

How good are LLM-based AI agents at operating a network?

Semester Thesis

Author: Viola Zucchi

Tutor: Pietro Ronchetti

Supervisor: Prof. Dr. Laurent Vanbever

March 2025 to July 2025

Abstract

This thesis investigates the feasibility of using Large Language Model (LLM)-based AI agents for diagnosing network issues in a live environment. Specifically, it evaluates whether commercially available models, operating in a zero-shot context and without domain-specific training, can accurately detect and reason about BGP misconfigurations. A controlled virtual network was developed using the Mini-Internet API, and various error scenarios were introduced to test the agents' diagnostic capabilities. Several models, specifically Gemini, GPT-4o, and o4-mini, were evaluated for performance, token usage, and cost. Results show that while more powerful reasoning models (like o4-mini) outperform others, they are also significantly more resource-intensive and still struggle with reliability and domain-specific nuance. This work highlights both the potential and the limitations of applying current LLM technology to autonomous network troubleshooting.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Task and Goals	1
1.3	Overview	2
2	Background and Related Work	3
2.1	Background	3
2.1.1	LLMs	3
2.1.2	AI Agents	4
2.2	Related Work	6
3	Design	8
3.1	Live Network	8
3.2	Agent Design	9
3.2.1	Base Structure	10
3.2.2	Tools	10
3.2.3	LLM Choice	12
3.2.4	System Prompt	13
3.3	Test Setup	13
4	Evaluation	15
4.1	Experiment design	15
4.1.1	Unwelcomed Access list	15
4.1.2	Invert Route-map	16
4.1.3	Remove lo Interface	16
4.1.4	Violate BGP Policies	17
4.1.5	No Next-Hop-Self	18
4.1.6	No Null0	19
4.2	Results	19
4.2.1	Gemini	20
4.2.2	GPT-4o	21
4.2.3	o4-mini	21
4.2.4	Token Usage	24
4.2.5	Expenses	25
4.3	Discussion	27

<i>CONTENTS</i>	iii
5 Outlook	29
5.1 Improvements to my work	29
5.2 Future Work	29
6 Summary	30
References	31
A System Prompt	I
B Hallucinatory responses	IV

Chapter 1

Introduction

In this chapter I am going to introduce the motivation for this project, list the associated tasks and state the end goal. At the end I am going to provide an overview of the report.

1.1 Motivation

Root cause analysis (RCA) refers to the process of identifying the underlying cause of a malfunction. In communication networks, diagnosing issues requires deep expertise and the ability to reason across diverse and complex systems. Traditionally, this task has been performed by human operators due to its high cognitive demands. However, the process is often time-consuming, with significant effort spent collecting data from multiple devices, logs, and configuration files. For example, diagnosing a connection loss may involve checking physical cabling, interface states, switch configurations, routing tables, and firewall rules.

Recent advancements in artificial intelligence (AI) have enabled automation of tasks that were once exclusively performed by humans. For instance, AutoGPT’s [20] capabilities demonstrated the feasibility of autonomous, multi-step LLM-driven agents. These developments motivate the application of AI to RCA in network systems. There are several reasons to consider AI a strong candidate for this task. First, data collection and interaction with network elements are largely text-based, aligning well with the capabilities of LLMs. Second, LLMs are particularly adept at handling diverse and heterogeneous inputs—a defining feature of RCA in communication networks. However, they are also known to generate inaccurate or entirely incorrect responses, a phenomenon known as hallucination.

To mitigate this, recent advancements have focused on agent-based workflows, which differ from traditional single-step AI interactions. These workflows involve iterative reasoning and structured decision-making, where an LLM-based agent interacts with its environment through defined interfaces. This allows the agent to request additional data, form hypotheses, and validate or revise its assumptions based on system feedback. In essence, agent-based methods introduce a feedback loop that makes AI behavior more adaptive and grounded in the context of the live system.

1.2 Task and Goals

The objective of this project is to assess the capacity of AI agents to diagnose network-related issues without any prior domain-specific training. Specifically, it investigates whether commercially available AI models, when used in a zero-shot learning context, can reason effectively about network behavior and protocol dynamics. Given the vast number of potential causes for such issues in a

network, the scope of the project is deliberately narrowed: it focuses exclusively on the Border Gateway Protocol (BGP), and only considers mistakes its configuration within its setup.

To carry out this investigation, the work was divided into a series of core tasks. These included building a suitable test environment, designing a custom AI agent, enabling interactions between the agent and the network, and evaluating its performance. The tasks are detailed as follows:

1. **Building a virtual network environment.**

I set up a virtual testbed that could emulate the behavior of the Internet on a smaller scale. Based on the mini-Internet project API [10], it uses Docker containers to simulate multiple Autonomous Systems (ASs) allowing the formation of BGP sessions between them.

2. **Introducing controlled network faults.**

To simulate realistic network problems, I introduced various BGP-related misconfigurations. These errors were selected for their ability to produce consistent and observable disruptions, such as connection losses or routing anomalies, while remaining repeatable for testing purposes.

3. **Designing an autonomous AI agent architecture.**

I explored modern approaches to agent design and adopted a ReAct-style architecture, customizing it to suit the requirements of this project. The agent structure was chosen for its ability to support iterative reasoning and dynamic tool use during problem-solving.

4. **Enabling communication between the AI agent and the network.**

To allow the AI agent to interact with the live system, I implemented a suite of tools that acted as an interface between the model and the network environment. These tools allowed the agent to retrieve information from the routers' configuration interface, namely the FRRouting command line, a looking glass service, a BGP troubleshooting manual, and other relevant sources.

5. **Establishing performance and evaluation metrics.**

To assess the agent's effectiveness, I defined a set of metrics capturing various aspects of performance. These included the number of steps taken to reach a diagnosis, token usage, execution time, success rate, and overall API cost. I then tested the agent across different scenarios and collected the corresponding data to analyze its behavior and efficiency.

1.3 Overview

This report is structured to evaluate the diagnostic capabilities of LLM-based AI agents in the context of network operations.

Chapter 1 introduces the motivation—automating root cause analysis in communication networks using AI agents—and outlines the project's scope and objectives.

Chapter 2 provides essential background on LLMs, AI agents, and related research in AI-driven networking.

Chapter 3 details the design of the live network, the custom AI agent with its features, and the evaluation setup.

Chapter 4 presents the test scenarios, results for each tested model, and a discussion on performance, cost, and limitations.

Chapter 5 proposes further improvements and future work.

Chapter 2

Background and Related Work

This chapter provides the foundational background in Artificial Intelligence necessary to contextualize the work presented in this report. It introduces key concepts related to Large Language Models and AI agents, covering all aspects relevant to the scope of this study. Additionally, it reviews related work in the field of communication networks, the application of AI agents to automate tasks traditionally performed by humans, and the intersection of these domains.

2.1 Background

2.1.1 LLMs

Large Language Models are AI systems trained on an enormous corpus of text, enabling them to extract and infer semantic meaning. This allows these models not only to perform a wide range of natural language processing tasks, but also to generalize to new tasks and domains with minimal or no task-specific training [14]. This ability is known as Few or Zero-shot learning and it is especially interesting because it allows to automate tasks without the need for additional training. Language models are able to process language and instructions through tokenization. This technique produces the units through which computational power, and subsequently monetary cost, is calculated. In the next subsection I am going to provide more in depth background knowledge on the use of LLMs as zero-shot learners and the Tokenization process.

Zero-shot learners

LLMs are capable of zero-shot learning, few-shot learning, and in some cases can be fine-tuning for specific tasks. In zero-shot and few-shot learning, the model is prompted to perform tasks it has not specifically encountered during training, relying on its general reasoning abilities acquired from extensive pretraining. Few-shot learning involves providing one or more examples within the prompt, whereas zero-shot learning includes no examples at all. In contrast, fine-tuning involves updating the model's internal parameters through additional training on task-specific data. While effective, this process is significantly more time-consuming and computationally intensive. In this report, I focus on the use of LLMs as zero-shot learners to evaluate their ability to reason about network-related issues without requiring further training. [11]

Tokens

Tokenization is the process of breaking data into chunks so that it can be processed by LLMs. There exist many methods of tokenization for different types of data, for different purposes and for different models. I am going to focus on textual data as this is the way information gets extracted from the network.

Input text gets broken down into small pieces, typically around four characters, and separation does not necessarily aligns with the start or the end of a word. Each piece, or token, gets assigned a number based on the frequency of the token and of the context of the input. [19] The way in which LLMs generate responses is by predicting one token at a time; the way it gets picked is based on which one is the most likely considering the input and the output produced so far. This process is called inference. To aid inference, LLMs use an attention mechanism. This means assigning soft weights to the input tokens in order to direct the model attention, highlight relationships within the data, and capture context. [22] This is what allows models to produce coherent text.

It's important to know there are limits to how many tokens a machine can process at once. Its maximum capacity is referred to as context window and its size can vary depending on the model being used. Tokens in the context can be of two types:

- **Prompt tokens**

Prompt tokens are all tokens that make up the input to an LLM.

- **Completion tokens**

Completion tokens are all tokens generated to answer a query. Reasoning models are capable of performing structured, multi-step thinking to solve problems or make decisions; to do so they spend extra tokens, also known as reasoning tokens. These latter aren't part of the output text, but are used by the model to produce better replies to more complex questions.

Tokens are particularly relevant because they capture LLMs performance and cost metrics. Each token requires a certain amount of power to process or generate, thus AI services are charged based on their consumption. Generally, pricing plans set a maximum number of available tokens, that can be shared between input and output. Furthermore, to support multiple concurrent users, there are typically limits for token generation per minute. [15]

Note that different language models offer varying strengths in reasoning abilities, computational demands, and supported data modalities. Some models are optimized for complex reasoning but require greater computational resources, while others support a broader range of input types beyond plain text. Selecting the appropriate LLM for a given application involves balancing performance and price point.

2.1.2 AI Agents

An AI agent is a system that utilizes a Large Language Model (LLM) to make decisions and manage the control flow of an application. This enables the agent to autonomously interpret inputs, select appropriate actions, and coordinate tasks. The degree of autonomy and control an agent possesses is largely determined by its underlying architecture. Agents can be as simple as routers; an application can continue along two or more potential logic path and the agent is only responsible of choosing which one. In this work, I adopt the structure introduced by ReAct agents, which integrates both *reasoning* and *acting* capabilities. [23] These agents operate by receiving an initial prompt and then initiating a loop of task-specific actions, each autonomously selected by the model. Throughout this process, reasoning phases are interleaved, during which the model assesses the outcomes of previously executed tasks and determines whether the original request has been

adequately addressed. This iterative cycle of acting and reasoning enables the agent to adaptively and incrementally progress toward completing complex objectives. Integral part of this particular agent are the following features:

- **Tools**

To perform actions, an agent relies on tools—predefined function calls that enable the model to interact with external systems. Tools are particularly important when actions require structured inputs, as their interfaces provide the necessary parameters and constraints. This allows the model to generate function calls in a format that aligns with the application’s requirements, ensuring reliable and context-appropriate execution.

- **Memory**

To effectively utilize information gathered during operation, agents are equipped with memory mechanisms. These mechanisms can be implemented in various customizable ways. A common and straightforward approach involves maintaining an accumulated context—which may include the initial prompt, intermediate reasoning steps, tool outputs, and any retrieved documents—and supplying it to the LLM as part of the prompt in subsequent reasoning phases. This allows the agent to maintain continuity and coherence across multiple steps in the decision-making process.

- **Planning capabilities**

The LLM operates within a control loop in which it autonomously determines the flow of execution. At each step, it decides whether to invoke additional tools or to conclude the process by generating a final response to the prompt, thereby terminating the loop. This dynamic control flow allows the agent to adapt its behavior based on the evolving context.

Another relevant element that is part of the agent make up is the *system prompt*. This is a predefined set of instructions provided at the start of each session. This prompt establishes the agent’s role, influences its response style, and can include essential background information that the model references throughout the interaction without needing to be repeated in every user prompt.

When building an agent, the main aspects of customization are: the choice of LLM, to fit the applications need; the selection of tools, to define its action’s capabilities and limits; and lastly the system prompt, to guide the model to fulfill its role.

Prompt Caching

A potential challenge in using such agents is the rapidly increasing token consumption associated with repeatedly feeding accumulated context into subsequent LLM prompts. To mitigate this, LLMs employ a technique known as *prompt caching*. This involves maintaining a cache of recently processed prompt segments and comparing them against the initial portion of new requests. When a match is found, the model can reuse pre-computed attention states, thereby reducing both latency and computational cost. Notably, some providers, such as OpenAI, offer discounted pricing for cached tokens, further incentivizing the use of this optimization strategy. [6], [18]

Model Entropy

LLMs are inherently non-deterministic because they rely on probabilistic methods, which means they can generate different outputs for the same prompt across multiple executions. This variability can be beneficial when creativity or diverse responses are desired. However, it poses challenges

during testing and evaluation, where consistent and reproducible behavior is often required. Fortunately, many models provide parameters that allow control over the randomness, effectively adjusting the *entropy* or uncertainty in their predictions to achieve more deterministic outputs when needed.

Two key parameters regulate the entropy of LLM outputs:

- **Temperature:** This parameter controls the randomness of the model’s output distribution. Lower values (e.g., close to 0) make the model more deterministic by increasing the likelihood of selecting high-probability tokens, while higher values produce more varied and creative outputs by flattening the distribution.
- **Top-p (nucleus sampling):** This parameter restricts the model’s token choices to the smallest subset whose cumulative probability exceeds a specified threshold p . For instance, a top-p value of 0.9 means the model samples only from tokens that collectively represent 90% of the probability mass, thereby reducing the chance of selecting unlikely or irrelevant tokens.

2.2 Related Work

This project builds on prior work in several areas, notably AI agent frameworks, network simulation environments, and recent efforts to benchmark and apply large language models (LLMs) to tasks traditionally handled by humans.

AI Agents and Human-like Tasks

Recent research has increasingly explored the ability of LLM-based agents to carry out tasks typically performed by humans. In particular, the authors of AutoGPT [20] introduced autonomous AI agents capable of goal-oriented planning and tool usage, while the authors of CAMEL [12] proposed a multi-agent framework for collaborative problem solving through role-based interaction. These systems demonstrate that LLMs can act with surprising autonomy and competence, even in unfamiliar domains. To support these capabilities, this project makes extensive use of the LangChain framework [2], a powerful toolkit designed for developing language model-driven applications. LangChain provides abstractions for managing prompts, memory, tool usage, and multi-agent coordination, making it ideal for constructing the complex workflows required for autonomous troubleshooting agents.

Mini-Internet for Network Simulation

The Mini-Internet API [10] is a lightweight network emulation framework designed to simulate small-scale internet environments for testing and experimentation. It allows researchers and developers to configure virtual hosts, routers, and links with customizable parameters, enabling the evaluation of network behaviors in controlled settings. In this project, the Mini-Internet API was used to set up a realistic network environment, providing the infrastructure necessary for this purpose.

Model Benchmarking

A number of standardized benchmarks have been developed to evaluate the diverse capabilities of large language models (LLMs), providing a foundation for comparative analysis across different architectures and configurations. These benchmarks assess models across a range of tasks, from factual recall and reasoning to commonsense understanding and mathematical problem solving.

- **MMLU** [9]: Tests factual knowledge and reasoning across 57 academic and professional subjects at an undergraduate level.
- **BIG-bench** [21]: A diverse suite of over 200 tasks evaluating reasoning, general knowledge, creativity, and emergent abilities.
- **HELLASWAG** [24]: Assesses commonsense reasoning by selecting the most plausible sentence continuations.
- **ARC** [3]: Grade-school science questions requiring logical inference and domain understanding.
- **GSM8K** [4]: Math word problems targeting arithmetic and multi-step reasoning at a grade-school level.
- **TruthfulQA** [13]: Measures a model’s ability to avoid generating false or misleading answers.

These benchmarks have become foundational in LLM evaluation and are widely cited in the literature to demonstrate strengths and limitations across domains. They also guide model development and selection in downstream applications requiring trustworthy reasoning, domain expertise, or interpretability.

Positioning This Work

Unlike prior work, this study focuses specifically on the feasibility of applying general-purpose LLM agents to communication network troubleshooting without any domain-specific pretraining. The novelty lies in testing whether a reasoning-capable LLM, when provided with external tools and structured workflows, can compensate for its lack of inherent networking knowledge and still perform meaningful diagnostics.

Chapter 3

Design

My system consists of three main components, as illustrated in Figure 3.1. Component ① is the *live network*, a virtual environment that simulates a miniature version of the Internet. Component ② is the *AI agent*, equipped with all necessary elements to interact with the network and enable testing of its reasoning capabilities. Number ③ and final component is the *testing setup*, which includes a set of scripts designed to modify the network state and introduce controlled errors. In this chapter, I will introduce each component in detail and explain their respective designs.

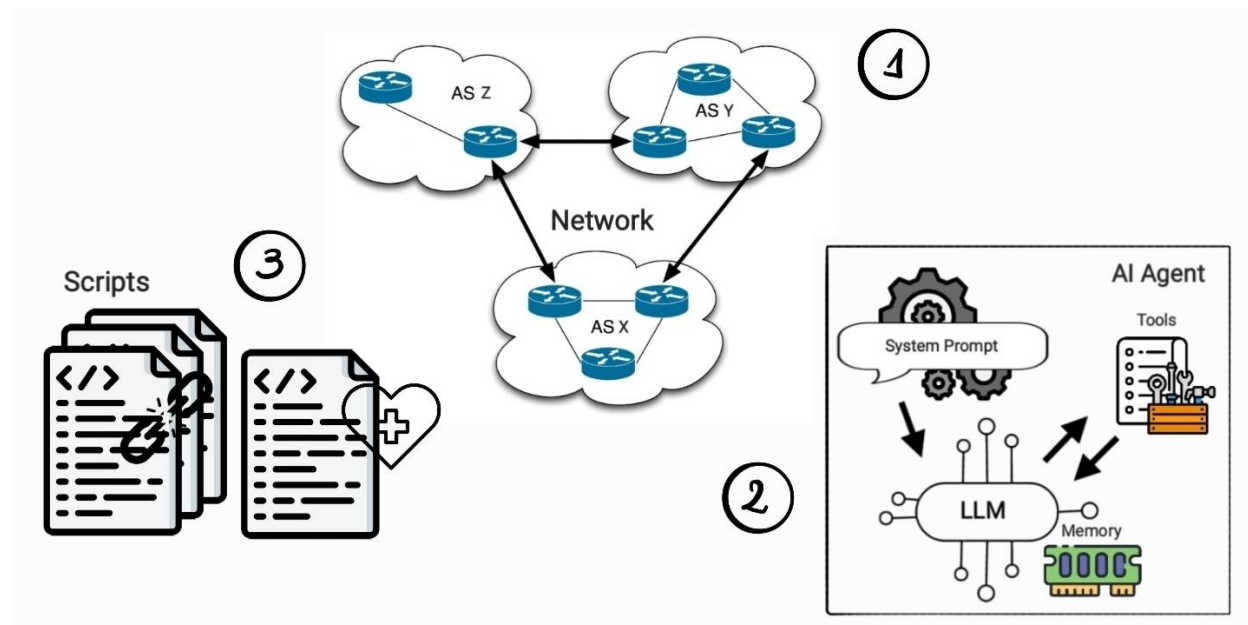


Figure 3.1: System components

3.1 Live Network

In this section, I describe the setup of the network used for this project. The network was configured using the Mini-Internet API [10], with the setup details based on their official documentation and wiki. Each network component operates within its own isolated Linux container, ensuring modularity. Although the API simulates network elements down to Layer 2, this project focuses solely on Layer 3 components, specifically routers and hosts.

I now present the network topology.

Figure 3.2 shows the AS topology, which consists of 12 ASs numbered 1 through 6 and 11 through 16. These ASs are organized into Tier 1, Tier 2 (transit ASs), and Tier 3 (stub ASs). The topology is further enriched by peering relationships established via Internet Exchange Points (IXPs), which provide additional connectivity among ASs.

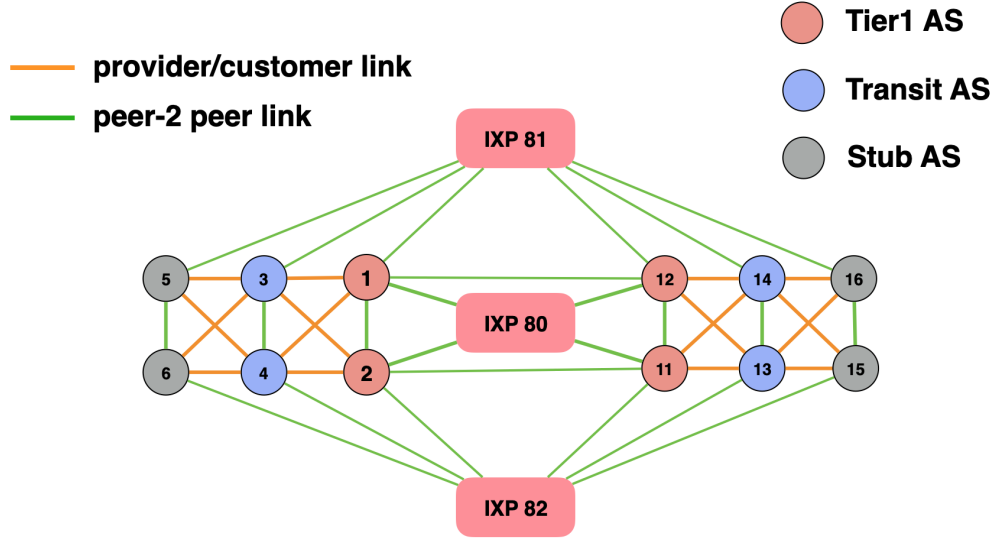


Figure 3.2: The inter AS topology.
This picture was taken from the Mini-Internet Wiki [10].

Figure 3.3 illustrates the intra-AS topology of ASes 3, 4, 13, and 14. This topology consists of eight routers, each directly connected to a dedicated host. Among these routers, six serve as border routers, with each connected to a different external AS, while the remaining two operate solely as internal routers.

All other ASes (1, 2, 5, 6, 11, 12, 15, and 16) are configured with a single router and its corresponding host. All containers and network interfaces follow a consistent naming convention. This consistency facilitates imparting condensed topology information to the agent; moreover, it allows us to infer relationships between devices solely from their names.

To establish BGP and OSPF sessions between routers, the Mini-Internet API leverages the FR-Routing protocol suite [5]. This software enables configuration of the routing protocol stack via the `vttysh` shell, which provides a comprehensive set of commands for protocol configuration, as well as retrieval of routing tables, peer status, and other relevant information.

This shell allows for the customized configuration of routing protocols on each router. I utilized these capabilities to: firstly, introduce intentional misconfigurations in the network; secondly, to enable the agent to collect live network data for diagnostic purposes.

3.2 Agent Design

Now that the network is set up I can move on to introducing the features of the AI agent it is supposed to interact with. The agent was developed using the LangChain framework [2], which offers convenient tools for creating ReAct agents [23] —a structure that integrates reasoning and acting capabilities, as detailed in Chapter 2. The initial base code for the agent was kindly provided by

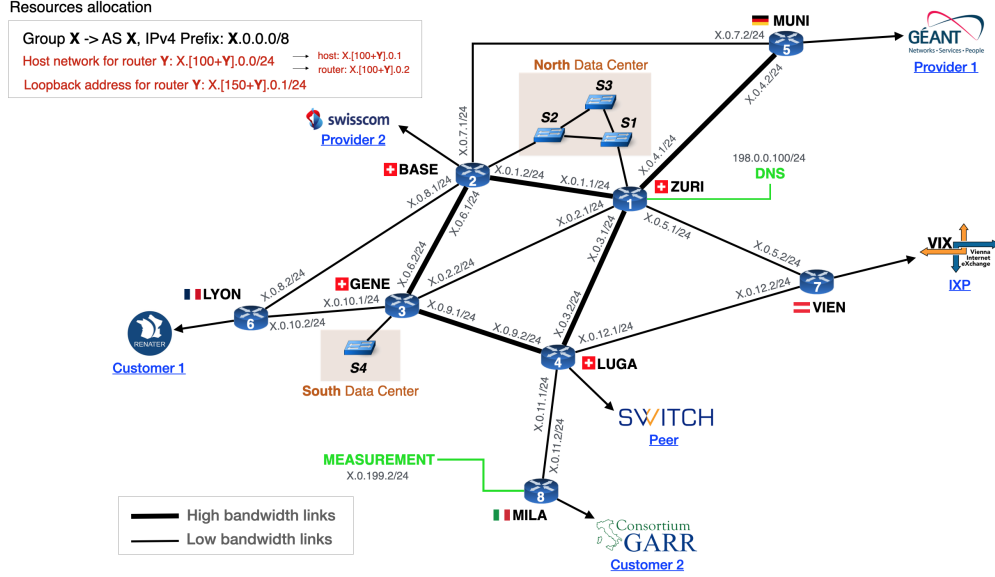


Figure 3.3: The intra AS topology.

This picture was taken from the Mini-Internet Wiki [10].

Ioannis Protopogeros. I have since modified and extended this code to tailor the agent’s functionality to the specific needs of this project.

In the following sections, I will expand on the agent’s base structure, detailing its core components and interaction process. I will also present the design choices made for its customizable elements, including the tools, the underlying LLM model, and the system prompt.

3.2.1 Base Structure

Figure 3.4 illustrates the key components of the AI agent. At its core, the agent is powered by a Large Language Model (LLM), which serves as the primary reasoning engine. As previously mentioned, this type of agent is provided with a memory mechanism that allows it to retain previously requested information and preserve contextual understanding. It is equipped with tools that function as peripheral interfaces that allow it to interact with the network. Additionally, a system prompt provides the agent with essential background information and guidance, helping define its role and tasks in the system.

A typical interaction with the agent unfolds as follows: it starts by receiving a prompt from the user. It then begins processing both the input and the system prompt to understand its assigned role and the task it needs to perform. The agent then initiates a sequence of reasoning steps, invoking tools as needed and interpreting their outputs in an effort to resolve the user’s query, this is the stage where interaction with the live network occurs. Once it has formulated a response, the agent outputs its answer to the terminal. Additionally, it generates a log file in .json format, which records the full sequence of reasoning and task-action steps, along with relevant metadata for each stage.

3.2.2 Tools

After having introduced the structure of the agent, I can now explain how it interacted with the network. I am now going to illustrate all the its tools, one by one, briefly explaining their function.

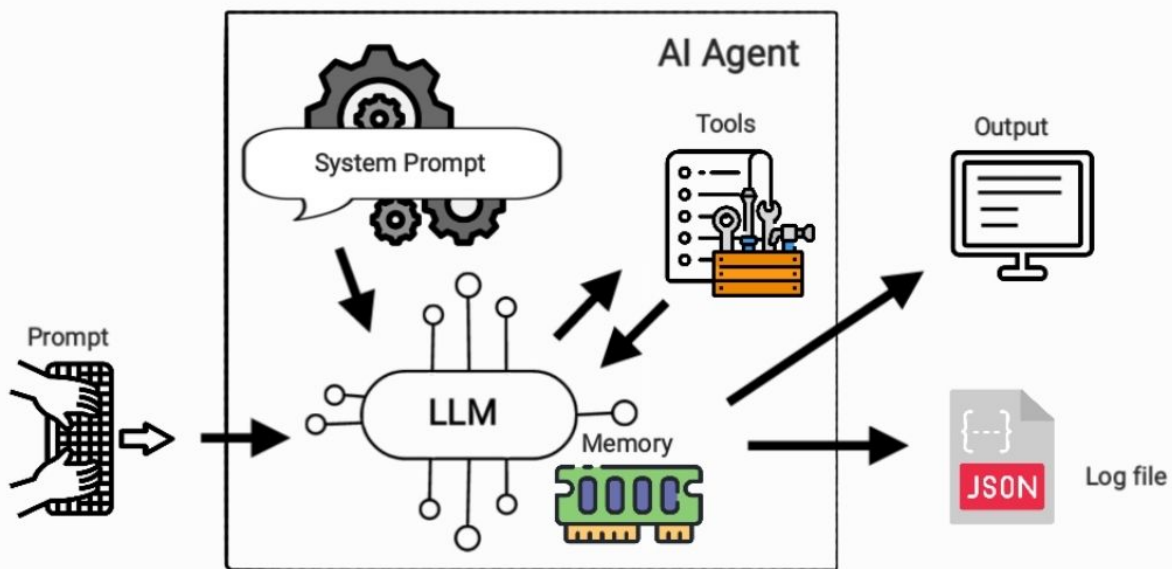


Figure 3.4: AI agent structure

- `list_bgp_containers`: Lists all Docker containers running BGP routers or hosts. This tool was part of the code handed to me by Ioannis Protogeros. I modified the code to include hosts too, and modified the way the output is presented to the agent: to save tokens I made sure only the name would be returned.
- `ping`: Pings a device from a container. This tool was part of the code handed to me by Ioannis Protogeros.
- `show_interface_brief`: Shows a brief summary of interfaces on a router container.
- `show_ip_route`: Shows the IP routing table of a router container.
- `show_ip_ospf_neighbor`: Shows the OSPF neighbors of a router container.
- `show_ip_bgp`: Shows the state of BGP sessions on a router container.
- `show_running_config`: Displays the current running configuration of a router container.
- `traceroute`: Performs a traceroute command from a container to a specified IP, with optional parameters.
- `find_router_ip`: Finds the IP address of a specified router container.
- `find_host_ip`: Finds the IP address of a specified host container.
- `looking_glass_dump`: Retrieves the looking glass output from the perspective of a given router container.

- **consult_man**: Returns pages from *Troubleshooting IP Routing Protocols* [1], an official CISCO handbook, as a text dump. This tool was added as an attempt to aid in root cause analysis of a specific BGP misconfiguration. The pages returned refer only to the solution of this misconfiguration.
- **ManSearch**: Answers questions related to troubleshooting common BGP problems using a searchable *Troubleshooting IP Routing Protocols* page database. This tool too was added as an attempt to aid in root cause analysis of a specific BGP misconfiguration, but this time is implemented as a queryable second agent that acts as manual expert using a combination of document processing, embedding, vector search, and retrieval-augmented generation (RAG).

Most of these tools are simple vtysh commands; the agent only needs to pick which container to inspect and the tool executes the corresponding command on the FRRouting interface of the specified container. This group includes: `ping`, `show_interface_brief`, `show_ip_route`, `show_ip_ospf_neighbor`, `show_ip_bgp`, `show_running_config`.

Most others exploit the access to the docker container and either use some of the services provided by the mini-internet, like `looking_glass_dump`; or they execute commands on the bash command line, like `traceroute`; or parse information to give a concise answer to the agent, like `list_bgp_containers`, `find_router_ip` and `find_host_ip`. **ManSearch** and **consult_man** are special tools meant to integrate a troubleshooting manual [1] into the agent system.

3.2.3 LLM Choice

The choice of LLM for the agent is easily customizable. Obviously, more powerful reasoning models are more likely to yield correct answers, but are also more expensive. I used three different models:

- **Gemini 2.0 Flash**: A lightweight variant in the Gemini family, optimized for speed and low latency with a context window of up to 1 million tokens. Input tokens are priced at \$0.10 per 1 million tokens, output tokens at \$0.40 per 1 million tokens, and cached input tokens are \$0.025. This model is best suited for straightforward queries and lightweight decision-making tasks due to its modest reasoning capabilities, while efficiently handling large documents or extended dialogue histories. [8], [7]
- **GPT-4o (GPT-4 Omni)**: OpenAI’s flagship model, balancing strong reasoning abilities with improved speed and cost-efficiency, it is the most capable excluding the o-series models. It supports a context window of up to 128,000 tokens. Pricing is set at \$2.50 per 1 million input tokens and \$10.00 per 1 million output tokens, cached input tokens are discounted to \$1.25. GPT-4o is designed for complex multi-step reasoning and tool use, making it ideal for applications requiring depth of understanding and scalability. Only text input is considered in this evaluation. [17], [16]
- **o4-mini-2025-04-16**: Part of OpenAI’s o-series, this model focuses on structured, multi-step reasoning tasks such as mathematical problem solving, logical inference, and coding. It supports a context window of up to 200,000 tokens. Pricing is \$1.10 per 1 million input tokens and \$4.40 per 1 million output tokens, cached tokens are offered at \$0.275 per million tokens. The model provides strong reasoning capabilities while remaining lightweight and cost-effective, making it suitable for applications demanding accurate step-by-step logic without the higher costs associated with flagship models. Only text input is considered in this evaluation. [17], [16]

To work around the issue of non-deterministic behavior, I set the model entropy to zero whenever I could. Unfortunately, not all models support setting a low temperature or nucleus sampling probability. For those models I used different techniques to circumvent the problem. See evaluation in chapter 4.

3.2.4 System Prompt

Finally the model needs to be given directions. The system prompt is a great place for giving the model general instructions and knowledge and can greatly impact the kinds of responses it returns. As I tested the agent I realized it would greatly benefit from understanding how the network topology was set up. So I used this space to inject as much information about the network as I could.

I employed three successive variations of the system prompt, each extending the previous one. The exact formulation of all the three system prompts can be found in the appendix, section A. Here I will summarize their content.

- **1st version:** The role of the agent is set to be a network manager and the model is made aware of having to manage a simulated network through a vttysh shell.
- **2nd version:** The agent is given information about the topology of the network in the form of a short description text. There are provided the number of ASs simulate, their AS numbers, their relationship to each other (Peer, Provider, Customer) and their IP prefix and address scheme. Furthermore, I provide a description of the layer three topology: for each router I list its OSPF neighbors with the addresses and subnet interface, as well as the services it is running (DNS, RPKI, MATRIX, etc.). Lastly, the scheme for the container's name is explained so that model may infer the appropriate routers.
- **3rd version:** The model is given information about how to interpret the names of the interfaces that are present on the routers.

3.3 Test Setup

Finally, I present how the agent was tested. I will describe how the experiments were conducted, from initial design to data collection.

The primary goal in designing the tests was to simulate realistic BGP misconfigurations that could plausibly be introduced by an inexperienced network administrator—particularly one encountering BGP for the first time. This idea was inspired by the Mini-Internet project, aiming to emulate mistakes commonly observed in the classroom. I have repurposed common mistakes made by students and present them to the agent as tests in order to evaluate its efficacy. A detailed description of each test type is provided in the following chapter (see Chapter 4).

The tests were designed with a gradient of difficulty in mind. The intent was to ensure that simpler tests would likely be handled successfully by the agent, while more complex ones might reveal its limitations—or ideally, demonstrate capabilities beyond expectations by passing all of them. It is important to note that when I devised the tests, my perception of difficulty was influenced by personal bias—rooted in what I, as a human, would consider challenging. However, the notion of difficulty is inherently subjective and highly dependent on the evaluating entity. Tasks that are straightforward for humans may pose significant challenges for machines, and vice versa. I will explore this discrepancy further in the discussion section of Chapter 4.3.

Another important design consideration was the nature of the prompts given to the agent. I deliberately kept these prompts minimal, including only the essential information. This is to simulate what would happen in a realistic use case: in a real-world environment, these prompts could originate from automated monitoring tools detecting anomalies in the network. The goal is to evaluate whether the agent can independently diagnose the issue based on sparse input, thereby offloading the diagnostic workload to the LLM. A secondary objective, therefore, is to understand what is the minimal amount of information that would be needed by the agent in order to accurately analyze the root causes of network anomalies.

To carry out the experiments, I created a dedicated Bash script—referred to as a *break* script—for every test type. Each script reproduces a specific failure scenario in the network. To streamline the process, I also developed a *reset* script to restore the network to its original, error-free state. It is important to note that the *reset* script is limited to reverting configuration file changes. As such, the misconfigurations introduced are limited to modifications in router configuration files; changes like shutting down interfaces were excluded, as they could not be reliably reverted.

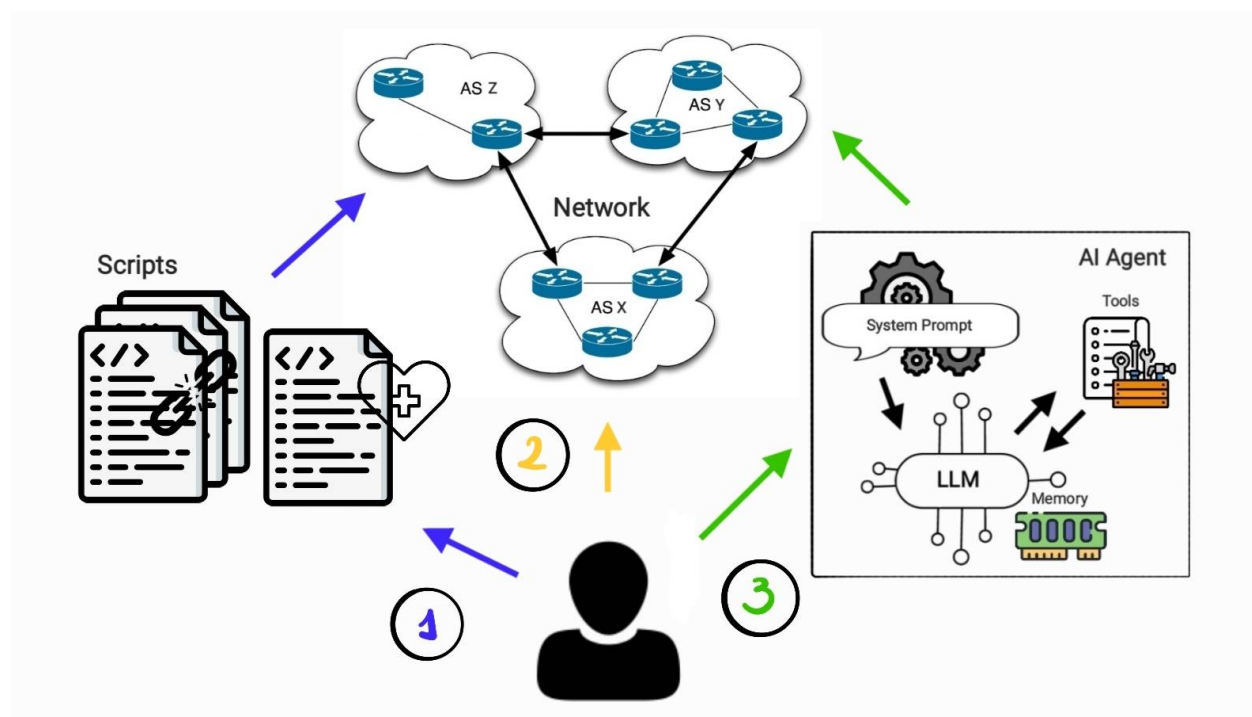


Figure 3.5: Test Setup and Phases

Figure 3.5 illustrates the main phases of the testing process. Each test follows a standard procedure. First, the *reset* script is executed to ensure that the network is functioning correctly and free of any residual errors. Next, the corresponding *break* script is run to introduce the specific misconfiguration. The network gets manually inspected to confirm the misconfiguration is applied and to observe its effects. Based on these observations, the prompt to be submitted to the agent is crafted. Finally, the agent is queried using the generated prompt and allowed to autonomously interact with the network in order to provide a diagnosis.

Chapter 4

Evaluation

In this chapter I will go into three sections: first, the experiment design, where I provide a detailed description of the tests conducted; secondly, results presentation, where I systematically illustrate and explain the data collected; and finally, the discussion, in which I interpret the findings, explore their implications, and identify potential limitations.

4.1 Experiment design

In this section, I describe the experiments designed to evaluate the agent’s performance in handling network misconfigurations. A total of six tests were devised, each involving deliberate modifications to the configuration files of one or more routers. These changes either cause a loss of connection or alter the routing paths within the network. Particular attention is given to Autonomous Systems (ASs) 3, 4, 13, and 14, which possess intra-AS topologies and therefore present the most relevant scenarios for inducing misconfigurations; these will be referred to as the *target* ASs. For each test, I will detail the specific modifications made, the resulting impact on the network, possible variations, and the exact prompts provided to the agent. Finally, I will add my rating of the difficulty of the test.

4.1.1 Unwelcomed Access list

Misconfiguration:

For this test I picked a *target* AS and a second AS. The intention is to make sure the latter is avoided from the *target* AS, meaning any route passing through it is automatically rejected. To make this happen, I created an `as-path access list` named `UNWELCOMED` that flags all routes starting, ending or passing through the second AS. Then I created a new route-map also named `UNWELCOMED` that denies all routes matching the `UNWELCOMED` AS-path access-list, but lets through all the others. Finally, apply the route-map to all OSPF neighbors as a incoming route filter. The reason why the route-map is not applied to the neighbors outside of the AS is to keep all other traffic unaffected.

Effect:

The effect that this (mis)configuration has on the network is to make the *target* AS and the second AS lose connection to each other. A simple request to ping from either side would reveal the issue. Note that the border router from which the *unwelcomed* AS’ own prefix is advertised in the iBGP network, didn’t loose connection, this is a by product of not applying the route-map to external neighbors.

Variation:

A small variation to this test is changing the name of the access-list and route-map from UNWELCOMED to DO_NOT_TOUCH_THIS. The goal is to assess whether the agent is influenced by the wording in the configuration.

Prompt:

After having banned any path with AS 2 in the AS-path from AS 3 routers,

1. Router 3_VIENrouter has lost connection to router 2_ZURIrouter? Can you figure out what happened?

Rating:

I believe this test to be quite easy. There are multiple added lines to ban all routes passing through this one AS, so there isn't much subtlety. While it can be common to avoid malicious ASs, these statements would appear in the route-maps assigned to external neighbors, not internal.

4.1.2 Invert Route-map

Misconfiguration:

This test is inspired by the common error where 'deny' and 'permit' directives are inadvertently interchanged. Picking a border router of a *target* AS, all route-maps present in the configuration file get the permit and deny statements switched. This is done in the script by manually rewriting the 'inverted' statements for each route-map. The reason why picking a border router is important is that there are normally no route-maps assigned to internal neighbors, at least not in this network, so there would be no effect on internal routers.

Effect:

The result of these changes is that routes coming in from the border router are now blocked, as well as the routes going out. (The configuration would allow rpki invalid routes to go out too, but there are none in this network, so there is no additional effect). Because of the resiliency of the network, the routes lost get re-routed out of a different border router, but this means that routers that should be directly connected through a static route are using different paths to connect to each other.

Prompt:

After having inverted all route-maps on 4_MUNIrouter,

1. Router 4_MUNIrouter and 1_ZURIrouter are directly connected, but don't appear to be. Can you figure out what happened?

Rating:

I believe this test to be easy too. It is a bit more tricky as there are no suspicious additional statements, but, as one of the rules being modified is the one permitting its own prefix to be propagated outside, it is quite apparent that something happened to the route-map.

4.1.3 Remove lo Interface

Misconfiguration:

This misconfiguration started as an attempt to set the `loopback` interface down. This could not be done solely altering the configuration file. I noticed from early testing that the agent would pay attention to the interface status (up/down) and, as previously mentioned, changing it would be cumbersome. So, to work around the issue I decided to instead remove the configuration of the `loopback` interface. To achieve this, I removed the IP address connected to this interface. Note

that the interface would still appear to be up.

Effect:

The `loopback` interface act as a always up, stable identifier. It is crucial in routing protocols like OSPF and BGP as the router ID is selected to be the `loopback` IP. For example, a border router would internally redistribute an externally learned route substituting the next-hop with its own `loopback` address. If this IP goes missing the route would momentarily be available but would drop all packets. It would then switch to a different route as soon as the BGP updates are sent across the network.

Variation:

An obvious variation is to remove the IP address of a different interface. I wrote a script that would pick a border router and remove the interface connecting it to a different AS.

Prompts:

After having removed the `loopback` IP from `14_LUGArouter`,

1. Router `14_LUGArouter` (ip: `14.154.0.1`) seems to be unreachable from all other routers. Can you figure out what happened?

After having removed the IP address of interface `ext_12_ZURI` from router `13_BASErouter`,

2. Router `13_ZURIrouters` usually reaches router `12_ZURIrouters` through router `13_BASErouter`. Someone changed some configurations and now `13_ZURIrouters` uses `13_MUNIrouters` to reach `12_ZURIrouters`. Can you figure out what happened?
3. Router `13_BASErouter` and `12_ZURIrouters` are directly connected, but don't appear to be. Can you figure out what happened?

Rating:

I believe this test to be of medium difficulty. There are no suspicious statements, but as soon as the status of the interfaces is checked then it immediately evident that the IP address of the interface is missing.

4.1.4 Violate BGP Policies

Misconfiguration:

This test is supposed to figure out if the agent can reason about BGP policies. To investigate it, I made a *target* AS leak peer and provider routes to its provider. This was easily achieved by removing the `match community x` statement from the route-map of an outgoing link from a customer neighbor to a provider one.

Effect:

The effect produced is that the provider will prefer customer routes to the peer ones and re-route some of its traffic.

Prompts:

After having `4_MUNIrouters` violate policies and leak routes to AS 1,

1. Router `1_ZURIrouters` usually reaches router `2_ZURIrouters` through a direct connection. Something happened and now `1_ZURIrouters` uses `4_MUNIrouters` to reach `2_ZURIrouters`. Can you figure out what happened?
2. Router `1_ZURIrouters` usually reaches router `2_ZURIrouters` through a direct connection. Someone changed some configurations and now `1_ZURIrouters` uses `4_MUNIrouters` to reach `2_ZURIrouters`. Can you figure out what happened?

3. Router 1_ZURIRouter is using its customer, 4_MUNIRouter, to reach a provider of the customer, 2_ZURIRouter. This violates BGP policies. Can you figure out what happened?
4. Router 1_ZURIRouter is using router 4_MUNIRouter to reach 2_ZURIRouter. This violates BGP policies. Can you figure out what happened?

Rating:

I believe this test to be of medium difficulty too. It requires understanding the commercial relationship between ASs and recognizing that the policies are being violated. Once that is clear, the missing statement is quite easy to spot.

4.1.5 No Next-Hop-Self**Misconfiguration:**

In BGP the statement `next-hop-self` is quite important as it is necessary to internally redistribute externally learned routes. In the configuration of all routers, there is such a statement for each iBGP neighbor; this way there is no need to distribute the IP addresses of interfaces that connect to other ASs, but instead the next-hop of the redistributed advertisement is changed to the `loopback` IP address of the redistributing router. This test is meant to assess whether the agent can recognize the absence of these statements. The script targets the configuration file of a single border router and removes all `next-hop-self` statements for each iBGP neighbor from it.

Effect:

The effect is that all routes being redistributed internally by the affected border router are no longer working; this is caused by the fact that the other routers do not know how to reach the next-hop. Because the network is resilient, if there is another neighboring AS advertising a path for the lost prefixes, the corresponding border router will receive these functioning routes. When it realizes the old next-hop is unreachable, it will start to advertise the new path for the affected destinations and connection will be restored to the network. This implies that the path to those destinations has been re-routed.

Variation:

Instead of removing `next-hop-self` for all iBGP neighbors, only an internal router has the statement deleted in the *affected* border router configuration. Now there is no new route advertised, as all other border routers can still use the old routes. So the internal router loses connection to the AS reached through the *culprit* border router, since the next-hop is unreachable and there are no other routes advertised.

Prompts:

After having removed all `next-hop-self` statements from all neighbors on 13_BASErouter,

1. Router 13_ZURIRouter usually reaches router 12_ZURIRouter through router 13_BASErouter. Something happened and now 13_ZURIRouter uses 13_MUNIRouter to reach 12_ZURIRouter. Can you figure out what happened?
2. Router 13_ZURIRouter usually reaches router 12_ZURIRouter through router 13_BASErouter. Someone changed some configurations and now 13_ZURIRouter uses 13_MUNIRouter to reach 12_ZURIRouter. Can you figure out what happened?

After having removed the `next-hop-self` statement from 13_ZURIRouter on 13_BASErouter,

3. Router 13_ZURIRouter and 12_ZURIRouter have lost connection to each other. Can you figure out what happened?

4. Router `13.ZURIRouter` and `12.ZURIRouter` have lost connection to each other, but all other routers in AS 13 can still reach router `12.ZURIRouter`. Can you figure out what happened?
5. Router `13.ZURIRouter` has lost connection to router `12.ZURIRouter`, even though all other routers are still reachable. All the other routers from AS 13 can still reach `12.ZURIRouter`. Can you figure out what happened?
6. Router `13.ZURIRouter` and `12.ZURIRouter` have lost connection to each other. Can you figure out what happened? Use the manual if you aren't sure.
7. Router `13.ZURIRouter` and `12.ZURIRouter` have lost connection to each other. Use the manual to figure out what happened.

Rating:

I believe this test to be a bit more than medium difficulty. It requires understanding that the issue lies within the border router, which can be done relatively easily by analyzing the routes available for the indicated AS. Then the solution can be found relatively easily by paying attention to the lack of `next-hop-self` statements.

4.1.6 No Null0**Misconfiguration:**

Null0 is a virtual interface always up that drops all packets routed through it. It can be very useful for aggregating routes on the same subnet and advertising a single prefix, avoiding routing loops for addresses not connected to any device. This test removes the configuration about this interface in the configuration file of every router part of a *target* AS.

Effect:

Since BGP advertisements require that the prefix advertised is present in the routing table, once this disappears the routers aren't advertising their own subnet anymore, rendering the AS unreachable.

Prompts:

After having removed the Null0 interface from 14,

1. AS14 has lost connection to all other ASs. (AS14 is comprised of all containers starting with 14_). Can you figure out what happened?

Rating:

I believe this test to be hard. It requires understanding that there needs to be a route advertisement in the routing table specifically designated to advertise the whole prefix; its absence implies not being able to be advertise its own prefix. If we desired to fix it too, then it also requires knowing that a fake interface is required to be used just as a placeholder.

4.2 Results

In this section, I present all the relevant data collected during the evaluation phase. The results are organized into four subsections: Gemini, which covers the performance of the Gemini 2.0 Flash model; GPT-4o and o4-mini, each following the same format; and finally, Token Usage and Expenses, where I analyze the data related to token consumption and the monetary cost of running these experiments.

Before examining the results model by model, I will first explain the criteria used to determine whether a test was considered a pass or a failure. The only viable method I found was to manually

review each agent’s reasoning process and final output. As a result, the evaluation is not fully systematic and includes a few ambiguous cases where the outcome is not entirely clear. These instances are explicitly noted, and all corresponding agent outputs are included in Appendix B. One final note: not all tests were conducted for every model. Depending on the model’s observed performance, I adjusted the scope of the evaluation accordingly—setting more conservative objectives for models that performed poorly, and expanding the range of tests for those that showed promising results.

4.2.1 Gemini

Here I present all the results I collected testing the model ‘Gemini 2.0 Flash’. Note that for this model I had not set any parameter to control the entropy as I had not been aware of this possibility yet. This means that the outputs are highly random. To try and get an idea of what the model performance can be I repeated each test 3 times. In table 4.1 can be found the data.

- **Prompt:** The number refers to the enumeration listed in the explanation of the misconfiguration.
- **Misconfiguration:** Can either be the original one or a variation introduced beforehand.
- **Successful test:** Indicates whether I determined the final answer of the model to accurately individuate the issue.
- **Total tokens:** Displays the total amount of tokens spent for the experiment.
- **Tool calls:** Expresses how many tools the model called.

For the entirety of these tests the prompt version was kept the same as the 1st version. I have

Table 4.1: Results from testing Gemini 2.0 Flash

	Prompt	Misconfiguration	Successful test	Token usage	Tool calls
Unwelcomed	1	original	✓	14002	6
Access list	1	original	✗	44443	13
	1	original	✗	67394	15
Invert	1	original	✗*	14123	5
Route-map	1	original	✗*	14073	5
	1	original	✓*	14874	15
Remove	1	original	✗	34065	8
lo Interface	1	original	✗	26366	5
	1	original	✗	19480	4
No	1	original	✗	114406	16
Next-Hop-Self	1	original	✗	13777	6
	1	original	✗	26966	8
No	1	original	✗	22662	11
Null0	1	original	✗	54721	13
	1	original	✗	81953	15

marked the Invert Route-map test with an asterisk (*) due to the fact that the agent realizes

the route-map rules are impeding some advertisements, getting close to the solution, but also hallucinates other errors alongside it. For the exact output answers please refer to appendix B. The data shows some disappointing results, out of five tests, only two are successful; One of the two is debatably unsuccessful, and the other is correct one time out of three. In many instances the LLM hallucinates issues related to the 'black hole' present in the IP routes, which is actually necessary to advertise its own prefix.

4.2.2 GPT-4o

To try and get better results than the ones provided by Gemini I employed a more powerful LLM, GPT-4o and added more topology awareness through the system prompt. With this model I was able to set the entropy to zero, thus achieving a deterministic agent that I only needed to test once. To increase its understanding of the network I changed the system prompt to its 2nd version (3.2.4). I then conducted the first two experiments and realized that there was some information missing that would have been helpful to solve the task and then changed the prompt to its final version. Since the first experiment was successful I did not run it again, but re-run the second one. The ** in the prompt column in the table 4.2 indicate at which point the prompt has been changed.

The results from this second model where also not promising, only one test is successful. Although

Table 4.2: Results from testing GPT-4o

	Prompt	Misconfiguration	Successful test	Token usage	Tool calls
Unwelcomed Access list	1	original	✓	31666	10
Invert	1	original	✗	4387	2
Route-map	1**	original	✗	4975	2
Remove lo Interface	1	original	✗	34522	13
No	1	original	✗	6789	3
Next-Hop-Self	2	original	✗	7504	3

the model does not to hallucinate about the black hole, it instead is quite vague in its answers and does not investigate further, for tests **Remove lo Interface** and **Invert Route-map**; or is confidently wrong in the case of **No Next-Hop-Self**. In this specific instance it pointedly compares the AS-PATH length of the two contending routes, through **MUNI AS11**, **AS12**, and through **BASE AS12**, but it then claims that the one through **MUNI** is shorter than the one through **BASE**. This raises the question of whether the agent can understand the concept of path length.

4.2.3 o4-mini

Since the results from GPT-4o were not encouraging either, I did not waste time re-running the Gemini model with the advanced system prompt and instead looked for a more powerful reasoning model, eventually settling on o4-mini. Unfortunately, this model would not allow me to set its entropy. Due to the reasoning capabilities of this model, tests performed with it took several minutes to complete, whereas previously, the longest duration did not exceed one minute. Due to this reason and to the fact that it was quite expensive to run tests, I decided to only re-run tests if they turned out unsuccessful. This model proved better than its predecessor, so I extended the

scope of the testing. *For the entirety of these tests I used the 3rd version of the system prompt.* The tables 4.3 and 4.4 show the the summary of the results.

Table 4.3: Results from testing o4-mini

	Prompt	Misconfiguration	Successful test	Token usage	Tool calls
Unwelcomed	1	original	✓	236222	21
Access list	1	variation	✗	110263	13
	1	variation	✓	269070	21
Invert Route-map	1	original	✓	32819	6
Remove lo Interface	1	original	✓	34065	8
	2	variation	✗	80750	11
	3	variation	✓	12822	3
No Next-Hop-Self		see next table			
No	1	original	✗	282923	24
Null0	1	original	✗	293243	23
	1	original	✗	196537	19
Violate BGP Policies	1	original	✗	37488	5
	2	original	✗	41466	5
	3	original	✓	34991	5
	4	original	✗	50836	6
	4	original	✓	34444	4

This model performs much better and clears five tests out of 6. If we include the variation I also added, the number increases to eight out of nine.

Tests **Unwelcomed Access list**, **Invert Route-map**, **Remove lo Interface** and their variations are cleared quite easily, as well as the original misconfiguration of **No Next-Hop-Self**.

In order to pass **Violate BGP Policies** the agent requires to be told that BGP policies are being violated or it will not investigate the issue further.

During the test **No Null0** the model hallucinates pretty heavily getting nowhere close to the solution, so I just accepted this is too hard for the model to achieve.

Concerning the variation of the test **No Next-Hop-Self**, I ended up going down a rabbit hole trying to make the agent pass this test.

Adding hints to the prompt

The first attempt I made was to change the prompt and try to give more hints to the agent, with unfortunately no success.

Incorporating the manual through a second agent

Secondly I tried to integrate a troubleshooting manual into the agent offering it as a tool with the hope that more in depth knowledge of the issue would allow it to succeed. To avoid spending too

Table 4.4: Results from testing o4-mini on No Next-Hop-Self

	Prompt	Misconfiguration	Successful test	Token usage	Tool calls
No	1	original	✗	75049	9
Next-Hop-Self	2	original	✗	54815	8
No Manual	2	original	✓	68217	7
	2	original	✓	68217	7
	3	variation	✗	174445	15
	3	variation	✗	120299	12
	3	variation	✗	158286	14
	4	variation	✗	11573	2
	4	variation	✗	17245	3
	5	variation	✗	49051	8
	5	variation	✗	148223	14
	5	variation	✗	148223	14
Manual implemented as 2nd agent	3	variation	✗	134809	12
	3	variation	✗	101062	11
	6	variation	✗	199782	17
	6	variation	✗	211045	17
	7	variation	✗	100079	12
	7	variation	✗	108974	11
	7	variation	✗	69635	9
	7	variation	✗	81147	10
Manual implemented as a (small) text dump	7	variation	✓*	191476	12
	7	variation	✗	275041	16
	7	variation	✗	372361	19

many resources and trying to control what part of the manual would be queried, I only used the section of the manual specific to debugging the issue of a missing **next-hop-self**.

I first implemented this manual tool, **ManSearch** as second, more specialized agent that could answer queries about the manual. I tried prompting without giving any extra instruction, to see if it would call the manual itself; I then tried suggesting it could use the manual, but both times the agent did not use the tool. I then requested it to use the manual and this time it did, but only two times out of 4. The test still failed, the interaction between the two agent wasn't particularly smooth; the main agent would not be particularly good at asking question and the second one would not be very good at answering them.

Incorporating the agent through a text dump

So I then replaced the **ManSearch** tool with a second implementation that would return the entire textual content of the pages in the manual, **consult_man**. The goal was to improve the clarity of the knowledge transmitted. I directly prompted it to use the manual and on the first try it mostly solved the test. It is flagged with an asterisk (*) like the Gemini result since this result is ambiguous. Exact wording can be found in the appendix A. While it correctly identifies the root cause, the solution proposed is flawed; instead of installing the correct **next-hop-self** statement for neighbor 13.151.0.1 (13_ZURIrouter) on 13_BASErouter, the statement is suggested for neighbor 13.152.0.1 (13_BASErouter own IP address). This mistakes undermines the belief that the LLM

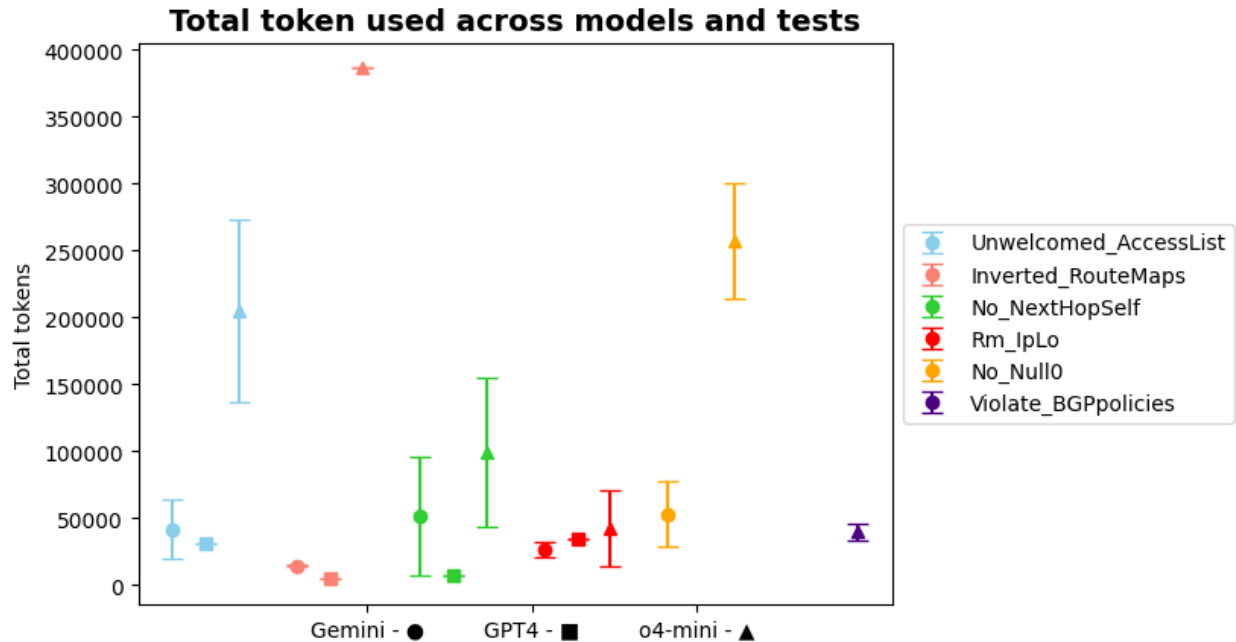


Figure 4.1: Total amount of tokens spent on average session. The results are grouped by test type and model type.

actually understands the issue, though I will note, finding an effective solution was out of the scope of this project. Refer to the discussion, section 4.3, for further analysis of these results.

4.2.4 Token Usage

Since tokens are central to understanding computational cost and resource consumption, I analyzed the metadata collected in the log files. I aim to address two questions: first, how many tokens do these models consume per test; and second, how many tokens are consumed by each individual tool. To answer these questions, I processed the data and generated several graphs to visualize the results.

Token consumption per model

Figure 4.1 shows the total amount of tokens, both prompt and completion, spent on the average session. The results are group by test type and model type. This graph shows that o4-mini model consumes the most amount of tokens, in some case significantly more that the other two models, which instead have comparable expenditures.

An important caveat is that not all tokens showed in the previous graph are counted the same when computing power and cost consumption. Most of these tokens are prompt tokens that need to be fed to the LLM to maintain context, which can thankfully be cached. While caching was not activated for Gemini, but is it reasonable to believe it would have comparable trends as GPT-4o and o4-mini. For these latter models, almost half of the tokens are cached, in some cases even up to 75%. Note that o4-mini has a consistently smaller percentage of tokens cached compared to GPT-4o.

The observation that o4-mini, a more powerful model, consumes more tokens and generally passes more tests raises several important questions.

1. Do more powerful models use more tools and gather more information?

To answer the question I collected data regarding the number of tool calls performed by each model at for every test type. The results are presented in figure 4.2. The graph does not indicate a definitive trend; the number of calls are sometimes comparable, sometimes many more, but it seems to depend more on the test type than on the model.

2. Does employing a greater number of tools lead to better, more accurate answers?

To explore this second question, I will refer to the data presented in the previous section. Looking at the tables 4.1, 4.2, 4.3, 4.4 and comparing the number of calls in the passed tests versus the failed ones, there is again no conclusive trend. For example, in the testing of Gemini, while more tool calls where conducive to somewhat pass the `Invert Route-map` test, the opposite was true for the `Unwelcomed Access list` test. Furthermore, the same test, `Unwelcomed Access list`, was passed by o4-mini when calling more tools, contradicting what happened for Gemini.

3. Is higher token consumption in certain models primarily driven by an increased number of tool calls?

To answer, I decided to plot the distribution of the total amount of tokens divided by the respective number of tool calls. The graph can be found in figure 4.3. What is illustrated reiterates what had been shown in the total amount of tokens, the o4-mini model spends more tokens, albeit the difference is less pronounced.

Token consumption by tool type

Here I conclude the comparison between models and consider instead the relation between tool type and token usage.

To investigate, I collected all the data on every tool ever used during testing. For each specific tool I listed all the instances it was called and counted all the tokens that were outputted during the call. Figure 4.4 present the graph were this information is summarized. As expected some tools are more expensive than others as their output is lengthier. I wanted to dedicated special attention to the two tools that implement the integration of the manual: `ManSearch` and `consult_man`. While the performance of `ManSearch` was not impressive, the amount of tokens spent for each calls are ten times less that those spent for `consult_man`. The latter was the most expensive by a huge margin, spending more than twice the amount of the second most expensive tool.

4.2.5 Expenses

Because I used API keys, I have detailed data on the exact amount of money spent for each model. The Gemini key did not require setting up a billing account, so the use of Gemini 2.0 Flash was completely free.

In contrast, GPT-4o and o4-mini required an OpenAI key linked to an account that tracked all requests and their associated usage.

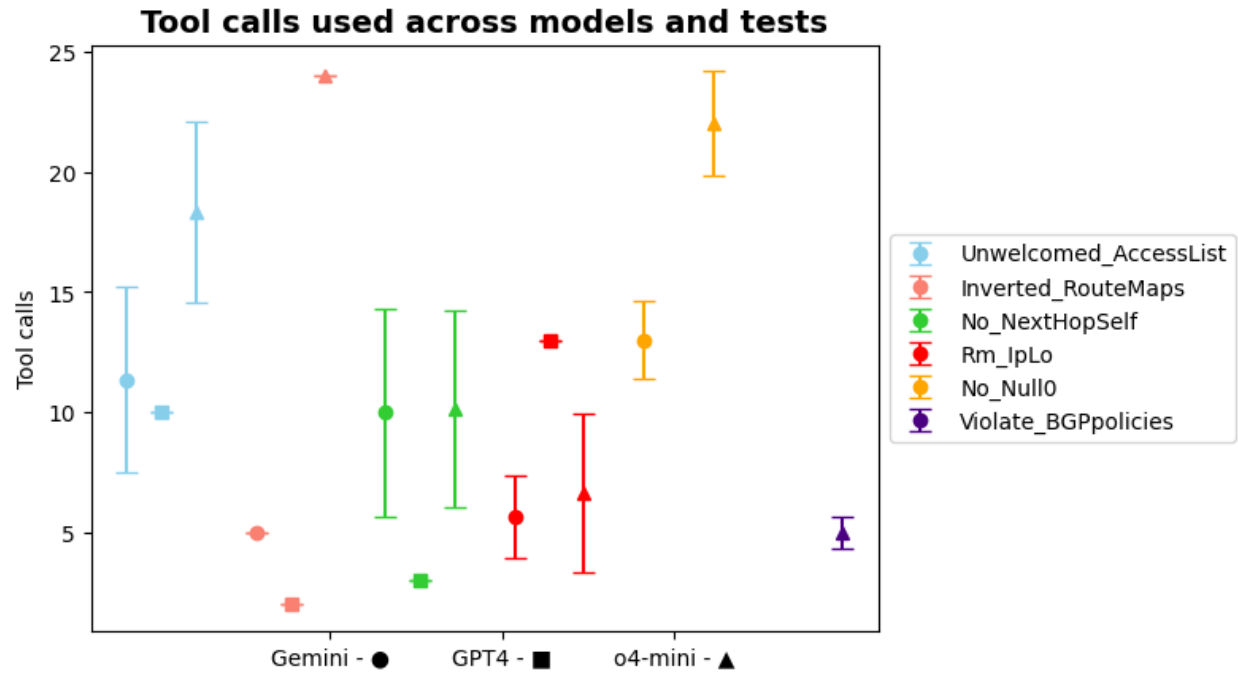


Figure 4.2: Number of tool calls requested during the average session. The results are grouped by test type and model type.

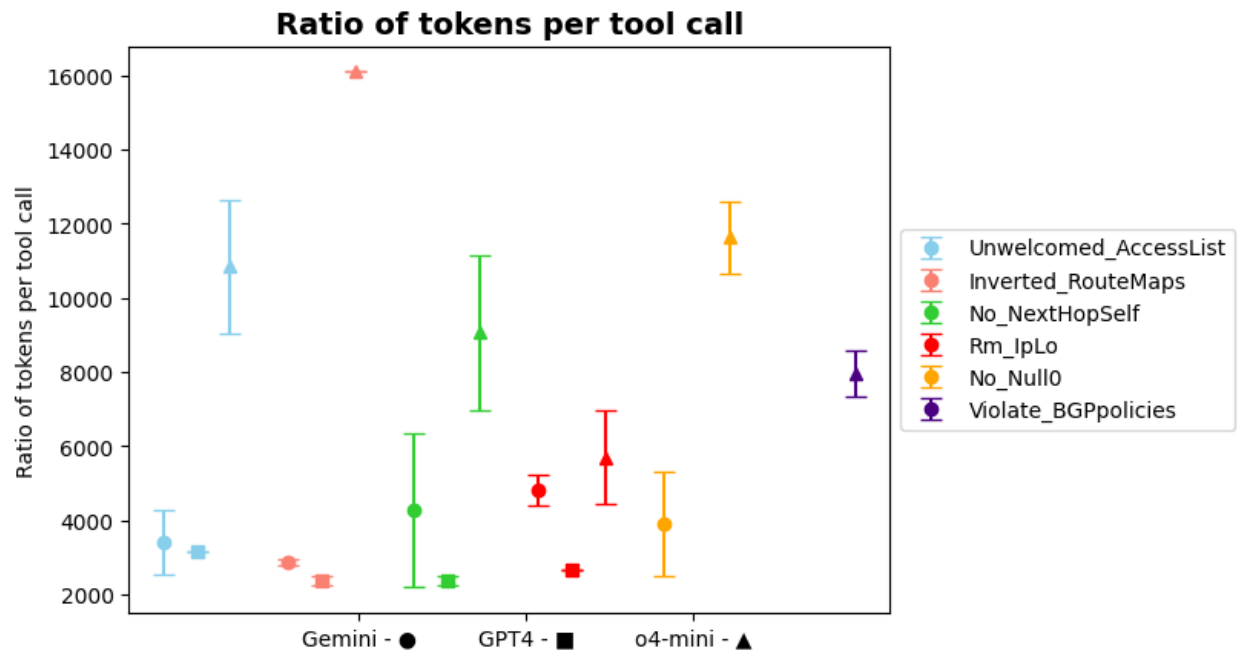


Figure 4.3: Total tokens per tool call for the average session. The results are grouped by test type and model type.

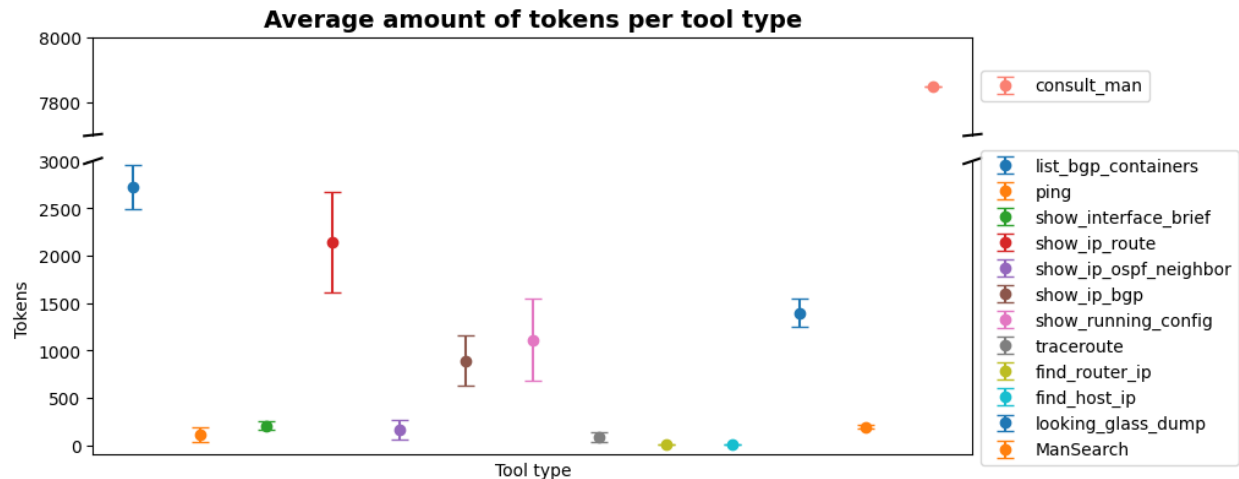


Figure 4.4: Average token usage per tool type

In total, I spent \$0.401 for GPT-4o and \$5.099 for o4-mini. Although GPT-4o is generally considered more expensive, I only used it for six tests and employed less token-intensive tools with it. To ensure a fair comparison, I will report the expenditure before implementing the manual tools. Notably, on June 13th, I ran the first six tests with o4-mini, spending \$1.437.

4.3 Discussion

In this section, I will discuss the data I have collected and share my opinions on whether using AI agents to troubleshoot networks is a viable and effective approach. I will compare different LLMs, taking into account their respective costs and performance. Additionally, I will explore the limitations of using non-deterministic models in troubleshooting scenarios. The discussion will also cover the strengths and weaknesses of AI agents in this context, highlighting what tasks they excel at and where they encounter difficulties. Finally, I will present the reflection on the attempts to incorporate a manual into the agent's knowledge base.

From the results gathered, it is evident that a more powerful reasoning model is necessary to effectively debug network misconfigurations. As demonstrated in the previous section, only the o4-mini model managed to pass a satisfactory number of tests, though it did not clear them all. However, o4-mini was also the most expensive model in terms of token consumption. While it was not the single most costly model overall, it did spend slightly more tokens than GPT-4o for the same number of tests, as detailed in the expenditure section. Analyzing the token usage data, it appears that the number of tool calls was not the key factor in whether a model passed a test. Instead, the advantage of o4-mini seems to stem from its extensive use of reasoning tokens, which are costly and cannot be cached. This explains why o4-mini had such higher token usage, as was demonstrated in figures 4.1, 4.3, and a reduced ratio of cachable tokens.

The unfortunate necessity of using this reasoning model means that the agent behaves non-deterministically as shown before. Although o4-mini was able to solve most of the tests, there is no clear indication of how consistently it can replicate this success. This uncertainty makes it difficult to accurately assess its true performance and undermines trust in deploying such a system in real-world scenarios. I believe further work is required to improve reliability before this technology

can be confidently adopted for network troubleshooting.

Considering the gradient of difficulty presented by the tests, the model performances largely aligned with my personal assessment, though I believe the machine and human agents each have distinct strengths. The model demonstrated strong capabilities in following and interpreting rules—such as those governing route-maps, access lists, and policy logic — which can sometimes be challenging even for experienced human operators. Conversely, it struggled with tasks requiring deeper network understanding, such as identifying which routers belong to the same AS or applying nuanced protocol knowledge, like the correct use of `next-hop-self` statements. These latter aspects are generally easier for network managers with field experience. This complementary nature suggests that integrating AI tools with human expertise could yield more effective troubleshooting outcomes.

Finally, I would like to comment on the attempt to integrate a manual into the agent. Unfortunately, this effort did not achieve the desired results. The `ManSearch` tool performed poorly, largely due to ineffective interaction between the two agents. Meanwhile, the `consult_man` approach barely passed the test, thanks to clearer information, but consumed an excessive number of tokens — even for consulting a small section of the manual. This high cost makes scaling this method to cover the entire manual impractical. A more viable path to incorporating domain-specific knowledge would be to either enhance the coordination between the agents or fine-tune the model with relevant field-specific data.

For a discussion on how to improve on these result go to the Outlook, chapter 5.

Chapter 5

Outlook

I believe that AI agents possess promising capabilities for troubleshooting network issues. However, they are still far from ready for deployment and require significant improvements to become truly reliable and adept at this task.

If I had more time to continue working on this project, there are several areas I would focus on to improve results.

5.1 Improvements to my work

Firstly, I would re-evaluate the integration of the manual. I would revisit the concept of an expert agent and experiment with more advanced techniques to ingest and process manual content — such as using more sophisticated loaders that perform Optical Character Recognition (OCR) and document layout analysis. The goal would be to scale the integration to cover the entire manual corpus and enhance the agent’s ability to generate relevant and accurate responses. Additional effort would also be required on the main agent’s side, particularly in instructing it on how to query the manual effectively and how to incorporate the calls to the manual tool into a the diagnostic flow.

Secondly, I would explore more advanced methods for embedding topology knowledge directly into the agent. Ideally, this would include restricting the agent’s knowledge and access to a single target AS, to better reflect real-world operational environments.

Lastly, I would introduce simulated network failures and evaluate the model on a broader and more diverse set of problems beyond static misconfigurations.

5.2 Future Work

In the longer term, I would recommend future work in several directions: the use of multiple collaborative agents to monitor, relay, analyze, resolve, and validate network issues; testing this technology in more complex and realistic environments where failures and misconfigurations occur simultaneously; and finally, fine-tuning a model specifically for the network troubleshooting domain to improve its efficiency and reliability.

Chapter 6

Summary

This thesis explored the use of LLM-based AI agents to autonomously diagnose network misconfigurations, specifically BGP-related issues, in a simulated environment. A custom agent was built using the LangChain framework, integrated with tools to interface with routers and enriched with network topology awareness through the system prompt. Three different LLMs—Gemini, GPT-4o, and o4-mini—were evaluated through structured experiments. Only the o4-mini model demonstrated satisfactory diagnostic capabilities, but at a high computational cost and with non-deterministic behavior. While the results suggest that LLM agents can assist with some aspects of network troubleshooting, the technology remains limited by model inconsistency, lack of domain knowledge, and high token consumption. Further development, including potentially model fine-tuning and better knowledge integration, is necessary to make these agents viable in production environments.

Bibliography

- [1] Zaheer Aziz et al. *Troubleshooting IP Routing Protocols*. 1st. Cisco Press, 2002. ISBN: 1587050196.
- [2] Harrison Chase. *LangChain*. Oct. 2022. URL: <https://github.com/langchain-ai/langchain>.
- [3] Peter Clark et al. “Think you have solved question answering? Try ARC, the AI2 Reasoning Challenge”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2018, pp. 4140–4153.
- [4] Karl Cobbe et al. “Training Verifiers to Solve Math Word Problems”. In: *arXiv preprint arXiv:2110.14168* (2021).
- [5] FRRouting Project. *FRRouting: A network routing software suite*. <https://frrouting.org>. Version 10.1.1 released 2024-09-12. 2024.
- [6] In Gim et al. *Prompt Cache: Modular Attention Reuse for Low-Latency Inference*. 2024. arXiv: 2311.04934 [cs.CL]. URL: <https://arxiv.org/abs/2311.04934>.
- [7] Google. *Gemini 2.0 Flash Pricing*. Accessed: 2025-07-03. 2025. URL: <https://ai.google.dev/gemini-api/docs/pricing>.
- [8] Google. *Gemini API Models Documentation*. Accessed: 2025-07-03. 2025. URL: <https://ai.google.dev/gemini-api/docs/models>.
- [9] Dan Hendrycks et al. “Measuring Massive Multitask Language Understanding”. In: *arXiv preprint arXiv:2009.03300* (2020).
- [10] Thomas Holterbach et al. “An Open Platform to Teach How the Internet Practically Works”. In: *SIGCOMM Comput. Commun. Rev.* (2020). URL: <https://doi.org/10.1145/3402413.3402420>.
- [11] Labelbox, Inc. *Zero-Shot Learning vs. Few-Shot Learning vs. Fine-Tuning: A technical walk-through using OpenAI’s APIs & models*. <https://labelbox.com/guides/zero-shot-learning-few-shot-learning-fine-tuning/>. Accessed: 2025-06-24. Mar. 2025.
- [12] Guohao Li et al. “CAMEL: Communicative Agents for “Mind” Exploration of Large Language Model Society”. In: *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*. arXiv preprint arXiv:2303.17760. 2023.
- [13] Stephanie Lin, Jacob Hilton, and Owain Evans. “TruthfulQA: Measuring How Models Mimic Human Falsehoods”. In: *arXiv preprint arXiv:2109.07958* (2021).
- [14] Humza Naveed et al. *A Comprehensive Overview of Large Language Models*. 2024. arXiv: 2307.06435 [cs.CL]. URL: <https://arxiv.org/abs/2307.06435>.
- [15] NVIDIA Corporation. *Explaining Tokens — the Language and Currency of AI*. <https://blogs.nvidia.com/blog/ai-tokens-explained/>. Accessed: 2025-06-24. Mar. 2025.
- [16] OpenAI. *OpenAI API Pricing*. Accessed: 2025-07-03. 2025. URL: <https://platform.openai.com/docs/pricing>.

- [17] OpenAI. *OpenAI Platform Documentation: Models Overview*. Accessed: 2025-07-02. 2024. URL: <https://platform.openai.com/docs/models>.
- [18] OpenAI. *Prompt Caching*. Accessed: 2025-06-25. 2024. URL: <https://platform.openai.com/docs/guides/prompt-caching>.
- [19] OpenAI. *What are tokens and how to count them?* <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>. Updated June 2025; accessed 2025-06-24.
- [20] Toran Bruce Richards. *AutoGPT: An Open-Source Autonomous AI Agent*. <https://github.com/Significant-Gravitas/AutoGPT>. Accessed: 2025-07-03. Mar. 2023.
- [21] Aarohi Srivastava, Eric Xia, Ananya Kirubarajan, et al. “Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models”. In: *arXiv preprint arXiv:2206.04615* (2022).
- [22] The Research Nest (XQ). *Explained: Attention Mechanism in AI*. <https://medium.com/the-research-nest/explained-attention-mechanism-in-ai-e9bb6f0b0b4d>. Accessed: 2025-06-24. June 2025.
- [23] Shunyu Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: 2210.03629 [cs.CL]. URL: <https://arxiv.org/abs/2210.03629>.
- [24] Rowan Zellers et al. “HellaSwag: Can a Machine Really Finish Your Sentence?” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2019, pp. 4798–4807.

Appendix A

System Prompt

- 1st variation:

```
"You are an expert network manager."  
"You may need to use tools to manage your simulated network, which includes  
↪ BGP routers."  
"You may need to list the BGP containers and execute vtysh commands on them.  
↪ You have an available tool for that"  
"You may need to figure out appropriate vtysh commands to analyse the  
↪ network. You can execute them with the appropriate tool. You can also  
↪ ping other devices to diagnose issues."
```

Note that this is part of the original code of Ioannis Protopogeros.

- 2nd variation: (only new lines)

```
"The network that you are working with 12 ASs. ASs 1, 2, 11, 12 are a top  
↪ clique. ASs 3 and 4 are customers of ASs 1 and 2. ASs 3 and 4 are  
↪ providers of 5 and 6. ASs 13 and 14 are customers of ASs 11 and 12. ASs  
↪ 13 and 14 are providers of 15 and 16."  
"These pairs are peers: (3, 4), (5, 6), (13, 14), (15, 16). These tuples are  
↪ connected through IXP (1, 3, 5, 12, 14, 16), (2, 4, 6, 11, 13, 15)"  
""ASs 3, 4, 13 and 14 have a layer 3 topology as follow:  
AS X has IP prefix X.0.0.0/8
```

```
Host network and subnets for router Y:  
Host: X.(100 + Y).0.1/24  
Router: X.(100 + Y).0.2/24
```

```
Loopback interface for router Y:  
X.(150 + Y).0.1/24
```

L3 (layer three) network:

ZURI (id 1) is a router connected to a Layer 2-DCN instance

It is connected to the following routers:

BASE on interface X.0.1.1/24

GENE on interface X.0.2.1/24

LUGA on interface X.0.3.1/24

MUNI on interface X.0.4.1/24

VIEN on interface X.0.5.1/24

It also runs the DNS service on port 198.X.0.100/24

BASE (id 2) is a router connected to a Layer 2-DCN instance

It is connected to the following routers:

ZURI on interface X.0.1.2/24

GENE on interface X.0.6.1/24

MUNI on interface X.0.7.1/24

LYON on interface X.0.8.1/24

It is connected to a Provider AS

GENE (id 3) is a router connected to a Layer 2-DCS instance

It is connected to the following routers:

ZURI on interface X.0.2.2/24

BASE on interface X.0.6.2/24

LUGA on interface X.0.9.1/24

LYON on interface X.0.10.1/24

LUGA (id 4) is a router connected to a routinator instance

It is connected to the following routers:

ZURI on interface X.0.3.2/24

GENE on interface X.0.9.2/24

MILA on interface X.0.11.1/24

VIEN on interface X.0.12.1/24

It also runs the RPKI service of port X.104.0.1/24

It is connected to a Peer AS

MUNI (id 5) is a router connected to a host

It is connected to the following routers:

ZURI on interface X.0.4.2/24

BASE on interface X.0.7.2/24

It is connected to a Provider AS

LYON (id 6) is a router connected to a host

It is connected to the following routers:

BASE on interface X.0.8.2/24

GENE on interface X.0.10.2/24

It also runs the MATRIX service on port X.0.198.2/24

It is connected to a Customer AS

VIEN (id 7) is a router connected to a host

It is connected to the following routers:

```

    ZURI on interface X.0.5.2/24
    LUGA on interface X.0.12.2/24
It also runs the MATRIX\_TARGET service
It is connected to a Peer AS

```

```

MILA (id 8) is a router connected to a host
It is connected to the following routers:
    LUGA on interface X.0.11.2/24
It also runs the MEASUREMENT service on port X.0.199.2/24
It is connected to a Customer AS""

```

```

""Note: the containers have name of the type:

```

- [AS num]_[router name in all caps]router, is a router where AS num
 - specifies the AS number, router names can be ZURI, BASE, GENE, LUGA,
 - MUNI, LYON, VIEN, MILA
- [AS num]_[router name in all caps]host, is a host where AS num
 - specifies the AS number, the router name specifies to which router it
 - is connected, it can end with a 0 or a 1, router names can be ZURI,
 - BASE, GENE, LUGA, MUNI, LYON, VIEN, MILA""

- 3rd variation: (only new lines)

```

""Note: routers'interfaces have name of the type:

```

- host, meaning it's connected to a host, might have a 0 or 1 to
 - differentiate when there are multiple
- lo, referring to the loopback, it's used as a stable
 - source/destination IP address for routing protocols like OSPF, BGP,
 - and EIGRP.
- port_[router name in all caps], refers to a connection with another
 - router belonging to the same AS, router names can be ZURI, BASE,
 - GENE, LUGA, MUNI, LYON, VIEN, MILA
- ext_[num]_[router name in all caps], refers to a connection with
 - another router belonging to AS specified in num, router names can be
 - ZURI, BASE, GENE, LUGA, MUNI, LYON, VIEN, MILA
- ixp_[num], refers to a connection with an ixp
- dns_[num], ssh, matrix_[num], measurement_[num], refer to services
 - available, you can ignore these
- [router name in all caps]_L2, ignore it, might have .[num] at the end

```

""

```

Appendix B

Hallucinatory responses

Model: Gemini 2.0 Flash

Test: Invert Route-map

1st:

It appears that AS14 can still ping its neighbors, but it is not receiving routes
→ from external ASs. Also, looking at the BGP configurations, AS14 is
→ advertising its own network 14.0.0.0/8, but it's not being accepted by other
→ ASs. The issue seems to be a routing policy that is preventing AS14's routes
→ from being accepted by other ASs.

Model: Gemini 2.0 Flash

Test: Invert Route-map

2nd:

It appears that AS14 is not completely isolated. From the BGP table outputs, all
→ routers in AS14 have routes to other ASs, but they are not the best routes.
From the running config of 14_MILArouter, it seems that the router is configured
→ to peer with AS16. The ping to 179.14.16.16 is successful, so the
→ connectivity is there.

The OSPF neighbor relationship with 14.154.0.1 (14_LUGArouter) is also up.\\
The problem seems to be that AS14 routers are not advertising the routes they
→ learn from other ASs to their iBGP peers.

Looking at the BGP configuration, I see that the `next-hop-self` command is
→ configured for all iBGP neighbors. This means that the routers are changing
→ the next hop attribute to their own IP address when advertising routes to
→ their iBGP peers.

However, the other ASs might not have a route to these internal IP addresses.\\
To fix this, I will try to remove the `next-hop-self` command from the BGP
→ configuration of all AS14 routers.

Unfortunately, I cannot execute commands to modify the configuration.

Model: Gemini 2.0 Flash

Test: Invert Route-map

3rd:

From the `show ip bgp` outputs, we can see that all routers in AS14 have a route
 ↳ to network 14.0.0.0/8, which is the network for AS14. However, they are not
 ↳ receiving routes from other ASs, except for AS11, AS12, AS13, and AS16.\\
 From the `show running-config` output for 14_MUNIRouter, we can see that it has
 ↳ a BGP neighbor relationship with 179.11.14.11, which is a router in AS11. The
 ↳ config also shows route-maps LOCAL_PREF_IN_11 and LOCAL_PREF_OUT_11.\\
 From the `show running-config` output for 14_VIENRouter, we can see that it has
 ↳ a BGP neighbor relationship with 180.81.0.81, which is a router in AS81. The
 ↳ config also shows route-maps IXP_IN_81 and IXP_OUT_81.\\
 The ping tests from 14_MUNIRouter to 179.11.14.11 and from 14_VIENRouter to
 ↳ 180.81.0.81 are successful, indicating that there is basic connectivity
 ↳ between these routers.\\
 The looking glass output from AS1 shows that it is receiving routes from AS11,
 ↳ AS12, AS13, and AS16.\\
 It seems that AS14 is only peering with AS11, AS12, AS13, and AS16. The other ASs
 ↳ are not configured to peer with AS14, or there might be some filtering in
 ↳ place that is preventing the routes from being exchanged.

Model: o4-mini

Test: No Next-Hop-Self

1st:

I think you are not actually missing an underlying TCP session between
 ↳ 12_ZURIRouter and 13_ZURIRouter { rather what you have lost is reachability
 ↳ of all of AS 12's routes inside AS 13. In FRR terms 13_ZURIRouter is
 ↳ learning the 12.0.0.0/8 prefix (and its peers) via IBGP from its
 ↳ IBGP{neighbor 13_BASERouter, but it is **never** installing them into its RIB
 ↳ because the BGP-Next-Hop is not reachable.\\
 \\

Step by step:\\

\\

1) On 13_ZURIRouter you will see the AS 12 prefixes in the BGP table with a
 ↳ next-hop of 179.12.13.12 (that is 12_ZURI's address):

```
13\_ZURI# show ip bgp 12.0.0.0/8
  BGP routing table entry for 12.0.0.0/8
  Paths: (1 available, no best path)
    Not advertised to any peer
    i12      179.12.13.12 from 13.152.0.1 (13.152.0.1)
      Origin IGP, metric 0, localpref 100, valid, internal
```

2) Because that next-hop (179.12.13.12) never made it into OSPF on 13_ZURI (it
 ↳ was never in any IGP { OSPF on 13_ZURI only knows about 13.x.x.x networks),
 ↳ 13_ZURI has **no** route to 179.12.13.12:

```
13\_ZURI# show ip route 179.12.13.12
  % Network not in table
```

- 3) When BGP sees an IBGP-learned prefix whose NEXT-HOP is not reachable, it
 ↳ refuses to pick a best-path and will not install the route in the RIB (see
 ↳ \IBGP-Learned Route Not Getting Installed ... Cause: IBGP Next Hop Not
 ↳ Reachable" in the manual).
- 4) The usual cure is either
- have your IGP carry the EBGp-peering subnet (179.12.13.12/30) so that the
 ↳ next-hop is resolvable, or
 - tell your 13_BASErouter to do a `neighbor 13.152.0.1 next-hop-self` so
 ↳ that it resets the NEXT-HOP to its own loopback (which *is* in 13_ZURI's
 ↳ IGP).

In your case the fix is to add on 13_BASErouter, under the address-family ipv4
 ↳ unicast, one more \next-hop-self" for neighbor 13.152.0.1:

```
router bgp 13
  address-family ipv4 unicast
    neighbor 13.152.0.1 next-hop-self
  exit-address-family
```

Once you do that, 13_ZURI will see all of AS 12's routes with a next-hop of
 ↳ 13_BASE's loopback (13.152.0.1), which *is* in OSPF, and the prefixes will
 ↳ go right into the IP RIB.