing the opposites on the structuredness spectrum.

In Section 6.1, the Neural Router experiment results on the CardiffNlp dataset are presented. Here, the structure of data visualisations is also explained. In Section 6.2, the results on the SmartCampus dataset are presented using identical data visualisations as with CardiffNlp. This section also presents cross-dataset trends briefly, while an in-depth analysis of the cross-dataset performance of the Neural Router is explored and discussed in more detail in Chapter 7. The results presented are the result of mean values from five independent iterations of the Neural Router, where each programme stage is monitored separately. The efficiency metrics presented in this chapter capture the most essential aspects of the Neural Router, with exhaustive results found in Appendix B.

6.1 Neural Router Results on CardiffNlp

6.1.1 Accuracy

To present the accuracy of the Neural Router, a set of plots have been created that visualise the accuracy across the different configurations used to evaluate the system. The *Precision*, *Recall*, and *F1-Score* can can be viewed in the conglomerate Figure 6.4. Here, we see one bar plot per metric, each containing six bars, and every bar representing one configuration of the Neural Router. The bars are divided into two groups of three. The groups indicate the used *LLM model*, while the three bars indicate the *Token-Limit-Per-Call*. The exact configuration is displayed under each bar.

From the accuracy metrics, we can see a number of trends for the CardiffNlp dataset. First, the difference between using *gpt-4o* and *gpt-4o-mini* is negligible, as the results

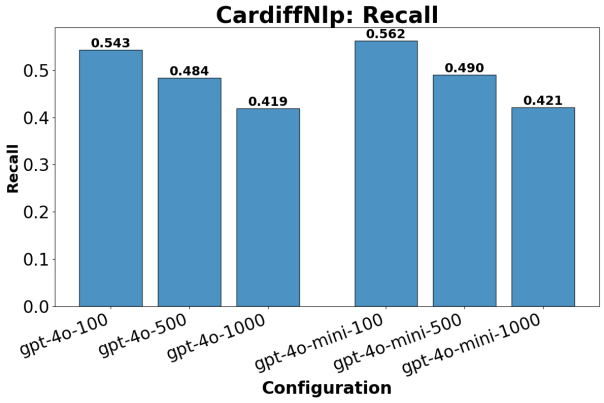


Figure 6.1: Recall on CardiffNLP

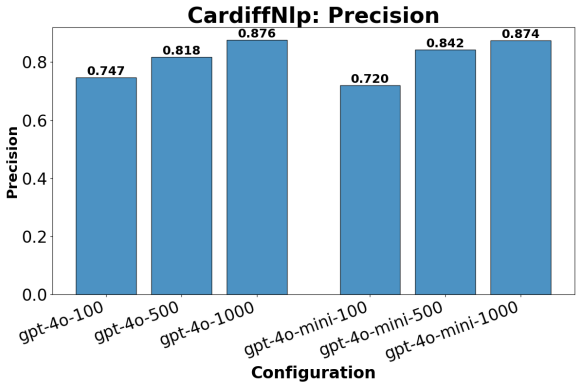


Figure 6.2: Precision on CardiffNLP

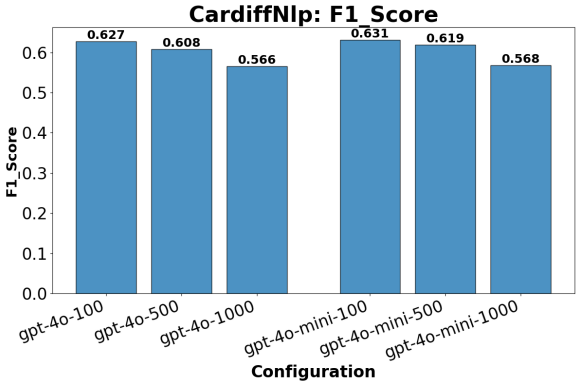


Figure 6.3: F1 Score on CardiffNLP

Figure 6.4: Performance Metrics on CardiffNLP Dataset

show very similar absolute numbers and show the same trends across Token-Limit-Per-Call configurations. Second, with increasing *Token-Limit-Per-Call*, a pattern of higher *precision* seen in Figure 6.2, but lower *recall* seen in Figure 6.1, can be seen. That is, as the LLM handles more data in a single prompt, it produces fewer matches as a whole, including both correct and incorrect matches. However, this trend is stronger when considering *recall*, leading to an overall negative trend for *F1-Score*, seen in Figure 6.3, as the Token-Limit-Per-Call increases. The highest overall accuracy (F1-Score) is *0.631* and is achieved configuration *gpt-4o-mini-100*. The lowest F1-Score is *0.566*, and was produced using configuration *gpt-4o-1000*.

6.1.2 Efficiency

The efficiency metrics used to evaluate the Neural Router are different *execution time* metrics as well as *token usage*. To be able to evaluate the system at a detailed level, each component was evaluated separately. Due to the disparity between the number of *subscriptions*, *114 in CardiffNlp* compared to the number of *publications*, *1000*, an overwhelming majority of resources are used in *publication-handling* and *matching-handling* compared to *subscription handling*.

In conglomerate Figure 6.10, the *Execution Time*, *LLM Inference Time*, *Subscription-Handling Latency*, *Real-time Latency*, and *Total Token Usage* are visualised. *Real-time* in this context means *Publication-Handling + Matching-Handling*. Each plot consists of stacked bars to visualise a metric for every stage of the Neural Router separately. Each stage has its own colour explained in the legend. As with the plots for accuracy, each bar signifies a single Neural Router configuration. The six bars are grouped into two sets of three defining *chosen LLM configuration*. These plots use numbers *1-6* to codify the configurations, which can be found in the legend.

From the plot in Figure 6.5, visualising *execution time*, we can see that most of the time goes to stages *subscription-parsing*, *publication-parsing*, *publication clustering*, *publication-optimisation*, and *pub-sub matching*. Especially time-consuming is *publication-clustering*. There is a trend where *publication-clustering* execution-time decreases as the Token-Limit-Per-Call increases. This is an emerging behaviour, as Token-Limit-Per-Call has *no direct* effect on the publication clustering process.

Important to note about Execution Time is that LLM inference is performed in the cloud with limitless parallel invocations. This means that the disparity in the number of *publications* and *subscriptions* is not directly reflected in the execution time in addition to any additional network congestion.

To understand the true LLM Inference Time, see Figure 6.6, where the aggregated *LLM inference time* for each stage is visualised. Here, the disparity between the number of publications (1000) and subscriptions (114) becomes clear as the vast majority of LLM inference time is found in stages where publications are handled (*publication-parsing, publication-optimisation, pub-sub matching*).

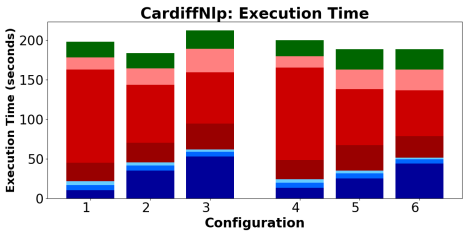


Figure 6.5: Execution Time on CardiffNlp

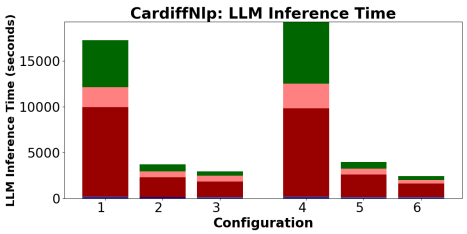


Figure 6.6: CardiffNlp: LLM Inference Time

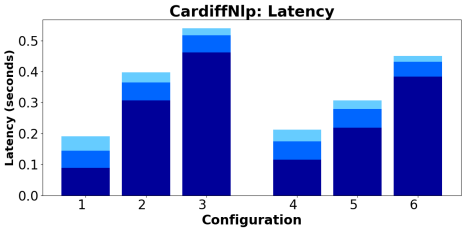


Figure 6.7: CardiffNlp: Subscription Handling Latency

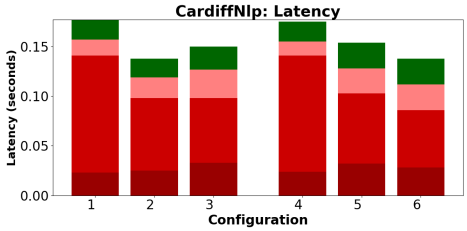
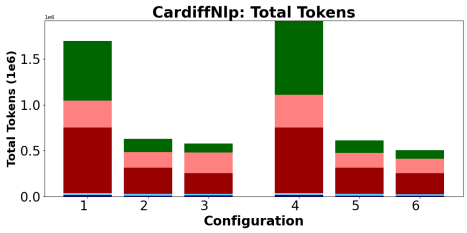


Figure 6.8: CardiffNlp: Publication- and Matching-Handling Latency



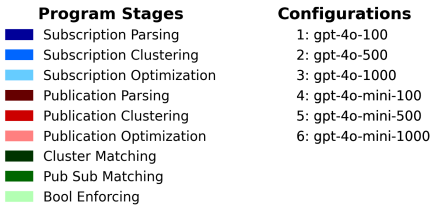


Figure 6.9: CardiffNlp: Total Tokens

Figure 6.10: Efficiency metrics on CardiffNlp Dataset

The *Subscription-Handling Latency* of the Neural Router on CardiffNlp can be viewed in Figure 6.7. Here we can see that the latency of handling a single subscription increases as the Token-Limit-Per-Call increases. This is true when using both *gpt-4o* and *gpt-4o-mini*. In general *gpt-4o-mini* is somewhat faster. The best Subscription-Handling Latency is however achieved with configuration *gpt-4o-100* and lays just under 200 milliseconds.

In Figure 6.8, it can seen that Real-Time-Latency is across the board lower than Subscription-Handling Latency. The Real-Time Latency ranges from around 140 milliseconds to 170 milliseconds. The trends of Real-time Latency is somewhat inconclusive. Configurations 1 and 3, along with 4 to 6 indicate a *decrease* in latency as Token-Limit-Per-Call *increases*. However, this is not true for configuration 2, breaking a perfect pattern.

The *Total Token Usage* can be seen in Figure 6.9. Mirroring the pattern seen in *LLM Inference Time*, Publication-Handling and Matching-Handling dominate in Total Token Usage, due to the higher data loads seen in these programme stages. The specific LLM model used has a minor effect on the total number of tokens used. A sharp drop in Total Token Usage can be seen between configurations 1 to 2 and 4 to 5, with a further smaller drop from 2 to 3, and 5 to 6.

Finally, the *Token Distribution* of the Neural Router on CardiffNlp can be viewed in Figure 6.11. Here, *total-*, *instruction-*, *data-*, and *output tokens* are separated in six pie-charts representing every Neural Router configuration. From this, it is clear that the specific LLM has a negligible effect on the token distribution. The second major trend that can be seen is that as the Token-Limit-Per-Call increases, the share of data-tokens increases. A final pattern that can be seen is a sharp rise in output-token share from a token-limit-per call *100* to *500*.

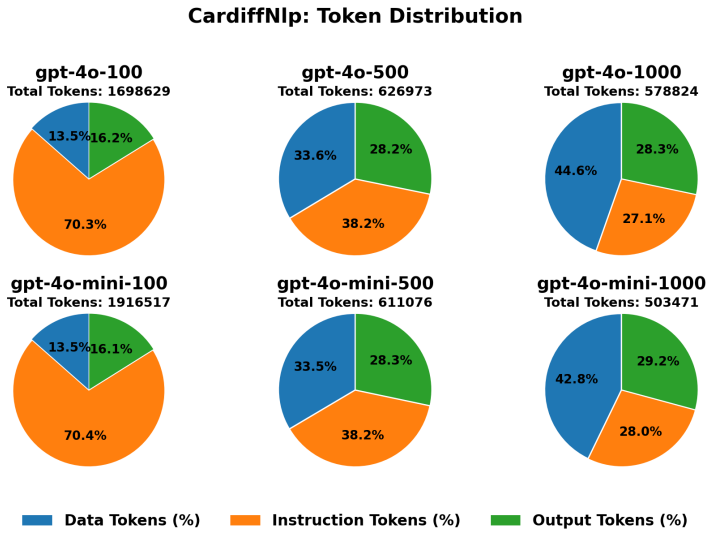


Figure 6.11: CardiffNlp: Token Distribution

6.2 Neural Router results on SmartCampus

6.2.1 Accuracy

The accuracy results for the Neural Router on the SmartCampus dataset can be viewed in collection Figure 6.15. The figures follow an identical structure as those presenting the accuracy of the Neural Router on CardiffNlp.

As with CardiffNlp, the specific LLM used has negligible impact on accuracy metrics. Unlike the trend seen on CardiffNlp, with a falling accuracy as the *token-limit-per-call* increases, on SmartCampus there is no such pattern. More precisely, on SmartCampus all accuracy metrics are very stable across all configurations. On SmartCampus, the Neural Router achieves a high F1-Score of *0.900* with configuration *gpt-4o-mini-500* and a low F1-Score of *0.878* with *gpt-4o-mini-100*.

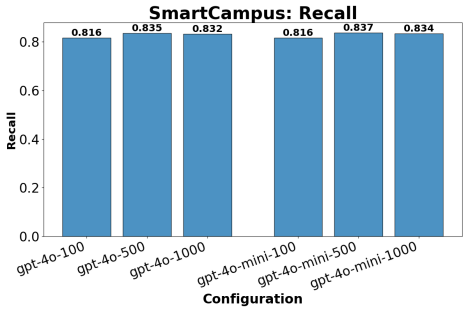


Figure 6.12: Recall on SmartCampus

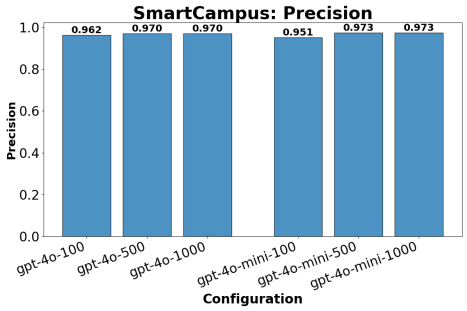


Figure 6.13: Precision on SmartCampus

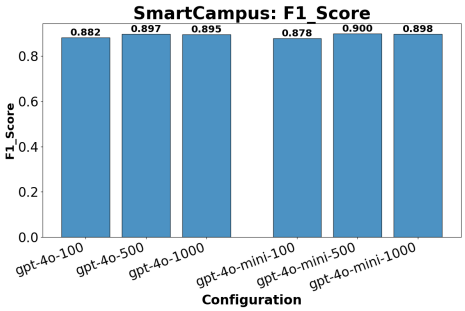


Figure 6.14: F1 Score on SmartCampus

Figure 6.15: Accuracy Metrics on SmartCampus Dataset

6.2.2 Efficiency

In conglomerate Figure 6.21 the efficiency metrics on SmartCampus can be seen. Here, *Execution Time* is found in Figure 6.21, *LLM Inference Time* in Figure 6.17, *Subscription-Handling Latency* in Figure 6.18, *Realtime-Latency* in Figure 6.19, and *Total Token Usage* in Figure 6.20. The following paragraph lists the many similarities between results on CardiffNlp and SmartCampus, while noting minor differences.

1.) The majority of resources are used in *publication-handling* and *matching-handling*, due to the larger amounts of data found here. This can be seen in absolute *Execution*

Time, *LLM Inference Time*, and *Total Tokens*, where *Subscription-Handling* programme stages contribute a small fraction of all resource usage. **2.)** Execution Time and Real-time Latency drops with configuration 2 and 5 on both datasets. On SmartCampus configuration 6 sees an increase in Execution Time and Real-time Latency. **3.)** The total *LLM Inference Time* is the highest when *Token-Limit-Per-Call* is 100 and falls significantly when switched to 500. While already significant on CardiffNlp, the fall is even sharper on SmartCampus. **4.)** The latency of handling subscriptions increases as the Token-Limit-Per-Call increases and is higher than handling a single Publication and Matching. **5.)** Total Tokens used do not depend on the specific LLM used, even less so on SmartCampus. **6.)** Total Token Usage is the highest with configurations *gpt-4o-100* and *gpt-4o-mini-100* and falls sharply when the Token-Limit-Per-Call is changed to 500. With Token-Limit-Per-Call changed to 1000 there's a continued drop in Total Token Usage.

One core difference between the handling of CardiffNlp and SmartCampus dataset is that the absolute values are higher in all categories. That is, Execution Time is higher, LLM Inference Time is higher, both Latency metrics are generally higher, and Total Token Usage is higher.

Finally, the *Token Distribution* of the Neural Router on SmartCampus and can be viewed in Figure 6.22. Here, it can be seen that the trends are similar to those seen in the token distribution on CardiffNlp. First, the specific LLM used has a negligible effect on token distribution. Second, increasing the Token-Limit-Per-Call from 100 to 500 sees a big proportional increase in data-tokens used. As with CardiffNlp, there is a sharp rise in output-token share from a token-limit-per call *100* to *500*.

6.3 Flexibility of the Neural Router

The Neural Router achieves sufficient accuracy and efficiency in handling both natural language and structured numerical data using identical configurations. While there is no single configuration that performs optimally at all tasks, *gpt-4o-mini-500* and *gpt-4o-mini-1000* stand out as the configurations that provide the most balanced approaches.

While gpt-4o-mini-1000 is slightly more efficient on CardiffNlp, gpt-4o-mini-500 is most efficient on SmartCampus. On the other hand, gpt-4o-mini-500 achieves the best F1-Score of all configurations on SmartCampus, and is only 2% gpt-4o-mini-100 on CardiffNlp.

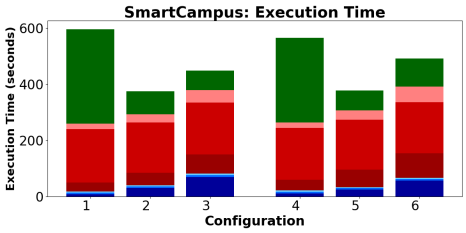


Figure 6.16: Execution Time on SmartCampus

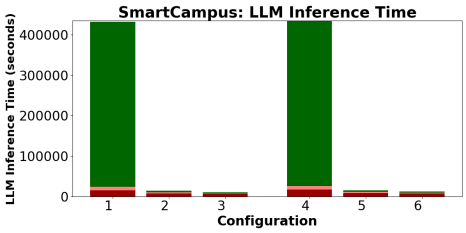


Figure 6.17: CardiffNlp: LLM Inference Time

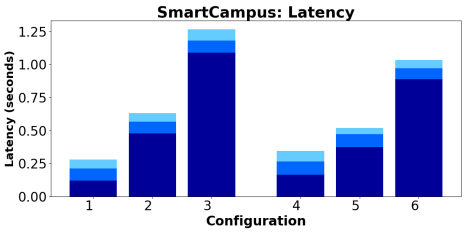


Figure 6.18: SmartCampus: Subscription Handling Latency

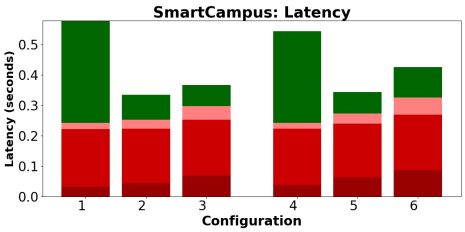
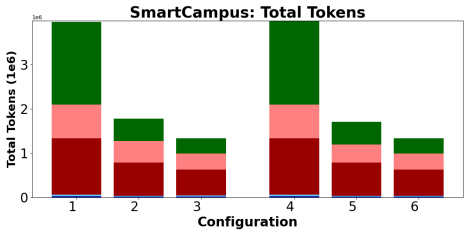


Figure 6.19: SmartCampus: Publication- and Matching-Handling Latency



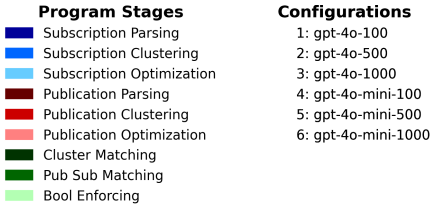


Figure 6.20: SmartCampus: Total Tokens

Figure 6.21: Performance on SmartCampus Dataset

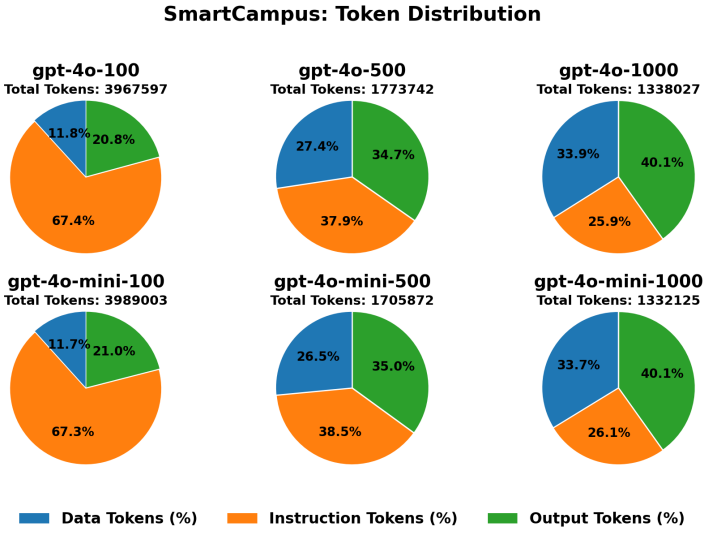


Figure 6.22: SmartCampus: Token Distribution

7 Discussion

In this chapter, the results of the evaluation of the Neural Router are analysed and placed in context. Additionally, potential shortcomings and future improvements for the Neural Router are suggested. Finally, an analysis of use cases is given considering both current results and potential improvements.

7.1 Insights from Results

The philosophy of the Neural Router is to be able to *route any data with sufficient enough accuracy and efficiency until an optimized solution is provided.* To prove the feasibility of a routing strategy capable of achieving such a system, this thesis evaluated the Neural Router on two datasets representing opposites on the structuredness spectrum. The first dataset, CardifNlp, consisting of social media posts, tests the Neural Router's ability to handle ambiguous natural language, while the second dataset, SmartCampus, consisting of IoT sensor readings, represents structured numerical data.

7.1.1 CardiffNlp: Ground Truth Ambiguity

While the Neural Router achieved an F1-Score of 0.900 on SmartCampus, the accuracy on CardiffNlp was only 0.631. However, when examining the CardiffNlp results manually, it was noticed that the ground truth provided by the original dataset is even less clear than anticipated. More precisely, it was found that an overwhelming majority of the time, the results produced by the Neural Router were *just as correct if not even more correct* than the ground truth. For example, in Figure 7.1, we see a match produced by the Neural Router enforced by the ground truth. Here we see that the ground truth has also included subscription *science_&_technology* for the publication, but it could be argued that this is a bit of a stretch and that the true positive *news_&_social_concern* is more reflective on its own.

With many of the False Positives and False Negatives being due to ambiguities in the ground truth, it means that the *practical accuracy* of the Neural Router on CardiffNlp,

might actually be much higher than what the pure F1-Score suggests. In future experiments custom subscriptions based on specific publications should be created for a more precise evaluation.


```
{
    "publication": {
        "publication_id": 1,
        "publication": "Conservative activist family behind grassroots
                        anti-quarantine Facebook events - Democratic
                        Underground {{URL} via {{USERNAME}}"
    },
    "true_positives": [
        {
            "subscription_id": 64,
            "subscription": "news_&_social_concern"
        }
    ],
    "false_positives": [],
    "false_negatives": [
        {
            "subscription_id": 8,
            "subscription": "science_&_technology"
        }
    ]
}
```


Figure 7.1: Example of a parsed publication with classification results.

7.1.2 Hallucination Challenge

Manually studying *False Positives* and *False Negatives* produced by the Neural Router on the non-ambiguous SmartCampus dataset reveals that the number one issue is LLM hallucination. Specifically, this includes merging subscriptions/publications where they shouldn't be merged as well as under/over matching. The struggle of LLMs to handle numerical data is a well-known issue related to hallucination (Xie, 2024).

Additionally, according to the manual inspection of False Positives and False Negatives produced by the Neural Router on SmartCampus, hallucination does not seem to follow a specific pattern. That is, while one iteration of the experiments can produce hallucinations on some data, another iteration will handle that data correctly.

7.1.3 Token-limit Optimum

Another key finding from the results is the identification of some kind of *optimum* for the *token-limit-per-call*, laying at around *500* tokens. This is evident from multiple accuracy and efficiency metrics, where accuracy is the highest or almost just as high, while being significantly more efficient when using Token-Limit-Per-Call *500*. For example, the F1 score is only *-0.012* lower than the best F1 score on CardifNlp, and *is* the best with a token limit per call of 500 on SmartCampus. At the same time, we see a drop in *Execution Time*, *Real-time Latency*, *LLM-inference time*, and *total token use*.

The only exception to this optimum is Subscription-Handling Latency which is the lowest with token-limit-per-call at *100*. While, the results show that token-limit-per-call *500* seems to perform the best as a general rule, there is no reason why different token-limits could not be used at different programme stages. For example, Subscription-Handling could use token-limit-per-call *100*, while the other stages use *500*.

7.1.4 Data Intensity

Another finding from the results comes from comparing the performance metrics between CardiffNlp and SmartCampus. SmartCampus is generally a dataset that is much heavier to handle seeing higher Execution Times, LLM Inference Times, Latency, and Token Usage. The explanation for this is that SmartCampus is a much more data-intensive dataset, when compared to CardiffNlp.

For example, while one publication in CardiffNlp may be parsed into two or three data parts, SmartCampus publications are often divided into 15 data parts. This means that while originally consisting of 1000 publications from both datasets, programme stages after publication parsing handle about *2000-3000* thousand data parts for CardiffNlp, but around *15 000* for SmartCampus. This naturally leads to higher values for the performance metrics in SmartCampus. While Real-time Latency was calculated by dividing execution time by number of publications, another more reflective metric could be to divide by the number of eventual data parts, which would better capture the true data load since it takes data-intensity into consideration.

7.1.5 Emergent Behaviour

Finally, an emergent and unexplained behaviour of the Neural Router which can be derived from the results is the following: From an execution time and latency standpoint, the proportion of programme-stage Publication Clustering decreases as the *token-limit-per-call* increases. The emergent aspect of this is that *token-limit-per-call* has no direct effect on the clustering process since clustering does not utilise this parameter at all.

This suggests that there are some indirect effects from the programme-stage publication parsing that affect the publication-clustering process. Investigating the origin of these indirect effects remains an open question for future research.

7.2 Neural Router: Future Directions

While the implementation and evaluation of the Neural Router has demonstrated the feasibility of *Structure-Agnostic and Configuration-Free content-delivery* to some degree, there is still room for a lot of improvement.

7.2.1 Proof-of-Concept vs. Production Ready

The current proof-of-concept implementation of the Neural Router focusses solely on the task of routing data and lacks other characteristics of Pub-Sub systems, such as *data-streaming*, *data-queues*, *consistent storage*. In future works a more production ready version of the Neural Router should be developed and integrated into the Agentic Pub-Sub Architecture. In this process, there also needs to be a clear defining of responsibilities between the two. There are also several quality requirements such as *security* that haven't been considered at all in this thesis and that need thorough contemplation.

In production the Neural Router would need to run in a completely asynchronous mode where both subscription-, publication-, and matching-logic can run in parallel with data entering and exiting the system freely. This allows for avoiding performance bottlenecks and reaching maximum efficiency.

An aspect of the Neural Router that has not been fully implemented is the initialisation process, where a system operator first launches the Neural Router. This user input would

consist of meta-data about the use-case data along with some example data. Depending on exactly how responsibilities are set up, the Neural Router or Overseer can use this user input to reason about the context and improve LLM prompts. For the proof-of-concept implementation, this has been simulated due to time constraints.

The Neural Router lacks several operationally important capabilities related to routing. For example, in the current implementation, publications and subscriptions are clustered after parsing. In a production environment where data are streamed into the system, clustering should occur periodically. In periods of non-clustering, publications and subscriptions need to be routed to existing clusters. Clustering is then activated according to some strategy, for example, time intervals, a number of new subscriptions/publications having entered the system, or once cluster cohesiveness falls under a certain threshold.

When considering data streaming and periodic clustering, the question of *advertising* also comes up. Advertisements are messages that publishers send to the router in a Pub-Sub system to let it know what kind of data they intend to produce. Based on advertisements, the router can create optimised routing strategies even before real data enters the system.

Another production-related aspect that needs consideration is the deployability of the Neural Router. For flexible and optimised deployment in diverse and dynamic environments, the foundational architecture would need to be a microservice-orientated architecture where different programme stages can be deployed separately.

7.2.2 Multi-modality

The Neural Router has thus far only been tested on textual data. However, the same logical framework could easily be expanded to enable the routing of images. This could be done by adding multimodal AI models for extracting image keywords. In case of images as subscriptions, tying keywords together is a Boolean expression. This would usually consist of AND, since the visual format seldom captures the NOT operator.

The challenges of multimodal content still persist in the form of ambiguity of the vocabulary and semantic gaps (Zhu et al., 2024). Large Language Models, and Vision Language Models can help alleviate such challenges.

7.2.3 Combating Hallucinations

To combat the challange of hallucination seen in the Neural Router techniques for comparing output with input was applied, but this creates inefficiencies. The Neural Router also utilises formatted JSON, which ensures that output adheres to a predefined JSON format using Python models. This process enforces a specific JSON format by pruning available tokens in the generation process. That is, if the format demands a "}", the LLM will not be able to generate anything but that token.

While OpenAI allows for using formatted JSON, SynCode (Ugare et al., 2024) allows for defining custom grammar which reduces syntax errors in generating Python and Go code with 96.07%. Using the same logic of forcing the LLM to adhere to a specific format or grammar, but algorithmically and dynamically editing the pruning filters during the LLM inference process could be a key to avoid hallucination problems. More precisely, between each token generation the output thus far could be passed to an algorithm that adjusts the grammar/pruning filter accordingly. Dynamic Token Pruning has already been tested showing improvements in both LLM Inference Time, and Memory Usage, while retaining accuracy (Keith et al., 2024).

In the case of the Neural Router this idea could be used in all programme stages where LLM inference is used. For example, in programme stage publication parsing we have publications with ids *1, 27, 35, 13*. First, the pruning filter could remove any other numbers that are present in the token options whenever generating a token for the field *publication_id*. Second, every time a publication has been parsed, the pruning algorithm could remove the publication from the pruning filter to make sure that it is not handled more than once.

Another, less sophisticated and more resource-intense method, which however could prove to remove almost all hallucinations with very high probability, would be to handle all data more than once and use the intersection as the final result. This is based on the assumption that there is no pattern as to how hallucination occurs, which according to the manual examination of the results, there is not. For example, if we have a 1% risk of producing a hallucination for some data with one handling of data. This would mean a 0,01% risk of intersection hallucination with two handlings and 0,0001% with three handlings. That is, the intersection would almost certainly contain *no hallucinations*. Now, because of the

large amount of data and only manual examination, it is not exactly certain how high the hallucination rate is, but the principle is the same. The down side of this technique is of course using *number of handlings* more LLM-related resources to achieve this, while never achieving 100% certainty due to the inherently probabilistic nature of LLMs.

7.2.4 Architectural Improvement

A potential enhancement to the core architecture of the Neural Router has been conceptualised, but not evaluated. This architectural improvement builds on switching the order of programme stages, Pub-Sub Matching, and Boolean Enforcing. While the current architecture narrows down the eventual search space for the LLM by clustering and optimisation, the eventual condition-data matches need to be enforced by the logical expressions extracted from subscriptions at the programme stage Subscription-Parsing. This means that a lot of potentially unnecessary matches are made by the LLM and then disposed of by a computationally much lighter technique, which is inefficient. The change to the architecture would enforce the Boolean expressions straight on the cluster matches produced by the programme stage Cluster-Matching. This would mean only presenting condition-data matches to the LLM in programme stage Pub-Sub matches that are actually able to create true matches avoiding the inefficiencies of finding condition-data matches that break down once *all* conditions enforced by the boolean expression.

While potentially more efficient, the above-described architectural change does not come without open questions. First, enforcing the boolean expressions on the condition-cluster to data-cluster matches would mean breaking down the clusters in some sense and would need an additional programme stage to achieve a sensible transition to the then final stage of Pub-Sub Matching. In interest of time, it was thus decided to leave this for future work.

7.3 Potential Use Cases for the Neural Router

The results of the proof-of-concept Neural Router highlight specific scenarios in which its application is appropriate, as well as cases where it may not be suitable. The evaluation of use-cases are considered both from a perspective of proved performance and from a perspective where the Neural Router has been improved, as well as in the context of the Agentic Pub-Sub Architecture.

7.3.1 Cloud-to-Edge deployment

Using gpt-4o or gpt-4o-mini across configurations and across datasets has a negligible effect on accuracy. This demonstrates that the Neural Router allows a less powerful model to achieve the same or better accuracy as a bigger, more powerful model. Although the efficiency metrics used in this thesis cannot capture the resource use of LLM inference, due to them running on proprietary cloud, a smaller model is both faster and less resource intensive than a bigger model.

The negligible effect of the LLM opens up the possibility to run the Neural Router locally on resource-constrained machines without relying completely on cloud services for LLM inference. For example, in an IoT network, smaller models could run on the edge of the network while still achieving the same accuracy as with big models that need to run on the cloud.

7.3.2 Indeterminism: A Challenge

Due to relying on probabilistic techniques the Neural Router does not achieve 100% accuracy, both when considering *recall* and *precision*. This is deciding for safety-critical use cases which thus could not rely simply on the Neural Router. For example, in the case of monitoring a patient in critical condition, a false negative could lead to the death of the patient, while false positives would mean that the nurse checks in on him even when nothing is wrong. That is, in a health monitoring system, False Positives would lead to *inefficiency*, but False Negatives could lead to *death*. In such a case, the currently achieved recall of the Neural Router would pose too great a risk to be used.

Conversely, False Positives can also be problematic in certain scenarios. For instance, in an automatic retaliatory weapons system, a False Positive detection of an enemy attack could have catastrophic consequences. A striking example occurred in 1983 when a Soviet early warning system mistakenly identified intercontinental ballistic missiles launched by the United States against the Soviet Union (Unal et al., 2022). Humanity narrowly avoided full-scale nuclear war thanks to a mid-level engineer in the Soviet Air Defence Forces defying protocol by waiting on additional confirmation and not immediately informing higher command about the perceived attack. Also in this case, the Neural Router would not be reliable enough to handle such critical scenarios.

While both suboptimal recall and precision can cause problems in safety critical situations, in most Pub-Sub systems, false negatives are harder to deal with for a number of reasons. First, it is harder to detect something *not happening* as opposed to detecting when something unintended *does happens*. For example, to return to the example of *nuclear threat detection system*. While a false positive almost had catastrophic consequences, it did raise the alarm to give chance of reaction. On the other hand, if a nuclear threat detection system had a false negative, it would completely defeat the purpose of said system.

Furthermore, it is also harder to recover errors that occur from something *not happening* than something *happening too much*. For example, a strategies for dealing with suboptimal precision are *deduplication* to remove duplicate messages and the analysis and logging of false positives in an effort to keep track of system consistency.

7.3.3 Intent-based: A Strength

While the Neural Router does not achieve perfect precision and recall, its intent-based nature makes it inherently well-suited for handling ambiguous or unstructured data. This characteristic is particularly valuable in scenarios where exact matches are difficult to define, but meaningful associations are still necessary.

For example, the Neural Router can be employed to match subscriber interests with various content sources, such as *news articles*, *real-estate listings*, and *job alerts*. By analysing user preferences and contextual information, it can dynamically adapt recommendations to ensure that users receive the most relevant and timely updates. This capability extends beyond simple keyword matching, leveraging intent recognition to provide a more nuanced and effective filtering system.

Additionally, the Neural Router can serve a crucial role in data monitoring and database management. It can continuously analyse the incoming data streams to identify relevant updates and integrate them into a document-based database. This ensures that stored information remains current and reflective of the latest developments. Whether applied to research repositories, legal databases, or market intelligence systems, the Neural Router could help automate data curation by intelligently routing new information to update the existing material.

7.3.4 Efficiency, Scalability, and Cost

The discussion around use-cases from an efficiency standpoint is very nuanced with several interlinked and overlapping concepts. First efficiency metrics unlike accuracy are continuous so it is much harder to pinpoint exact suited or unsuited use-cases. That is, while it is possible to argue that a patient-monitoring system cannot accept a single false negative, the appropriateness of efficiency is always a spectrum. When studying metrics from a use-case perspective, *Latency* affects usability for critical systems in real time. Another perspective is *cost*. Yet another perspective is *scalability* which in turn can be viewed both from a latency and cost perspective itself.

First, how powerful hardware is used affects general execution time and especially how many parallel LLM calls affect latency. If concurrent use is low and an LLM is run at the edge, it can be very efficient considering minimal network latency. If, however, the concurrent LLM inference calls stack up, the less powerful hardware available at the edge might prove a bottleneck for scalability. If using LLM inference from a model deployed in the cloud with virtually unlimited compute, it is possible to force down latency to a minimum, while network latency plays role. Currently, the Neural Router has only been tested with LLMs running in the cloud, achieving latencies in the *low hundreds of milliseconds*. This would not satisfy use-cases where a system needs to react rapidly in real time. However, in content-recommendation systems where real-time reactivity is not as crucial, better use could be expected. Again, when you take the context of the Agentic Pub-Sub Architecture into consideration, the focus might be a bit different since the Neural Router is only a temporary solution.

Second, the number of tokens used can be translated directly into a monetary cost. With the most balanced configuration of the Neural-Router *gpt-4o-mini-500*, the dollar-cost for handling a single publication-subscription match on CardiffNlp is around *0.0001€* and *0.0005€* on SmartCampus. While this is more expensive than traditional topic-based matching solutions, they are not comparable systems since they are static rule-based incapable of flexible handling of diverse data. Additionally, these prices are specific to the use of OpenAi's proprietary model. Using an open-source LLM running on dedicated hardware could dramatically reduce costs.

7.3.5 In the context of the Agentic Pub-Sub Architecture

Finally, the role of the Neural Router within the context of the Agentic Pub-Sub Architecture, also needs to be taken into consideration. First, this means that the Neural Router functions as a temporary and fall back router, while APSA can provide the determinism needed for both safety critical and general systems where the Neural Router might not be suited in its current form.

Additionally, scenarios where the immediate functionality of the Neural Router could outweigh small inaccuracies are situations where a small percent of incorrect routed data is preferable than no communication at all. To be more precise, such functionality could become very useful in different forms of *crises* where resources are scarce and existing systems might need to be repurposed quickly. The immediate functionality of the Neural Router could also be used for rapid prototyping where deterministic matching is not yet critical. In this context, APSA could start working on an optimised solution already during the development process.

Finally, the Structure-Agnostic and Configuration-free operation of the Neural Router and APSA could be use-full as a fall-back mechanism in an already existing Pub-Sub system. That is, even if the system already has its custom solution, the Neural Router could provide a routing option for when the custom solution might fail due to unexpected data. This would allow the system to continue to function instead of failing. Meanwhile, APSA would analyse the source of the mismatch and fix the error autonomously.

8 Conclusions

This thesis addressed key challenges of ever-increasing data volumes, data complexity, and the reliance on application-specific solutions to match subscriber interests with producer data. Putting emphasis on flexibility, this thesis explores the characteristics of an intelligent Pub-Sub system to handle these challenges. The proposed Agentic Pub-Sub Architecture outlines essential requirements and the foundational structures for an intelligent Pub-Sub system while the main focus is on implementing and evaluating APSA's core matching logic; the *Neural Router*, designed for *structure-agnostic and configuration-free content delivery*. On a grander scale this work contributes to intelligent systems and a paradigm shift towards intent-based systems.

8.1 Findings

This thesis was guided by the following three research questions:

RQ1: What features and requirements are presented in recent literature on natively intelligent Pub-Sub systems?

Addressed in Section 2.3, this research question was explored through an overview of recent literature on intelligence in the Pub-Sub paradigm. The findings indicate that recent research on native intelligence in Pub-Sub systems focusses on a variety of topics, ranging from high-level architectural overviews to specialised features such as *improved performance*, *personalized data presentation*, and *complex data handling*.

RQ2: How can the foundational structure of a natively intelligent Pub-Sub system be designed to achieve structure-agnostic and configuration-free content delivery while self-improving over time?

No single solution can handle everything with optimal performance. Hence, this thesis introduces the foundational structure of the Agentic Pub-Sub Architecture (APSA), which allows for immediate routing of any data while autonomous processes find an optimised solution over time.

RQ3: How can a data routing logic be implemented and evaluated to demonstrate the feasibility of structure-agnostic and configuration-free content delivery?

The core matching logic of APSA, the *Neural Router*, was implemented and evaluated using two datasets representing opposites on the structuredness spectrum to demonstrate its flexibility and feasibility for structure-agnostic, configuration-free content delivery. The evaluation showed strong performance on both unstructured and structured data, confirming its effectiveness in handling different data types without manual configuration.

8.2 Future Directions

During the design of the Agentic Pub-Sub Architecture and implementation and evaluation of the Neural Router, several insights and topics for further research were identified. First, the era of truly autonomous and intelligent systems seems to be rapidly approaching. For the Pub-Sub paradigm, intelligence can have several upsides, ranging from intent-driven matching to personalised outputs.

Future research should explore ways to enhance the intelligence of Pub-Sub architectures while balancing computational efficiency and mitigating risks related to indeterministic AI operation. Techniques that improve the reliability and accuracy of intelligent systems will be crucial in achieving this vision.

8.3 Final Words

Writing this thesis has been an invaluable learning experience. It has deepened my technical knowledge and sparked several new interests and ideas. It has also refined my skills in designing and evaluating systems and improved my scientific writing ability.

Furthermore, working as part of a larger team has provided valuable insight into how individual contributions fit into a broader context. This journey has reinforced two key lessons: *it is possible to learn a lot in a short time*, and even more crucial, *never be afraid to ask the questions you feel are important.*

Bibliography

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. (2022). "Flamingo: a visual language model for few-shot learning". In: *Advances in neural information processing systems*, 35, pp. 23716–23736.

Antypas, D., Ushio, A., Camacho-Collados, J., Neves, L., Silva, V., and Barbieri, F. (2022). "Twitter topic classification". In: *arXiv preprint arXiv:2209.09824*.

Ataei, P. and Litchfield, A. (2022). "The State of Big Data Reference Architectures: A Systematic Literature Review". In: *IEEE Access*, 10, pp. 113789–113807. DOI: 10.1109/ACCESS.2022.3217557.

Bharath, R. S., Suri, G., Dewangan, V., and Sonavane, R. (2025). *When Every Token Counts: Optimal Segmentation for Low-Resource Language Models*. arXiv: 2412.06926 [cs.CL]. URL: https://arxiv.org/abs/2412.06926.

Chafi, F. Z., Fakhri, Y., and Aadi, F. Z. A. H. (2020). "Introduction to Internet of Things' Communication Protocols". In: *International Conference on Advanced Intelligent Systems for Sustainable Development*. Springer, pp. 142–150.

Chapuis, B., Garbinato, B., and Mourot, L. (2017). "A Horizontally Scalable and Reliable Architecture for Location-Based Publish-Subscribe". In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 74–83. DOI: 10.1109/SRDS.2017.16.

Chiu, T. K. (2023). "The impact of Generative AI (GenAI) on practices, policies and research direction in education: A case of ChatGPT and Midjourney". In: *Interactive Learning Environments*, pp. 1–17.

Dhoni, P. (2023). "Exploring the synergy between generative AI, data and analytics in the modern age". In: *Authorea Preprints*.

Donta, P. K., Srirama, S. N., Amgoth, T., and Annavarapu, C. S. R. (2022). "Survey on recent advances in IoT application layer protocols and machine learning scope for research directions". In: *Digital Communications and Networks*, 8(5), pp. 727–744.

Eldeeb, E. (2023). *The Smart Campus Dataset*. DOI: 10.21227/xe4q-ax22. URL: https://dx.doi.org/10.21227/xe4q-ax22.

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). "The many faces of publish/subscribe". In: *ACM computing surveys (CSUR)*, 35(2), pp. 114–131.

Feng, Y., Ding, L., and Xiao, G. (2023). "Geoqamap-geographic question answering with maps leveraging LLM and open knowledge base (short paper)". In: *12th International Conference on Geographic Information Science (GIScience 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Franklin, S. and Graesser, A. (1999). "A software agent model of consciousness". In: *Consciousness and cognition*, 8(3), pp. 285–301.

Gheibi, O., Weyns, D., and Quin, F. (2021). "Applying machine learning in self-adaptive systems: A systematic literature review". In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 15(3), pp. 1–37.

Gong, T., Lyu, C., Zhang, S., Wang, Y., Zheng, M., Zhao, Q., Liu, K., Zhang, W., Luo, P., and Chen, K. (2023). "Multimodal-gpt: A vision and language model for dialogue with humans". In: *arXiv preprint arXiv:2305.04790*.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. (2025). "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning". In: *arXiv preprint arXiv:2501.12948*.

Huang, R., Li, M., Yang, D., Shi, J., Chang, X., Ye, Z., Wu, Y., Hong, Z., Huang, J., Liu, J., et al. (2024). "Audiogpt: Understanding and generating speech, music, sound, and talking head". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 21, pp. 23802–23804.

Isahagian, V., Muthusamy, V., and Slominski, A. (2023). "Publish-subscribe with large language models: Improving expressiveness with natural language and generated custom notifications". In: *Proceedings of the 24th International Middleware Conference: Demos, Posters and Doctoral Symposium*, pp. 29–30.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. (2024). "Openai o1 system card". In: *arXiv preprint arXiv:2412.16720*.

Jiang, P., Wen, C.-K., Yi, X., Li, X., Jin, S., and Zhang, J. (2024). "Semantic Communications Using Foundation Models: Design Approaches and Open Issues". In: *IEEE Wireless Communications*, 31(3), pp. 76–84.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361*.

Karpathy, A. (2023). *Intro to Large Language Models*. YouTube video. Accessed: 2025-03-01. URL: https://www.youtube.com/watch?v=%3CVIDEO_ID%3E.

Keith, C., Robinson, M., Duncan, F., Worthington, A., Wilson, J., and Harris, S. (2024). "Optimizing large language models: A novel approach through dynamic token pruning". In.

Lazidis, A., Tsakos, K., and Petrakis, E. G. (2022). "Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems". In: *Internet of Things*, 19, p. 100538.

Lee, Y., Kang, H. B., Latzke, M., Kim, J., Bragg, J., Chang, J. C., and Siangliulue, P. (2024). "PaperWeaver: Enriching Topical Paper Alerts by Contextualizing Recommended Papers with User-collected Papers". In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1–19.

Lei, B., Janssen, P., Stoter, J., and Biljecki, F. (2023). "Challenges of urban digital twins: A systematic review and a Delphi expert survey". In: *Automation in Construction*, 147, p. 104716.

Li (Jan. 2025). "OpenAI's 4o-mini Has Only 8B, Claude 3.5 Sonnet Reaches 175B". In: Accessed: 2025-02-27. URL: https://medium.com/ai-disruption/openais-4o-mini-has-only-8b-claude-3-5-sonnet-reaches-175b-9c1c55c53970.

Li, H. (2022). "Language models: past, present, and future". In: *Communications of the ACM*, 65(7), pp. 56–63.

Maniezzo, V., Boschetti, M. A., and Manzoni, P. (2022). "Self-adaptive Publish/Subscribe Network Design". In: *Metaheuristics International Conference*. Springer, pp. 478–484.

Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., et al. (2020). "Language models are few-shot learners". In: *arXiv preprint arXiv:2005.14165*, 1.

Milo, T., Zur, T., and Verbin, E. (2007). "Boosting topic-based publish-subscribe systems with dynamic clustering". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 749–760.

OpenAI (2025). *Latency Optimization Guide*. https://platform.openai.com/docs/guides/latency-optimization. Accessed: 2025-02-02.

OpenAI et al. (2024). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

OpenStreetMap (2025). *OpenStreetMap: Map view at latitude 65.45, longitude 26.07*. Accessed: 2025-01-20. URL: https://www.openstreetmap.org/#map=5/65.45/26.07.

Oracle (2024a). *Oracle Message Broker*. Last accessed: February 22, 2024. URL: https://docs.oracle.com/cd/E19879-01/821-0028/aercs/index.html.

Oracle (2024b). *Oracle Message Broker*. Accessed: January 16, 2025. URL: https://docs.oracle.com/cd/E19316-01/.

Packer, C., Fang, V., Patil, S., Lin, K., Wooders, S., and Gonzalez, J. (2023). "MemGPT: Towards LLMs as Operating Systems." In.

Qian, C., Han, C., Fung, Y. R., Qin, Y., Liu, Z., and Ji, H. (2023). "Creator: Tool creation for disentangling abstract and concrete reasoning of large language models". In: *arXiv preprint arXiv:2305.14318*.

Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., and Chalef, D. (2025). "Zep: A Temporal Knowledge Graph Architecture for Agent Memory". In: *arXiv preprint arXiv:2501.13956*.

Red Hat (2024). *Red Hat AMQ*. Last accessed: February 22, 2024. URL: https://access.redhat.com/documentation/en-us/red%20hat%20amq/6.1/html/product%20introduction/fmbscalable.

Saleh, A., Morabito, R., Tarkoma, S., Pirttikangas, S., and Lovén, L. (2024a). "Towards message brokers for generative ai: Survey, challenges, and opportunities". In: *arXiv preprint arXiv:2312.14647*.

Saleh, A., Tarkoma, S., Pirttikangas, S., and Loven, L. (2024b). "Publish-Subscribe for edge intelligence: Systematic review and future prospects". In: *Available at SSRN 4872730*.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. (2024). "Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face". In: *Advances in Neural Information Processing Systems*, 36.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2024). "Reflexion: Language agents with verbal reinforcement learning". In: *Advances in Neural Information Processing Systems*, 36.

Tarkoma, S. (2012). *Publish/subscribe systems: design and principles*. John Wiley & Sons.

Tarkoma, S., Morabito, R., and Sauvola, J. (2023). "AI-native interconnect framework for integration of large language model technologies in 6G systems". In: *arXiv preprint arXiv:2311.05842*.

Ugare, S., Suresh, T., Kang, H., Misailovic, S., and Singh, G. (2024). "SynCode: LLM generation with grammar augmentation". In: *arXiv preprint arXiv:2403.01632*.

Unal, B., Lewis, P., and Aghlani, S. (Mar. 2022). *Uncertainty and Complexity in Nuclear Decision-Making*. Chatham House. URL: https://www.chathamhouse.org/sites/default/files/2022-03/2022-03-07-nuclear-decision-making-unal-et-al_0.pdf.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, 30.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., et al. (2024). "A survey on large language model based autonomous agents". In: *Frontiers of Computer Science*, 18(6), p. 186345.

Wang, Y.-C., Xue, J., Wei, C., and Kuo, C.-C. J. (2023). "An overview on generative AI at scale with Edge-Cloud Computing". In: *IEEE Open Journal of the Communications Society*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in neural information processing systems*, 35, pp. 24824–24837.

Weil, C., Bibri, S. E., Longchamp, R., Golay, F., and Alahi, A. (2023). "Urban digital twin challenges: A systematic review and perspectives for sustainable smart cities". In: *Sustainable Cities and Society*, 99, p. 104862.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. (2023). "The rise and potential of large language model based agents: A survey". In: *arXiv preprint arXiv:2309.07864*.

Xie, Z. (2024). "Order Matters in Hallucination: Reasoning Order as Benchmark and Reflexive Prompting for Large-Language-Models". In: *arXiv preprint arXiv:2408.05093*.

Yan, L., Martinez-Maldonado, R., and Gasevic, D. (2024). "Generative Artificial Intelligence in Learning Analytics: Contextualising Opportunities and Challenges through the Learning Analytics Cycle". In: *Proceedings of the 14th Learning Analytics and Knowledge Conference*. LAK '24. Kyoto, Japan: Association for Computing Machinery, pp. 101–111. ISBN: 9798400716188. DOI: 10.1145/3636555.3636856. URL: https://doi-org.libproxy.helsinki.fi/10.1145/3636555.3636856.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2022). "React: Synergizing reasoning and acting in language models". In: *arXiv preprint arXiv:2210.03629*.

Zaarour, T., Bhattacharya, A., and Curry, E. (2022). "OpenPubSub: supporting large semantic content spaces in peer-to-peer publish/subscribe systems for the internet of multimedia things". In: *IEEE internet of things journal*, 9(18), pp. 17640–17659.

Zeeshan, T., Kumar, A., Pirttikangas, S., and Tarkoma, S. (2025). "Large Language Model Based Multi-Agent System Augmented Complex Event Processing Pipeline for Internet of Multimedia Things". In: *arXiv preprint arXiv:2501.00906*.

Zhang, Y., He, Z., Li, J., Lin, J., Guan, Q., and Yu, W. (2024). "MapGPT: An autonomous framework for mapping by integrating large language model and cartographic tools". In: *Cartography and Geographic Information Science*, 51(6), pp. 717–743.

Zhu, H., Huang, J.-H., Rudinac, S., and Kanoulas, E. (2024). "Enhancing Interactive Image Retrieval With Query Rewriting Using Large Language Models and Vision Language Models". In: *Proceedings of the 2024 International Conference on Multimedia Retrieval*. ICMR '24. Phuket, Thailand: Association for Computing Machinery, pp. 978–987. ISBN: 9798400706196. DOI: 10.1145/3652583.3658032. URL: https://doi.org/10.1145/3652583.3658032.

The Use of Artificial Intelligence in the Thesis

This thesis has used OpenAI's ChatGpt for a number of tasks. 1. brainstorming, 2. organisation of notes, 3. generation of boilerplate code, 4. finding the best word and simple sentence rephrasings. Writefull was used to correct grammar. AI has not been used as the basis for any claim or writing of text or anything else.

Appendix A: CardiffNlp Subscriptions

In Table 8.1 all CardiffNlp subscriptions along with their original topic-labls can be found.

Table 8.1: Topic Labels and Subscriptions

Topic-label	Subscriptions
arts_&_culture	art and culture, art, culture, culture and art, cultural art
entrepreneurship	entrepreneurship, entrepreneurial, entrepreneurism, entrepreneurialism, entrepreneurs
celebrity_&_pop_ culture	celebrity and pop culture, celebrity, pop culture, pop culture and celebrity, celebrity culture
diaries_&_daily_life	diaries and daily life, diaries, daily life, daily life and diaries, daily diaries
family	family, family life, family matters, family things, family theme
fashion_&_style	fashion and style, style and fashion, fashion topics, fashionable, fashion aspects
film_tv_&_video	film tv and video, film and television, film and video, video and tv, tv and film
fitness_and_health	fitness and health, fitness, health, health and fitness, fitness and wellness
food_&_dining	food and dining, food, dining, culinary, eating
gaming	gaming, video gaming, computer gaming, video games, computer games
learning_&_ educational	learning and education, learning, education, educating, education and learning
music	music, musician, music topics, music related, music aspects
news_&_social_ concern	news and social concern, news updates, news and issues, news feed, news matters

Topic-label	Subscriptions
other_hobbies	other hobbies, hobby themes, general hobbies, hobby topics, hobby things
relationships	relationships, relationship, relations, relationship theme, relationship topic
science_&_technology	science and technology, science, science and tech, technology, technology and science
sports	sports, sports related, sport, sport related, sport topics
travel_&_adventure	travel and adventure, travel, adventure, travel topics, travel focus
youth_&_student_life	youth and student life, youth, student, student and youth life, student life and youth

Appendix B: Exhaustive Experimental Results

In Table 8.2 the exhaustive results on CardiffNlp can be seen. In Table 8.3 the same for SmartCampus is found.

Table 8.2: CardiffNlp: Performance Averages for all Program Stages and all Configurations

Program Stage	Configuration	Execution Time	Latency	Throughput	LLM Inference Time	Total Tokens	Input Tokens	Data Tokens	Instruction Tokens	Output Tokens
Bool Enforcing	gpt-4o-100	0.019	0.0	53097.36	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-1000	0.003	0.0	368801.292	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-500	0.005	0.0	205937.759	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-mini-100	0.02	0.0	50662.183	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-mini-1000	0.003	0.0	373768.451	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-mini-500	0.004	0.0	233964.373	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-100	0.014	0.0	75784.188	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-1000	0.017	0.0	59959.25	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-500	0.018	0.0	57741.728	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-mini-100	0.015	0.0	73933.679	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-mini-1000	0.017	0.0	59764.862	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-mini-500	0.017	0.0	57209.414	N/A	N/A	N/A	N/A	N/A	N/A
Pub Sub Matching	gpt-4o-100	19.655	0.02	52.722	5153.027	652187.0	571481.0	51715.8	519765.2	80706.0
Pub Sub Matching	gpt-4o-1000	23.618	0.023	42.6	495.073	100172.8	71224.2	26607.2	44617.0	28948.6
Pub Sub Matching	gpt-4o-500	19.15	0.019	52.312	774.403	141975.2	105342.0	30906.2	74435.8	36633.2
Pub Sub Matching	gpt-4o-mini-100	20.316	0.02	51.514	6758.402	808052.2	707317.2	63147.8	644169.4	100735.0
Pub Sub Matching	gpt-4o-mini-1000	26.138	0.026	40.63	441.815	90708.6	64721.2	23156.4	41564.8	25987.4
Pub Sub Matching	gpt-4o-mini-500	25.996	0.026	45.031	726.349	134689.2	100059.2	29369.4	70689.8	34630.0
Publication Clustering	gpt-4o-100	117.724	0.118	8.503	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-1000	65.32	0.065	15.409	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-500	73.323	0.073	13.645	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-mini-100	117.291	0.117	8.526	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-mini-1000	57.971	0.058	17.27	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-mini-500	70.996	0.071	14.093	N/A	N/A	N/A	N/A	N/A	N/A
Publication Optimization	gpt-4o-100	15.583	0.016	65.379	2194.112	295260.4	236116.2	83502.0	152614.2	59144.2
Publication Optimization	gpt-4o-1000	29.29	0.029	35.351	618.553	222572.8	184030.4	135674.4	48356.0	38542.4
Publication Optimization	gpt-4o-500	20.984	0.021	48.516	643.235	170031.0	135618.0	85799.0	49819.0	34413.0
Publication Optimization	gpt-4o-mini-100	14.036	0.014	73.133	2700.765	357041.6	285145.6	101423.6	183722.0	71896.0
Publication Optimization	gpt-4o-mini-1000	25.979	0.026	39.163	384.811	158714.4	133623.4	97279.4	36344.0	25091.0
Publication Optimization	gpt-4o-mini-500	24.864	0.025	43.357	648.3	161174.4	128421.6	80989.6	47432.0	32752.8
Publication Parsing	gpt-4o-100	23.218	0.023	45.868	9788.169	714385.8	585709.0	89709.0	496000.0	128676.8
Publication Parsing	gpt-4o-1000	32.686	0.033	31.884	1715.956	228504.4	138832.2	90819.4	48012.8	89672.2
Publication Parsing	gpt-4o-500	24.952	0.025	41.377	2143.885	287831.6	188590.2	89787.0	98803.2	99241.4
Publication Parsing	gpt-4o-mini-100	24.149	0.024	42.443	9643.372	714881.4	585709.0	89709.0	496000.0	129172.4
Publication Parsing	gpt-4o-mini-1000	27.55	0.028	37.398	1501.901	228398.0	138805.2	90693.2	48112.0	89592.8
Publication Parsing	gpt-4o-mini-500	32.176	0.032	32.344	2471.044	288022.2	188852.6	89851.0	99001.6	99169.6
Subscription Clustering	gpt-4o-100	6.407	0.056	17.806	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-1000	6.292	0.055	18.178	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-500	6.644	0.058	17.521	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-mini-100	6.739	0.059	16.977	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-mini-1000	5.462	0.048	20.892	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-mini-500	6.822	0.06	17.013	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Optimization	gpt-4o-100	5.29	0.046	29.779	49.899	20249.4	18941.2	2565.2	16376.0	1308.2
Subscription Optimization	gpt-4o-1000	2.679	0.023	45.797	34.425	17862.0	16593.2	2517.2	14076.0	1268.8
Subscription Optimization	gpt-4o-500	3.712	0.033	42.808	34.258	17792.6	16537.8	2553.8	13984.0	1254.8
Subscription Optimization	gpt-4o-mini-100	4.224	0.037	35.712	49.777	20126.0	18826.6	2542.6	16284.0	1299.4
Subscription Optimization	gpt-4o-mini-1000	2.043	0.018	59.994	29.311	17486.6	16339.6	2355.6	13984.0	1147.0
Subscription Optimization	gpt-4o-mini-500	3.142	0.028	40.922	40.828	17923.6	16645.4	2569.4	14076.0	1278.2
Subscription Parsing	gpt-4o-100	10.107	0.089	11.663	107.763	16546.2	11097.0	1665.0	9432.0	5449.2
Subscription Parsing	gpt-4o-1000	52.722	0.462	2.226	86.045	9712.0	4603.4	2821.8	1781.6	5108.6
Subscription Parsing	gpt-4o-500	34.986	0.307	3.366	103.232	9342.6	4125.6	1820.0	2305.6	5217.0
Subscription Parsing	gpt-4o-mini-100	13.174	0.116	9.252	126.933	16415.8	11097.0	1665.0	9432.0	5318.8
Subscription Parsing	gpt-4o-mini-1000	43.809	0.384	2.696	68.053	8163.8	2992.0	1839.2	1152.8	5171.8
Subscription Parsing	gpt-4o-mini-500	24.937	0.219	5.311	67.799	9266.2	4138.8	1833.2	2305.6	5127.4

Table 8.3: SmartCampus: Performance Averages for all Program Stages and all Configurations

Program Stage	Configuration	Execution Time	Latency	Throughput	LLM Inference Time	Total Tokens	Input Tokens	Data Tokens	Instruction Tokens	Output Tokens
Bool Enforcing	gpt-4o-100	0.037	0.0	27570.762	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-1000	0.006	0.0	171739.379	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-500	0.011	0.0	94016.609	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-mini-100	0.037	0.0	27166.955	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-mini-1000	0.006	0.0	170001.182	N/A	N/A	N/A	N/A	N/A	N/A
Bool Enforcing	gpt-4o-mini-500	0.011	0.0	94407.343	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-100	0.008	0.0	127051.877	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-1000	0.009	0.0	106789.429	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-500	0.009	0.0	112746.942	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-mini-100	0.008	0.0	122432.276	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-mini-1000	0.009	0.0	108340.461	N/A	N/A	N/A	N/A	N/A	N/A
Cluster Matching	gpt-4o-mini-500	0.009	0.0	108615.4	N/A	N/A	N/A	N/A	N/A	N/A
Pub Sub Matching	gpt-4o-100	336.503	0.336	3.023	407701.123	1871957.6	1505690.4	122044.2	1383646.2	366267.2
Pub Sub Matching	gpt-4o-1000	69.083	0.069	15.011	2225.683	344718.8	223485.8	96910.2	126575.6	121233.0
Pub Sub Matching	gpt-4o-500	82.041	0.082	12.452	4073.241	502199.4	335551.6	100147.8	235403.8	166647.8
Pub Sub Matching	gpt-4o-mini-100	301.436	0.302	3.332	408346.403	1892918.4	1512275.8	121760.6	1390515.2	380642.6
Pub Sub Matching	gpt-4o-mini-1000	100.35	0.1	10.207	2646.246	341210.0	221440.0	93807.4	127632.6	119770.0
Pub Sub Matching	gpt-4o-mini-500	70.763	0.071	14.46	3797.195	510976.2	343261.0	102394.2	240866.8	167715.2
Publication Clustering	gpt-4o-100	189.898	0.19	5.274	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-1000	184.421	0.184	5.425	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-500	180.271	0.18	5.547	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-mini-100	184.914	0.185	5.408	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-mini-1000	182.055	0.182	5.496	N/A	N/A	N/A	N/A	N/A	N/A
Publication Clustering	gpt-4o-mini-500	177.813	0.178	5.625	N/A	N/A	N/A	N/A	N/A	N/A
Publication Optimization	gpt-4o-100	19.948	0.02	50.498	8955.736	762378.2	628508.0	191456.0	437052.0	133870.2
Publication Optimization	gpt-4o-1000	44.788	0.045	24.527	1625.079	363812.8	265481.6	199089.6	66392.0	98331.2
Publication Optimization	gpt-4o-500	29.195	0.029	35.519	2929.736	487648.0	358501.0	232683.0	125818.0	129147.0
Publication Optimization	gpt-4o-mini-100	19.22	0.019	53.435	8591.137	763810.2	629981.4	191983.4	437998.0	133828.8
Publication Optimization	gpt-4o-mini-1000	56.471	0.057	17.921	2156.037	364044.2	266162.6	198652.6	67510.0	97881.6
Publication Optimization	gpt-4o-mini-500	33.246	0.033	31.499	2763.457	408918.0	299618.8	195730.8	103888.0	109299.2
Publication Parsing	gpt-4o-100	31.943	0.032	31.507	14903.075	1272066.0	955816.0	149010.0	806806.0	316250.0
Publication Parsing	gpt-4o-1000	69.112	0.069	14.649	5919.49	591815.4	281464.0	149441.2	132022.8	310351.4
Publication Parsing	gpt-4o-500	43.437	0.043	23.374	7241.77	750388.6	438584.4	149069.2	289515.2	311804.2
Publication Parsing	gpt-4o-mini-100	37.457	0.038	27.484	17017.825	1272270.6	955816.0	149010.0	806806.0	316454.6
Publication Parsing	gpt-4o-mini-1000	87.143	0.087	11.96	6983.655	592756.2	282722.8	149894.0	132828.8	310033.4
Publication Parsing	gpt-4o-mini-500	62.18	0.062	16.594	8252.851	751009.8	438807.6	149131.2	289676.4	312202.2
Subscription Clustering	gpt-4o-100	5.753	0.09	11.135	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-1000	5.964	0.093	10.776	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-500	5.848	0.091	10.948	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-mini-100	6.275	0.098	10.34	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-mini-1000	5.413	0.085	11.828	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Clustering	gpt-4o-mini-500	6.329	0.099	10.432	N/A	N/A	N/A	N/A	N/A	N/A
Subscription Optimization	gpt-4o-100	4.384	0.069	17.374	57.754	36032.6	34273.4	2682.0	31591.4	1759.2
Subscription Optimization	gpt-4o-1000	5.454	0.085	13.008	29.154	21055.0	19872.8	2671.2	17201.6	1182.2
Subscription Optimization	gpt-4o-500	4.09	0.064	16.342	34.774	21674.0	20383.2	2685.4	17697.8	1290.8
Subscription Optimization	gpt-4o-mini-100	5.119	0.08	14.776	61.981	34843.2	33106.0	2672.4	30433.6	1737.2
Subscription Optimization	gpt-4o-mini-1000	3.938	0.062	16.848	24.769	19884.4	18709.6	2665.8	16043.8	1174.8
Subscription Optimization	gpt-4o-mini-500	3.012	0.047	22.609	28.86	23357.2	21984.4	2798.0	19186.4	1372.8
Subscription Parsing	gpt-4o-100	7.74	0.121	8.485	117.679	25162.6	19098.0	1986.0	17112.0	6064.6
Subscription Parsing	gpt-4o-1000	69.733	1.09	0.95	129.0	16625.4	10668.6	5758.2	4910.4	5956.8
Subscription Parsing	gpt-4o-500	30.537	0.477	2.292	109.026	11832.0	5910.6	2041.8	3868.8	5921.4
Subscription Parsing	gpt-4o-mini-100	10.631	0.166	6.457	144.948	25161.0	19098.0	1986.0	17112.0	6063.0
Subscription Parsing	gpt-4o-mini-1000	56.822	0.888	1.153	98.542	14230.2	8256.2	4387.4	3868.8	5974.0
Subscription Parsing	gpt-4o-mini-500	23.964	0.374	2.725	82.345	11611.2	5688.0	1968.0	3720.0	5923.2