ETH zürich

Networked Systems
ETH Zürich — seit 2015

Large language models' ability to reason about router configurations

Bachelor Thesis

Author: Rahul Bhuiyan

Tutor: Yu Chen, Alexander Dietmüller

Supervisor: Prof. Dr. Laurent Vanbever

September 2023 to January 2024

Abstract

In this thesis we attempt to utilise large language models (LLMs) to answer networking questions based on router configurations. We establish an approach to ask these questions and evaluate them on the November 2023 preview of OpenAI's GPT-4 Turbo specifically. Our findings suggest that this LLM performs with varying reliability and requires careful question formulations as well as extensive context to answer correctly.

Contents

1 Introduction 1
- 1.1 Motivation 1
- 1.2 Task and Goals 1
- 1.3 Overview 1

2 Background 2
- 2.1 Large language models 2
  - 2.1.1 GPT 2
- 2.2 Routing Project 3
  - 2.2.1 Structure of the networks 3
  - 2.2.2 Task list 3

3 Design 5
- 3.1 Questions 5
  - 3.1.1 Initial considerations 5
  - 3.1.2 Answer type 6
  - 3.1.3 Question topics 6
- 3.2 Prompting modes 7
- 3.3 Scoring 9
- 3.4 Use of the routing project 9
- 3.5 Question catalogue 10

4 Implementation 12
- 4.1 Concrete question catalogue 12
  - 4.1.1 Basic configuration extraction 12
  - 4.1.2 Network interaction 13
  - 4.1.3 OSPF 14
  - 4.1.4 iBGP 15
  - 4.1.5 Routing policy 16
- 4.2 Experiment framework 19
  - 4.2.1 Challenges while implementing 22

5 Results 24
- 5.1 Basic configuration extraction 24
- 5.2 Network interaction 26
  - 5.2.1 OSPF questions 26

5.2.2 iBGP questions 28
5.2.3 Routing Policy 30
5.3 Costs 32
5.4 Interpretation 32

6 Conclusion **34**
6.1 Outlook 34

References **36**

Chapter 1

Introduction

1.1 Motivation

At the beginning of 2023, we have observed a rise in popularity [4] of large language models due to their impressive text generation capabilities [2][15]. In particular, GPT-3 can generate human-like text to the point of being indistinguishable from real humans in some circumstances[2]. In the context of networking and network learning, one question arose naturally: How capable are such language models of understanding and configuring internet networks? Could we let these machine learning models generate router and network configurations for us? The very first step toward this goal is to understand these models' ability to understand a given network, for example based on a set of configurations of connected routers.

1.2 Task and Goals

Based on these motivations, the central question of the thesis has become: *How well do large language models understand networks and network interactions based on a set of router configurations?*

In order to obtain an answer to this question, we devised the following:

- A set of questions to let the model deduce network interactions information from configurations
- An approach to evaluate the responses from the model to the questions
- Automation processes to obtain a large dataset to make meaningful conclusions about the aforementioned ability

1.3 Overview

In Section 2, we provide background information on large language models and the Communication Networks' routing project [8]. They are central elements that the thesis relies on. Next, we explain conditions, rules, and considerations for the experiment design in Section 3. They are then used to set up the experiment in Section 4. Its results are generated and evaluated in Section 5. Lastly we summarize our insight and present ideas for the future in Sections 6.

Chapter 2

Background

2.1 Large language models

A *Large language model* (LLMs) is a type of machine learning model trained to process and produce natural language. These models are fed large amounts of text, e.g., from articles, papers, forums, etc. for training. They can be used as chatbots, where the user says or asks something in a *prompt* and the LLM processes the prompt and responds to it in natural language. This response by the LLM is referred to as a *completion*.

In November 2022, OpenAI released their (at the time) flagship model GPT-3 [2] in the form of the free-to-use chatbot *ChatGPT*. Its outstanding capabilities to produce natural language and the ability to consider the previous prompts and responses for its next response have made it very popular. Following that popularity, competing LLMs and user facing applications, such as Google's *Bard* [7] and Meta's *Llama* [16], have emerged in the subsequent months. At the time of writing, OpenAI's latest models are GPT-4 for end users and GPT-4 Turbo in preview for developers.

One way in which LLMs are often used is the generation of code and configuration snippets. When provided with a natural language description of some function in a specified programming language, LLMs like GPT-3 can generate said function with varying degrees of accuracy [3]. Despite not achieving total accuracy, they can still provide value to developers in the form of skeletons that can be adapted to the code. In this spirit, GitHub released GitHub Copilot [10], which is powered by a modified version of GPT-3 and can autocomplete code from natural language instructions. Because configurations can be relatively similar to code, it does not seem far off for LLMs to generate those as well. As the focus of this thesis is on the understanding of networks, the ability for LLMs to understand configurations of networking devices is critical.

The amount of information processed is usually measured in *tokens*. Generally, more words correspond to more tokens. But not every character corresponds to a token. It can correspond to single characters for less common words up to entire words.

2.1.1 GPT

OpenAI's Generative Pretrained Transformer (GPT) models have been highly influential in bringing LLMs to the mainstream [4]. And because we mainly evaluate GPT models in this thesis, it makes sense to explain some details of GPT that are used in the experiment design of the in Section 3.

The first model to gain widespread popularity was GPT-3. Its capabilities to write human-like text have garnered considerable attention from the mainstream media [4]. It shows strong performance in many natural language processing (NLP) tasks [2].

The successor is GPT-4. It is estimated to have six times the amount of parameters compared to GPT-3.5 [1].

The latest iteration is the *preview* of GPT-4 Turbo, released in November 2023 [11]. Its prices are currently half to a third of GPT-4's prices.

We interact with GPT using OpenAI's API by inserting our prompt in a JSON-object that has multiple values. The relevant value for this thesis is the **role**. It determines *who* is writing. The main values used in the thesis are `user` for us as the user and `assistant`, where the LLM places its completion.

Costs of calling the API are determined by token consumption. GPT-4 costs \$0.03 per 1000 input tokens and \$0.06 per 1000 output tokens (i.e. the text it generates) [12]. Compared to that, GPT-4 Turbo costs \$0.01 and \$0.02 for the same amount of tokens respectively.

2.2 Routing Project

One part of the course Communication Networks [8] is the routing project, in which groups of students get to set up and operate autonomous systems (ASes) to make up a greater mini-internet. Students start off with a template VM and have to complete a list of tasks, which guide through the various steps of setting up an AS for the given topology in figure 2.1. This includes the setup of direct connections between routers and hosts all the way up to configuring BGP business relationships with neighbouring student operated ASes. Tasks are completed by entering commands into the various routers' and hosts' prompts. The routers run a software called *FRRouting* [5].

The following gives a rundown of the structure, to better understand the design of the experiments.

2.2.1 Structure of the networks

The routing project is a small-scale "internet", where multiple student-managed ASes are connected to each other. Each AS has a fixed structure, as shown in Figure 2.1. They consist of eight routers named BERN, BASE, GENE, etc. Internally, the routers constitute a network running the Open Shortest Path First (OSPF) algorithm. Six of the routers are border routers, which are supposed to connect to border routers of neighbouring ASes or internet exchange points (IXPs) via BGP. These neighbours are at fixed points; Customers of an AS are connected at the bottom two border routers, providers at the top two, and peer and exchange points at GENE and STGA respectively.

2.2.2 Task list

The students work through the following set of tasks:

1. Establish host-router connections
2. Configure OSPF and load balancing for paths between ZURI and LUGA

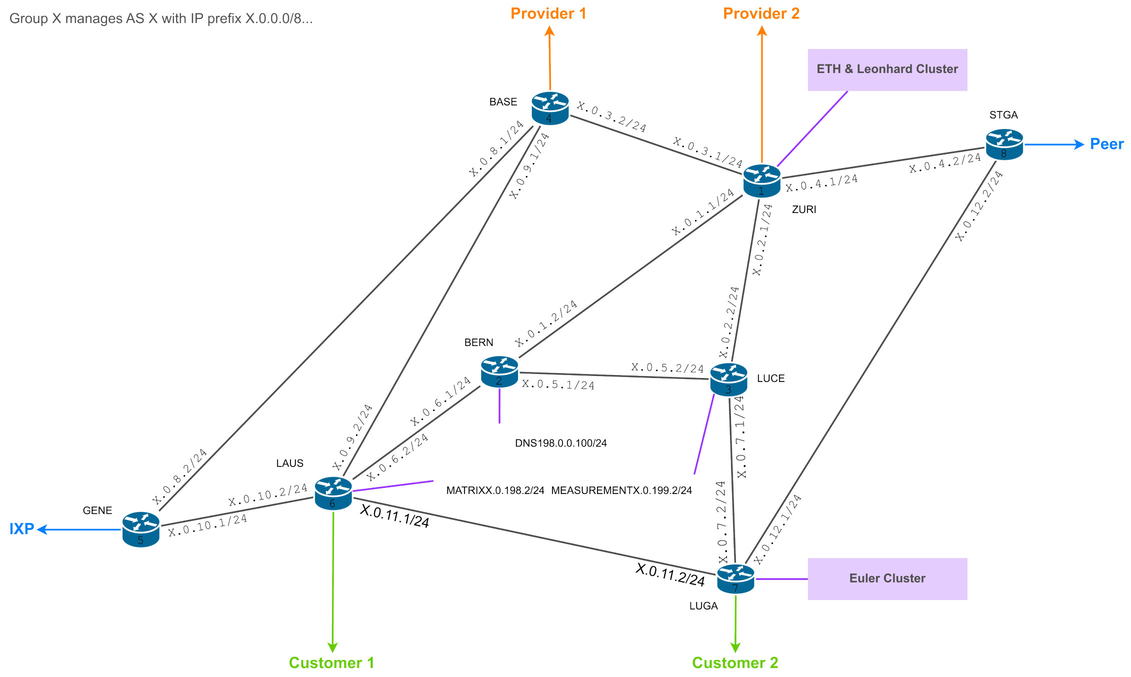

Figure 2.1: Graph showing the topology of an individual AS.

3. Configure IPv6 tunnel between ZURI and LUGA
4. Configure internal BGP (iBGP) full mesh
5. Establish BGP sessions with the neighbours as given by the topology
6. Configure basic no-transit policies for the business relationships using route maps
7. Configure BGP session with connected IXP
8. Optimize established business relationships by setting preferred customer/provider
9. Enable Resource Public Key Infrastructure (RPKI) as a security measure

Chapter 3

Design

Our goal is to determine an LLM's ability to answer increasingly complex networking questions. For that, we need a set of questions for the LLM to answer, evaluation metrics for each question, and an automation around them, as described in Section 1. Let us dive a bit deeper into each of these aspects.

3.1 Questions

The questions for the LLM are the main part of this thesis, as its answers determine its ability to reason about networks. Therefore, we need to consider in great detail many factors when creating the questions. First, we set critical conditions to always consider when looking at other factors. We next examine the effects of a question's formulation on the answer's quality. Here, we further establish the additional information required for each question. At the end of this section, we become more concrete about the expected answer types from a question and the question topics.

3.1.1 Initial considerations

The answers to our questions should fulfil two conditions: Correctness, but more importantly for this thesis, *parsability*. With parsability, we can properly evaluate the answer for correctness in the first place. The LLM's completion might contain additional text around the actual answer to a question, making it potentially difficult to find. Moreover, the response might be in an unforeseen format that cannot be appropriately extracted, and therefore it cannot be meaningfully utilized. Due to these factors, we require the LLM to respond in a well-defined format at a specific place, for example at the end or within the text separated by newlines. This way, we can properly extract answers and batch process them for e.g. evaluation or grading.

Further, a question's formulation directly influences the quality of an answer. For example, a question might be formulated more naturally or more precisely. The more naturally a question is formulated, the more likely it becomes ambiguous. By contrast, questions with excessive information leave little room for the LLM to reason by itself. This would undermine the intended objective of assessing the LLM's capabilities.

We may also include an *example scenario* in the question. It could potentially improve the answer in some ways, but also hurt it in others. On one hand, an unrelated example with a properly formatted answer could guide the LLM in the right direction. The LLM could directly learn from and emulate the example answer, yielding more consistent formatted answers. On the other hand, we may accidentally provide excessive information. The LLM would then consistently answer correctly for the same scenario, i.e. the routing project setup in our case. In a different scenario however, it would potentially neglect information outside what was used in the example. If a factor outside the example influences the answer in the real scenario, it might be ignored. Furthermore, the LLM may be inspired to not write down its reasoning before answering, since the example answer would only consist of said answer. OpenAI's prompt engineering guide [13] suggests for GPT that "giving time to think" improves answer quality.

To summarize, for each question we should have the following information:

- The question itself
- A required format (answer type)
- Optional example set with example data and answer

3.1.2 Answer type

For each question we set one of two *answer types*: Yes-no answers and custom answer formats. This depends on the type of information we want to retrieve. Each answer type has a different purpose and different advantages.

Yes-no answer type questions are predictable and can be easily evaluated, but they lack diversity. Questions with this answer type are about whether a set of conditions holds. Some use cases include: Whether two neighbouring routers have properly set up each other's interfaces or whether a certain AS filters out invalid RPKI routes properly. So it is useful to ask questions reduced to a minimal amount of information. It does mean however that we primarily test its knowledge and ability to check for certain conditions. The LLM's ability to intelligently combine and reorganize information is tested only to a certain extent.

Custom answer formats are more flexible in what questions the can accompany. They complement open-ended questions well with many more possible answers. Examples include seeking the most optimal answer for some scenario, e.g. the best network path or a preferred neighbour. We can also ask it to list multiple things for an answer.

The answer type is requested by a *format instruction.* Its formulation may also influence the result as it is part of the prompt after the question. We may experiment with different instructions; They may or may not contain smaller examples. These are distinguished from the previously mentioned example *scenarios.* Here, we do not provide example data in the form of fictional data. Instead, at most we describe a scenario in natural language, e.g., "For example, if the answer is X, format it like this:..." Compared to the previously mentioned full scenarios, we expect the answer quality to be less influenced.

3.1.3 Question topics

The last factor before building a question set is the questions' content. We distinguish between pure configuration extraction and network interaction. Configuration extraction questions give an idea about the LLMs basic understanding of the given language and syntax. However, interactions in a network, e.g. between routers and ASes, are the main area of interest. Let us describe these in more detail.

Basic configuration extraction

Configuration extraction questions give a partial idea about the LLM's knowledge on networking, the configuration language, and the ability to follow format instructions. First, they partially test its knowledge on networking concepts and protocols. For example, it should know what an IP address or an AS number is and return the values if demanded in natural language. Mostly however, they test the LLM's understanding of the given configuration language and its syntax. As an example from FRRouting, it should know that the syntax `neighbor ip route-map NAME` under the `router bgp` configuration means that the neighbors of this IP are filtered by a route map called `NAME`. Lastly, as these questions are less complex, we can see how well it follows the instruction format. Inconsistencies in this area violate the parsability requirement. So these questions let us experiment to achieve consistently formatted answers.

Network interaction

Ultimately, we want to know how well the LLM understands interactions between routers and between ASes. The main bulk of the questions should therefore cover these interactions. Networking is characterized by the interactions within and between networks. As such, questions covering them give a more substantive idea about its understanding. There are naturally more possibilities for open-ended questions here.

3.2 Prompting modes

As we established in Section 3.1.1, for each question we demand a format instruction and optionally an example set. The next step is to arrange these pieces of information into a prompt for the LLM. Similar to the question's formulation, the way the information is arranged may affect the answer's quality. Therefore, it makes sense to explore different arrangements and their effects on the answer. We refer to these arrangements as different prompting *modes*.

We present four different prompting modes: *Default*, *simulated conversation*, *split up* question and format instructions, and a simple *feedback loop*. Each of these modes tests out a different prompt engineering technique.

Default prompting mode

In this base mode, we provide the LLM with the necessary configurations and directly ask the question with a format instruction. We provide the LLM with all the necessary information at once. This is the simplest case.

> **user**: Provide (set of) configuration(s), question, and format instructions
>
> **assistant**: Provides final answer

Simulated conversation

In this mode, we aim to make the completions and the answer formats more consistent by feeding the LLM an example scenario first. By feeding our own example scenario, including an example answer, the LLM should know how to answer the question for the real data. We start by providing example data and the question in our first `user` prompt. Then for the example answer, we make GPT “think” that it answered the question correctly. This is achieved by inserting our answer into the `assistant`’s message, which normally contains the LLM’s completion. The OpenAI documentation [14] clearly suggests this use case. Lastly, we acknowledge the example answer as correct, provide the real scenario, and ask the question again. The LLM should consider “its” previous answer that we approved, and respond similarly. The result should be more consistent completions with properly formatted answers.

user: Provide example scenario with example data, ask question

assistant: Example answer inserted **by us**

user: Affirm the example answer as correct, provide the real data, and ask again to answer

assistant: Provides final answer

Split up question and format instructions

The idea of this mode is to let GPT reason about a question freely without format constraints first. In the default mode, the LLM would have to provide the properly formatted answer within the same prompt. But GPT might make more mistakes if it has to answer a question immediately as the documentation’s prompt engineering guide suggests [13]. Therefore, we ask the question without any format instructions and let GPT work out its reasoning and response. Then in a second step we ask to summarize its findings from the first completion in the desired answer format.

user: Provide data and question, *without* format instructions

assistant: Reasons and answers in its own way

user: Ask to summarize in format

assistant: Provides final answer

Simple automatic feedback loop

Inspired by *Verified Prompt Programming* in [9], we test here the LLMs ability to revise a wrong answer with simple feedback. This is to address the situation where the LLM oversees information in its initial answer. By prompting it to review it, it might realize what it missed and revise its answer accordingly. We request a new answer to a question, if the current one does not meet our requirements. We continue requesting new answers, until they either meet our requirements or until the hard limit for requests is reached. In the best case, the conversation ends before the loop even starts, which would correspond to the default case.

We distinguish between two different mistakes: Incorrect answer or incorrect format. An incorrect answer is acceptable to a certain extent. In this case, we can draw the conclusion that the LLM’s performance at answering a particular question is insufficient. However, we cannot use an incorrectly formatted answer further. It cannot be processed by programs and thus cannot be evaluated for correctness. We could only comment the LLM’s inability to fully follow instructions, regardless of how well it reasons about that question.

The final score is the score of the last answer, disregarding potentially better previous answers during the loop.

> **user**: Data + question, prompt identical to default mode
>
> **assistant**: Answers in wrong format
>
> **user**: Ask it to abide by the format requirement
>
> **assistant**: Answers in correct format, but wrong content wise
>
> **user**: Ask it to reconsider the information and try to answer again
>
> ...

3.3 Scoring

We must now devise a method to evaluate the LLM's answers. For this purpose we introduce a score. It is either a decimal number between 0 and 1 or the number -1 for improper formatting. However, since the decision on the improper formatting was made late in the process, most of the experiments will not contain this score. Since each question is different, it is logical to create a complementing scoring function for each question. We then return both the score and solution to compare to the given answer.

Each scoring function grades a completion with a score. First, it receives the completion and the data for the solution as input. Then it does three things: Extract the answer from the completion, compute the solution from the data, and compute the score by comparing answer and solution. The closer the answer is to the solution, the closer the score should be to 1, which would be fully correct. Inversely, an incorrect answer receives a score of 0. If no answer can be extracted according to specification, it means the answer format was not respected, resulting in a score of -1.

A yes-no answer can be readily extracted. The scoring becomes (mostly) binary as well, with the answer either matching (score 1) or not matching the solution (score 0). And if there is no "yes" or "no", then the format was not respected, resulting in a score of -1.

For a custom format answer, the difficulty of extraction fully depends on the question and the type of information we want to extract. That makes the computation of the score itself slightly more challenging. We settled for a simple conversion to a Python `set` and a comparison of the solution and answer sets, scoring by how close the answer set is to the solution set. For example, if the answer comes as a list of multiple items, we would need to handle different orders as equivalent. We achieve this by splitting the list into elements of a set. A Python set is *unordered*, i.e., there is no order of elements. Different orders of the correct answers would therefore match the solution equally.

3.4 Use of the routing project

As the final part of our design considerations, we determine the scenario to base our questions around and the data set to ask the questions on. The Communication Networks' routing project is suitable for this task. It manages to be sufficiently small in scale and be relatively comprehensive at the same time. It guides through setting up various parts of an autonomous system, including interactions with different ASes.

We base our questions around the routing project's tasks. As the task list covers many aspects, it presents itself as a suitable base for questions. With the previous considerations in mind, we reformulate the task into viable questions for the LLM.

As alluded to in Section 2, each group operates one AS consisting of eight routers. The main part is the configuration of the routers according to the tasks. In the end, each of these routers' configurations is collected and graded. These batches of eight submitted configurations make up the data for our operations. In other words, the LLM should extract answers from these configurations as well as deduce network interactions based on them.

3.5 Question catalogue

Taking the routing project's tasks as basis, we create our catalogue of questions' *contents*, i.e. what we expect to obtain from each question. Our result is the Table 3.1. The concrete formulations of each question and format requirements are in Section 4.

We have ignored the host-router connections since our main focus is the interactions *between* routers.

Project task	**LLM question content**
(Configuration extraction)	1. Return IP prefix in desired format 2. Return AS location of current router 3. List configured interfaces 4. List configured interfaces with IPv6
Establish host-router connections	
Configure OSPF and load balancing for paths between ZURI and LUGA	5. Existence of path between router A and B 6. Shortest path between A and B based on OSPF
Configure IPv6 tunnel between ZURI and LUGA	
Configure internal BGP (iBGP) full mesh	7. Check iBGP full mesh of our AS 8. Check iBGP all neighbours configured for router X 9. List missing iBGP links in the entire AS 10. List missing links to iBGP neighbours for single router
Establish BGP sessions with the neighbours as given by the topology	
Configure basic no-transit policies for the business relationships using route maps	
Configure BGP session with connected IXP	
Optimize established business relationships by setting preferred customer/provider	11. Pick neighbour (customer or provider) that our AS prefers to receive traffic from 12. List all routed advertisements through our AS from neighbour X to other neighbours 13. List allowed transit paths from customers to providers or vice versa
Enable Resource Public Key Infrastructure (RPKI) as a security measure	14. Check if routes with invalid RPKI value are blocked

Table 3.1: Converting the routing project's task list to questions for the LLM

Chapter 4

Implementation

We now describe the concrete set of questions and how they are tested in the experiment framework. The set of questions is based on the routing project's tasks and will be tested out on certain student groups' configuration sets.

4.1 Concrete question catalogue

The Tables 4.1 and 4.2 show the *configuration extraction* and *network interaction questions* respectively. Additionally, an answer example is given in the fourth column. Each question or question group is explained by its concrete occurrence in the routing project, including relevant FRRouting configuration lines, and the process of formulating them.

We arrived at each of these formulations through iterative experimentation and continuous adjustments within the GPT playground environment. The objective is to run each of these questions through the experiment runner described in Section 4.2 and evaluate their performance on a larger scale. Multiple runs of the same questions will give us an idea on the consistency of GPT's performance.

To simplify the explanation, we assume we are setting up our own AS, as if we participated in the routing project ourselves. Therefore, we often use terms such as *our AS* and *our customers/providers*. In the question formulations, the current / "our" AS is declared by its number explicitly, when needed.

4.1.1 Basic configuration extraction

Table 4.1 contains the questions to test its basic knowledge about FRRouting and the ability to respond in format. Basic networking knowledge is covered by the FRRouting testing. Since we are mainly interested in syntax processing here, configurations of single routers suffice for testing these questions.

The first two questions are not part of the routing project explicitly. Question 1 asks GPT to return the AS number based on a single router's configuration. Question 2 is to return the prefix advertised by the AS. Answers for both are found in an FRRouting configuration on the lines listed

ID	Objective	Question	Example answer
1	Return IP prefix in desired format	*What IP prefix does this router's AS advertise?*	`19.0.0.0/8`
2	Return AS location of current router	*In what AS is this router located?*	`19`
3	List configured interfaces	*What devices are directly connected to our current router?*	`lo : 63.153.0.1/24`
4	List configured interfaces with IPv6	*List all the ipv6 connections that this router has.*	`ZURI-L2.10 : 28:200:10::3/48`

Table 4.1: Concrete formulations for each extraction question.

```
router bgp 63
  ...
  !
  address-family ipv4 unicast
    network 63.0.0.0/8
    ...
  exit-address-family
exit
```

Listing 4.1: BGP configuration lines relevant for retrieving AS number and advertised prefix in FRRouting.

in Listing 4.1. Based on it, GPT should format answers as `63` and `63.0.0.0/8` for the AS number retrieval and advertised prefix question respectively.

Next, we take look at question 3. In the routing project, students need to set up all directly connected interfaces and the loopback interface `lo`. The relevant configurations are structured as in Listing 4.2. Question 4 is simply a variation of this question for IPv6 interfaces specifically. An IPv6 address is declared with `ipv6 address` {`IPv6 address with prefix`} in the `interface` configuration. GPT's task in both is to parse the `interface` configurations and return the answer in the desired format.

4.1.2 Network interaction

Table 4.2 contains every network interaction question. It is split in three parts: OSPF-, iBGP-, and routing policy-related questions.

```
interface NAME
  ip address {ip address}
exit
```

Listing 4.2: Basic FRRouting structure for an interface configuration.

ID	Topic	Question	Example answer
	OSPF		
5	Existence of path between router A and B	*Does a path from router {A} to router {B} exist?*	`Yes` or `No`
6	Shortest path between A and B based on OSPF	*What are the shortest OSPF paths between {A} and {B}? Unless specified, the default OSPF weight is 10.*	`ZURI -> BASE -> GENE`
	iBGP		
7	Check iBGP full-mesh of our AS	*These routers comprise one entire AS. Considering that, does this AS have a full iBGP mesh configured?*	`Yes` or `No`
8	Check iBGP all neighbours configured for router X	*These routers comprise one entire AS. Considering that, has router {router} configured all its bgp neighbors properly?*	`Yes` or `No`
9	List missing iBGP links in the entire AS	*These routers comprise one entire AS. Considering that, has every router configured their bgp neighbors properly? List all the missing bgp sessions, if there are any.*	`ZURI -> LAUS` or `None`
10	List missing iBGP links for a single router	*These routers comprise one entire AS. Considering that, does router {router} have any bgp neighbor missing?*	`ZURI -> LAUS` or `None`
	Routing policy		
11	Pick neighbour (customer or provider) that our AS prefers to receive traffic from	*Here are the routers of the entire autonomous system {this_as}. AS {this_as} is the provider of AS {cust_1} and AS {cust_2}. To which customer is AS {this_as} advertising a more desirable route?*	`AS 19`
12	List all routed advertisements through our AS from neighbour X to other neighbours	*These routers make up the entirety of AS {this_as}. To which neighboring ASes does AS {this_as} propagate AS {neighbor_as}'s advertisements?*	`19 -> 15` or `None`
13	List allowed transit paths from customers to providers or vice versa	*These routers make up the entirety of AS {this_as}. On one side, we have the ASes {neigh_3} and {neigh_4}. On the other side, we have the ASes {neigh_1} and {neigh_2}. Between them sits AS {this_as}. To which of the ASes {neigh_3}/{neigh_4} does AS {this_as} route advertisements from AS {neigh_1}/{neigh_2}?*	`19 -> 15` or `None`
14	RPKI: Check if `invalid` routes are blocked	*These routers comprise one entire AS. Considering that, does this AS accept incoming routers with an 'invalid' RPKI value?*	`Yes` or `No`

Table 4.2: Concrete formulations for each network interaction question, grouped by topic.

4.1.3 OSPF

The first part of network interaction is internal. Open Shortest Path First (OSPF) is the internal gateway protocol (IGP) used in the routing project. It uses standard graph algorithms to determine the shortest path from A to B. The graph of our AS topology is directional, i.e. path weights can be configured in each direction.

Question 5 concerns the basic OSPF setup. The setup involves declaring each connected interface by their subnets listed in Listing 4.3. This includes setting the current router's loopback subnet as a `network` entry and its concrete address in `ospf router-id`.

Thus, to check whether OSPF has been set up properly, checking the existence of a path between

```
router ospf
  ospf router-id 63.151.0.1
  ...
  network 63.0.2.0/24 area 0
  ...
exit
```

Listing 4.3: OSPF configuration in FRRouting.

```
interface NAME
  ip address {ip address}
  ip ospf cost {number}
exit
```

Listing 4.4: Adding OSPF link costs for a certain connecting interface in FRRouting.

any arbitrary routers A and B in the AS suffices. We decided to first formulate it as a yes-no question. For the solution in the routing project, this question should always be answered with yes.

In the routing project, the next step for students is to make sure that only the following paths are load balanced (in both directions):

- `ZURI - BERN - LAUS - LUGA`
- `ZURI - LUCE - LUGA`

This must be done by changing OSPF weights or costs on certain links. In FRRouting, one changes link weights by adding `ip ospf cost {number}` to an interface as seen in Listing 4.4. In FRRouting, the software used in the routing project, each link has an OSPF weight of 10 by default.

Question 6 attempts to check the fastest paths for arbitrary routers A and B. For A=ZURI and B=LUGA (*and* vice-versa), the answer should naturally be the two paths mentioned above. The answer format is the names of the routers on the path (including A and B), separated by spaces and dashes. Here is an example: `ZURI - BERN - LAUS - LUGA`. Hence, the formatted answer for ZURI - LUGA should be the previously mentioned two allowed paths (without the bullet points).

4.1.4 iBGP

A full-mesh in an AS' internal BGP configuration is advisable in many situations; Otherwise, routes may not be propagated to neighbours properly. As such, the routing project requires setting up a full-mesh between the eight routers. The procedure for each router is:

- to add the internal neighbours' loopback address as a BGP neighbour with the same AS number
  (`neighbor {neighbour's loopback} remote-as {internal AS number}`),
- to advertise the current router's loopback as the source address
  (`neighbor {neighbour's loopback} update-source lo`,
  assuming `lo` is the configured loopback interface),

- and to configure `next-hop-self` on border routers

The task of GPT is then to learn each router's loopback IP address and check whether the above are present for each neighbour in each router. In the routing project, the above commands would thus appear seven times in each router's BGP configuration, as each of the eight routers has seven internal neighbours.

While experimenting in the GPT playground, we tried asking the yes-no question whether our AS has configured the iBGP full-mesh properly. This question has been more difficult for GPT than anticipated. Hence, it made sense to dive deeper and arrive at four questions here.

Both answer types are used: The first one, as discussed previously, is a yes-no question, simply checking whether a full-mesh has been configured properly. The second answer type is to list every missing link in a given format. For example, if router ZURI did not configure its session with router LAUS, then GPT should print `ZURI -> LAUS`. Otherwise, it should indicate a full and proper configuration with `None`. Additionally, each question is posed for the entire AS and each router individually, resulting in four questions.

4.1.5 Routing policy

After the setup of BGP and business relationships, optimization of the policy configuration represents a large part of the routing project. Considering its complexity and consequences, it makes further sense to dedicate additional effort in this part. During experimentation, it became promptly clear that due to that complexity, some routing project specific information has to be explained to GPT in the question. As a result, most formulated and tested questions imply the customer/provider/peer structure.

The routing project policies should be as follows:

- Prefer the low latency customer and provider for both outbound and inbound traffic
- Block unprofitable transit paths

Let us focus on the first policy. It requires setting values such as `localpref` for outbound traffic and AS path prepending or multi exit discriminator (MED) values for inbound traffic. They are set in the *route maps*. Route maps are declared and applied to neighbours as illustrated in Listing 4.5.

In question 11, we handle the outgoing advertisements case. The goal for GPT is to identify route-maps applied with the `out` keyword. Then it would find the AS path prepending and MED values to pick the better customer or provider. This question in particular was heavily impacted by the formulation. The initial formulation was (for the customer case): "Which customer does our AS prefer to get traffic from?"

GPT's answer was structured as seen in Listing 4.6. It did manage to output the correct answer in the correct format immediately. However, it understood it as an outbound question, arguing with the outbound route maps. With numerous subsequent formulations, it generally fell back to arguing with outbound route maps. In the end, the question seems to have been posed *too naturally*. The final reformulation to "To which customer is our AS advertising a more desirable route?" has brought more consistent results. Only with this precise formulation does GPT realize which information it has to work with.

```
...
router bgp 48
  neighbor ...
  ...
  !
  address-family ipv4 unicast
    network 48.0.0.0/8
    ...
    neighbor 12.34.56.78 route-map receive_ads in
    neighbor 12.34.56.78 route-map propagate_ads out
  exit-address-family
exit
!
...
route-map receive_ads permit 10
  set local-preference 500
exit
!
route-map propagate_ads permit 10
  set as-path prepend 48 48
exit
!
...
```

Listing 4.5: Creating and applying route maps in FRRouting. The keyword `permit` filters routes matching these conditions as allowed. All other routes are implicitly denied.

```

AS {our AS} prefers to get traffic from the customer with the highest local preference
     set in the BGP configuration. Local preference is a BGP attribute that determines
    the preferred exit path for traffic leaving an autonomous system.

From the provided configurations, we can see that router LAUS (AS {our AS}) has set a
    local-preference of 300 for routes received from its customer AS {our customer 1}:

route-map {outbound traffic}
...

Similarly, router LUGA (AS {our AS}) has set a local-preference of 301 for routes
    received from its customer AS {our customer 2}:

route-map {outbound traffic}
...
...

Final answer:
AS {our customer 2}
```

Listing 4.6: Sample answer for question 11 (early formulation). The answer is correct, but the reasoning is wrong.

```
# Route map at border router with peer, receiving from peer
route-map receive_adverts permit 10
  set local-preference...
  set community 58:1
  ....
exit
...
# Route map at border router with provider, propagating to provider
route-map propagate_adverts deny 10
  match community 58:1
exit
```

Listing 4.7: Usage of community values. In this example, routes originating at the peer will not be propagated to the provider due to the `deny` keyword.

Due to time constraints, the outbound traffic version of this question was left out.

Questions 12 and 13 concern the no-transit policies. In the routing project (and in general), the transit paths should abide by the *Gao-Rexford conditions* [6] and only allow financially profitable paths. For example, allowing transit from a peer to a provider would make our AS lose money; Our AS would have no costs on the peer connection, but it would have to pay the provider to forward traffic, resulting in an overall loss. In contrast, in the transit from customer to provider, our AS would still pay our provider to forward traffic, but it would generate revenue from receiving the customer's traffic.

By the Gao-Rexford conditions, the following paths should be blocked:

- peer → provider
- provider → peer
- peer → peer

Such transit policies are configured by a route map's `permit` or `deny` keywords and community values. In Listing 4.5, there are only `permit` route maps. This can be enough as all routes not allowed by a `permit` route map are implicitly denied. Community values can be used to tag routes when passed between routers within our AS. These tags can be matched for in route-maps for filtering. This can be used to stop routes received from peers from being propagated to providers: The border router at the peer sets a community value for received routes in a route map. The router bordering the provider then configures its route maps to not propagate it further. With this, the provider → peer traffic is blocked. Listing 4.7 shows the configuration in FRRouting.

In question 12, we choose a neighbour and ask which transit paths originating from this neighbour through our AS are allowed. GPT thus needs to check the right route maps based on community values. With this information, it will then omit blocked paths, printing only allowed ones. The format should be `neighbour X -> other neighbour`, e.g. `56 -> 59` for X = AS 56 and the other neighbour being AS 59. We decided that it should not print our AS (i.e. not `56 -> 58 -> 59` if our AS is 58), since it is implied.

Question 13 works with the routing project's structure more closely. The goal is to check whether

```
route-map receive_adverts permit 10
  match rpki valid
exit
```

Listing 4.8: Setting RPKI in route maps. The following map allows for all routes with a `valid` RPKI value.

```
{
    "question" : "...",
    "expected_format" : "...",
    "example_data" : "...",
    "example_answer" : "..."
}
```

Listing 4.9: The question object.

the transit between our customers and providers is ensured. We list every such allowed path between the two "groups" in the same format as in the second policy question.

Lastly, question 14 serves to improve coverage of the routing project's tasks. We are asking whether routes with an `invalid` RPKI value are allowed in the AS. Originally, this question was intended as a basic configuration extraction question. However, the realization occurred that it requires the knowledge about all router's route maps.

RPKI is configured by setting the cache server and `match` rules in route maps. One can filter routes based on their rpki values `valid`, `invalid`, and `notfound`. An example is shown in Listing 4.8.

Here we expect GPT to simply check if there is no `rpki invalid` values in any `permit` route map.

4.2 Experiment framework

The following is a brief overview of how the experiments are run. The experiment framework is implemented in distinct parts of Python code. It consists of the question object, scoring functions, the main experiment runner, experiment functions, and helpers. The *question object* groups question, format instructions, and an optional example together. They are complemented by the *scoring functions* that compute a solution and use it to evaluate GPT's answer. *Experiment functions* make use of them through the *main experiment runner*. It itself uses many *helpers* to concretely prompt, score, and export the results. Each of the question objects receives a unique identifier, that are used in the experiment and scoring functions' names as well. Figure 4.1 visualizes this code flow.

Questions object

Each question is accompanied by an expected format and optionally an example. We implemented this in code by the structure listed in listing 4.9. This format facilitates the generation of the prompt for the LLM.

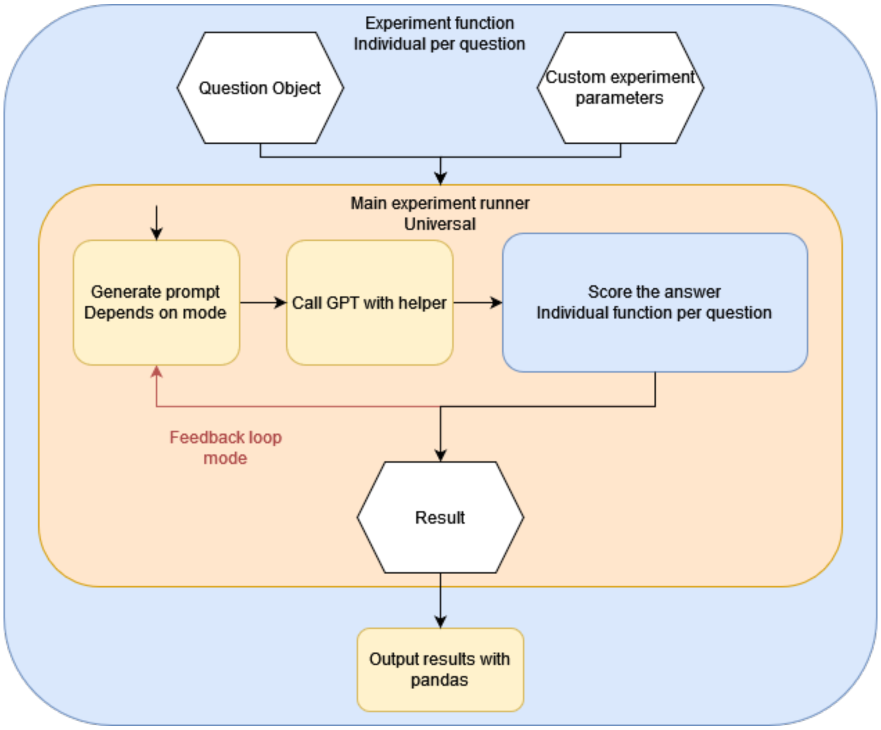

Figure 4.1: Diagram of the basic experiment code flow.

Composing the prompt

This object structure is used in the function `config_create_prompt_default` to generate the concrete prompts. This generation is performed in two steps: Putting together the relevant configuration snippets and making the prompts according to the mode. The configuration files are large and make for large prompts. As explained in more detail in 4.2.1, this causes problems. As such, we collect relevant configuration parts with a configuration extraction helper (called `config_extract_multi`). Then the prompt composing helper (called `config_create_prompt_default`) generates the desired prompt. It does not take in any concrete mode. Instead, it takes the flags `include_example` and `include_format`. They are used by the main experiment runner to individually generate prompts by mode.

Now we provide the concrete formulations used in the composition. We illustrate them using the question: "In what AS is this router located?". Further, we use the following format instructions: "List only the AS number. For example, if the router is located in AS 53, your answer should be: 53"

The program would make the following prompts:

Default prompting mode:

> **user** "You will answer questions about the following FRRouting configurations: (Real Data)... *In what AS is this router located? List only the AS number. For example, if the router is located in AS 53, your answer should be: 53*"
>
> **assistant** Provides final answer

Split question and format instructions:

user “You will answer questions about the following FRRouting configurations: (Real Data)... *In what AS is this router located?*”

assistant Provides intermediate answer

user “Now summarize your findings. *List only the AS number. For example, if the router is located in AS 53, your answer should be: 53*”

Simulated conversation:

user “You are going to answer questions about FRRouting configurations. Given this example set of config snippets: (Example data)... *In what AS is this router located? List only the AS number. For example, if the router is located in AS 53, your answer should be: 53*”

assistant Our inserted example answer

user “This is correct. Now answer the same question, but for this set of configurations for a network unrelated to the previous example: (Real Data)... Remember to answer in the same format as before.”

Basic feedback loop:

user “You will answer questions about the following FRRouting configurations: (Real Data)... *In what AS is this router located? List only the AS number. For example, if the router is located in AS 53, your answer should be: 53*”

assistant Provides wrongly formatted answer

user “You got the format wrong. Please summarize your answer in the format requested before.”

assistant Provides correctly formatted, but wrong answer

user “It seems like you didn’t get the answer right. Review the given information again and respond in the format requested before.”

assistant Provides correct answer

The program inserts exactly each of the non-highlighted parts in this manner.

Scoring functions

Next are the scoring functions. They follow a consistent function signature and structure shown in listing 4.10. We extract the solution from the configurations and each AS’ grading reports. We heavily rely on string manipulation here, as there is no easy to use FRRouting parsing library, meaning we are performing this task manually.

Experiment runner

The main experiment runner (called `run_experiment_multi_configs`) puts everything together. It calls the other parts to generate prompts, make API requests, and score the responded completion. It consists of the main experiment runner and wrapping experiment functions of each question. The experiment functions of the questions are wrappers around this runner. They access the data and contain adjustable parameters for each question. This can be used to ask the question about, for example, a different router or a different neighbouring AS.

```
def answer_score(configs_list, completion, args):
    # Extract answer from completion
    ...
    answer = find(...,completion,...)
    ...
    # Compute solution from configs_list
    ...
    solution = find(...,configs_list,...)
    ...
    # Bring answer and solution into same python data structure
    ...
    solution_set = set(...,solution_str,...)
    answer_set = set(...,answer,...)
    ...
    # Compute solution from degree of overlap
    score = overlap(solution_set,answer_set)

    return score, solution
```

Listing 4.10: General structure of each scoring function.

Lastly, using the well-known Python library *pandas*, we collect all results in a table and export it. The function for this in the code is called `pandas_save`.

4.2.1 Challenges while implementing

Pricing of the API was an initial concern. It therefore made sense to investigate how to keep costs low. Each prompt costs a certain amount of input and output tokens. Before the preliminary release of GPT-4 Turbo, there was only plain GPT-4. Calls to it are quite costly, at $0.03/1000 input tokens and $0.06/1000 output tokens. To get an idea for how expensive this is, we ran a set of seven early configuration extraction questions on 32 router configurations, i.e. about 224 requests. In every request we sent the router's entire configuration and one of the questions. This entire run cost more than $5 to execute. Thus, it made sense to reduce the tokens per request.

The best way was to reduce the configuration part to a minimum. The extraction question about listing interfaces did not need any BGP configurations or route maps. In this case, the configuration part of the prompt was reduced to less than half.

As the field of LLM's is still emerging, changes to services are bound to happen. OpenAI released a preview of GPT-4 Turbo in November 2023, improving GPT in various ways. GPT-4 Turbo accepts a larger amount of context than GPT-4, meaning prompts and history can be larger. It is reasonable to conduct our testing on the latest LLM, to draw the most current conclusions. Since we are working with OpenAI's library, changing the model is a matter of changing a single parameter. It is possible to compare the different versions side-by-side. However, due to time constraints, assessing a single model's performance was the most reasonable course of action.

Rate limitations have proved to be a heavy constraint. Therefore, some code handling rate limit errors had to be added to run experiments in batches unattended. For a public API with limited computing power, rate limits are standard practice. In GPT-4 Turbo's case, that limit has been tight, as it is still in preview. As of writing, the current daily limit is 500,000 tokens. Batch testing all the questions multiple times from previous attempts has exhausted this limit rapidly. Using the tenacity library, we were able to request the API multiple times unattended.

As the limit was tight, processing all experiments would take a long time. It was therefore reasonable to leave the code running unattended on a virtual machine.

Chapter 5

Results

We ran the experiment setup discussed in the previous chapter. We specifically used the November 2023 preview of GPT-4 Turbo. Each question was run with various settings and variations as we saw fit. Each variation was run three times.

We use two groups' configuration sets. We refer to them as *Group 1* and *Group 2*. Group 1 solved all tasks in the routing project correctly. Group 2 has erroneously configured their transit policies. They did not set up transit to their customers from any neighbour and did not prefer their low delay provider and customer. We test mainly on Group 1's configurations and additionally on Group 2 where their configurations contain mistakes, for comparison. Results in the tables are for Group 1, unless specified.

We shorten the names of prompting modes as such:

Default The default prompting mode with question and format instructions together.

Split Split question and format instructions in two steps

Example Simulated conversation with example data and answer

Feedback Feedback loop if score is too low

GPT has a multitude of custom parameters. However, for this experiment, they are all unmodified.

5.1 Basic configuration extraction

Each question in this category was run in the *Default*, *Split*, and *Example* modes for each of Group 1's router configurations. With three modes, eight routers, and 3 runs per combination, each question was run 72 times in some variation, making for 216 total runs.

With these simpler questions, we already see a pattern: Listing a single thing is not problematic for GPT. However, in question 3, the low scores in the *Default* mode indicate the need for good formulations and different approaches to prompting.

Question 1: Advertised prefix

Question formulation "In what AS is this router located?"

Format instruction "List only the AS number. For example, if the router is located in AS 53, your answer should be:
53"

GPT always answers this question correctly, but in *Default* mode, it never follows the format fully; It always generates a full sentence around it, for example: "The router's AS (Autonomous System) advertised IP prefix is (...).0.0.0/8." With the *Split* and *Example* modes, it becomes clear to GPT that it should leave out any sentence in its final answer.

Mode	**Average score**	**Median score**
Default	0.5	0.5
Split	1.0	1.0
Example	1.0	1.0

Table 5.1: Results for question 1.

Question 2: AS location

Question "In what AS is this router located?"

Format (detailed) "List only the AS number. For example, if the router is located in AS 53, your answer should be:
53"

We can confidently say that GPT answers this question consistently correctly. It is not very complex and the instructions are clear. As such, the high score on this question is not unexpected. There was a single outlier out of the 72 runs where GPT added additional text around the answer number. Results are shown in Table 5.2.

Mode	**Average score**	**Median score**
Default	0.979	1.0
Split	1.0	1.0
Example	1.0	1.0

Table 5.2: Results for question 2.

Question 3 & 4: List interfaces

Question formulation [IPv4] "What devices are directly connected to our current router?"

Format instruction [IPv4] "List them in a format like this (unrelated example):
lo : 153.34.245.0/17
port_WIEN : 91.23.0.21/8"

Question formulation [IPv6] "List all the ipv6 connections that this router has."

Format instruction [IPv6] “List them in a format like this (unrelated example):
port_ROME : 28:200:20::3/48
port_WIEN : 54:100:15::3/48
If there are none, answer with “None”.”

An immediate insight is that the formulation of the IPv4 question is ambiguous, which is obvious in hindsight. We have not specified in the format instructions explicitly that it should only reply with IPv4 addresses. In many runs GPT seems to list the interfaces’ IP addresses as the prefixes, as seen in Figure 5.1. Technically under the utilized formulation, the loopback would not be considered a “directly connected device”. The data of the *Example* mode might have overridden the flaw of the question formulation, seeing the score.

The IPv6 question is quite more precise. It seems to have helped immensely with the score as a result, next to the fact that only the routers ZURI and LUGA have IPv6 connections.

Question 3: IPv4

Mode	**Average score**	**Median score**
Default	0.297	0.0
Split	0.504	0.646
Example	0.969	1.0

Question 5: IPv6

Mode	**Average score**	**Median score**
Default	0.813	1.0
Split	0.979	1.0
Example	1.0	1.0

Table 5.3: Results for questions 3 and 4. Question 4 scores noticeably higher.

```
 # GPT completion sample         # Solution
host :  (...).102.0.0/24        host :  (...).102.0.2/24
lo :  (...).152.0.0/24          lo :  (...).152.0.1/2
port_LAUS : (...).0.6.0/24      port_LAUS : (...).0.6.1/24
port_LUCE : (...).0.5.0/24      port_LUCE : (...).0.5.1/24
port_ZURI : (...).0.1.0/24      port_ZURI : (...).0.1.2/24
```

Figure 5.1: Sample answer and solution for question 3.

5.2 Network interaction

Tables from 5.4 onward contain the results to each question. Every score is an average of three runs, unless noted otherwise.

5.2.1 OSPF questions

Question 5: Existence of OSPF path

Question formulation “Does a path from router {A} to router {B} exist?”

Formulation 2 “Is a host connected to router {A} able to ping a host connected to router {B}?”

Format instruction “Make sure that the last line of your response contains a plain “Yes” or “No” as the answer to the question.”

Format instruction 2 “’Answer with either “Yes” or “No”.”

ID	Question and settings	Mode	Average score	Answer type
5	Existence of path: Formulation 1, Format Instruction 1	Default Split	1.0 1.0	Yes-no
	Existence of path: Formulation 2, Format Instruction 1	Default Split	1.0 0.92	
	Existence of path: Formulation 1, Format Instruction 2	Default Split	0.667 1.0	
6	Shortest OSPF path: Format Instruction 1	Default Split Feedback	0.417 0.75 0.75	List in format
	Shortest OSPF path: Format Instruction 2	Default Split Feedback	0.583 0.75 0.417	
	Shortest OSPF path: Format Instruction 3	Default Split Feedback	0.319 0.833 0.75	

Table 5.4: Results for the OSPF questions.

This question was run for two router pairs: `STGA - GENE` and `ZURI - LUGA`. The former is the longest path in the network and the latter is the path to be load balanced. As such, each score here is the average of six runs.

Somewhat expectedly, the format instruction 2 lowers the score, at least in the *Default* mode. The plain instructions given often make GPT add a dot after its “Yes”/“No” answer. This is a formatting error, since it could have been embedded in a sentence. Other than that, it can answer this yes-no question with a high degree of accuracy.

Question 6: OSPF shortest paths

Question formulation “What are the shortest OSPF paths between {A} and {B}? Unless specified, the default OSPF weight is 10.”

Format instruction “In the last lines of your response, list the paths like this (unrelated example):
STGA - ZURI - BASE
STGA - ZURI - BERN - BASE”

Format instruction 2 “Make sure to summarize the paths at the end of your response in a plain list like this (unrelated example):
ZURI - LUCE - LUGA
ZURI - STGA - LUGA”

Format instruction 3
“Without explanation, list all these paths like this (unrelated example):
ZURI - LUCE - LUGA
ZURI - STGA - LUGA”

GPT can solve this question most of the time. Many times it scored 1.0, but scored 0.0 a few times, because it did not fully follow the format, as seen in Figure 5.2. Listing multiple items continues to make answers less consistent.

```
# Completion                  # Solution
[...]                         ZURI - LUCE - LUGA
1.  ZURI - LUCE - LUGA        ZURI - BERN - LAUS - LUGA
(Cost:  20)
2.  ZURI - BERN - LAUS -
LUGA (Cost:  20)
```

Figure 5.2: Sample answer and solution for question 6.

Due to a problem with this question's scoring function, we retroactively scored the same answers after some adjustments to the function. The *Default*, *Split*, and *Example* mode answers are not affected. For the *Feedback* mode, answers during the loop were scored with the function before the adjustments. As such, only the final answer is scored with the current version. If it used the new version during the entire loop, it would likely have been stopped earlier since it would have found a correct answer earlier.

5.2.2 iBGP questions

ID	Question and settings	Mode	Average score	Answer type
7	Check iBGP full-mesh of our AS: Format instruction 1	Default	0.0	Yes-no
		Split	1.0	
		Example	0.333	
		Feedback	1.0	
	Check iBGP full-mesh of our AS: Format instruction 2	Default	1.0	
		Split	0.5	
		Example	0.0	
		Feedback	0.167	
8	Check iBGP all neighbours configured for routers STGA, GENE, BERN	Default	0.778	Yes-no
		Split	0.889	
9	List missing iBGP links in the entire AS	Default	0.333	List in format
		Split	0.6	
10	List missing iBGP links each router	Default	0.958	List in format
		Split	0.958	

Table 5.5: Results for the iBGP questions. Scores are averaged by mode.

Question 7: Check if AS has full-mesh

Question formulation "These routers comprise one entire AS. Considering that, does this AS have a full iBGP mesh configured?"

Format instruction "Answer with either "Yes" or "No"."

Format instruction 2 "Make sure that the last line of your response contains either "Yes" or "No" as your answer to the question."

Despite being in the yes-no answer format, question 7 has been difficult to make GPT answer correctly. GPT has to examine eight large configuration snippets to determine the answer. Despite this, some scores seem significantly better than others.

With the first instruction, GPT only answers with the actual requested answer, i.e. it does not "think" or reason through the configurations before answering. This likely lowers the answer quality. As such, it scores poorly in the *Default* mode. The *Split* mode by contrast is likely the optimal formulation for this format instruction: In the first step, GPT can "think", then it has to answer directly according to this simple instruction. The *Feedback* mode's high score is likely because it can answer with the opposite here if it answered wrong.

Compared to this, the second instruction scores better in the *Default* mode. In the answers, GPT generates its reasoning, as anticipated. In the *Split* mode however, the answer is never put separately in the last line, in all three runs. The word "contains" in the instructions might make it ambiguous. The *Example* here scores poorly as well. This could point to the provided example scenario being confusing. It conflicts with the instruction, since we expect GPT to "think".

Question 8: Check if router X has full-mesh

> **Question formulation** "These routers comprise one entire AS. Considering that, does router {router} have any bgp neighbor missing?"
>
> **Format instruction** "Make sure that the last line of your response contains a plain "Yes" or "No" as the answer to the question."

GPT answers correctly most of the time, with the few wrong answers impacting the score greatly. It manages to answer correctly more often with individual routers. As we ran this question for three routers, the scores here are averages of 9 runs each.

Question 9: List missing links of AS

> **Question formulation** "These routers comprise one entire AS. Considering that, has every router configured their bgp neighbors properly? List all the missing bgp sessions, if there are any."
>
> **Format instruction** "In the last lines of your response, summarize your answer in this exact format:
> `ZURI -> LAUS`
> In this example answer, ZURI did not set up its iBGP session with LAUS properly. This example answer is unrelated to the current scenario of the configurations.
> If there is no link missing, then make sure to put a "None" in the last line of your response instead."

This is a more complicated question than the yes-no pendant. The scores are similarly low. This could be an indication that the entire AS is too large as a scope.

Question 10: List missing links of router X

> **Question formulation** "These routers comprise one entire AS. Considering that, does router router have any bgp neighbor missing?"
>
> **Format instruction** "In the last lines of your response, summarize your answer in this exact format:
> `ZURI -> LAUS`
> In this example answer, ZURI did not set up its iBGP session with LAUS properly. This example answer is unrelated to the current scenario of the configurations.
> If there is no link missing, then make sure to put a "None" in the last line of your response instead."

These results show high consistency and that GPT will perform better when the question has a smaller scope. This question has been run for all of Group 1's eight routers, three times in each mode. Thus, the results are averages of 24 total runs.

Generally, the AS-encompassing questions tend to perform poorly compared to their single router pendants. Additionally, the *Split* mode tends to perform noticeably better than the other modes. This can be explained by the fact that GPT is "forced" to reason before formatting its answer. It does not however make GPT answer questions perfectly.

5.2.3 Routing Policy

ID	Question and settings	Mode	Average score	Answer type
11	Pick preferred customer inbound traffic, Group 1	Default	1.0	Answer in format
		Split	1.0	
	Pick preferred provider inbound traffic, Group 1	Default	1.0	
		Split	1.0	
	Pick preferred customer inbound traffic, Group 2	Default	0.0	
		Split	0.0	
	Pick preferred provider inbound traffic, Group 2	Default	0.333	
		Split	0.0	
12	List advert. paths from neighbour X to other neighbours, Group 1	Default	0.0	List in format
		Split	0.267	
	List advert. paths from neighbour X to other neighbours, Group 2	Default	0.667	
		Split	0.333	
13	List allowed transit paths from customers to providers, Group 1	Default	0.667	List in format
		Split	1.0	
	List allowed transit paths from providers to customers, Group 1	Default	0.0	
		Split	0.333	
	List allowed transit paths from customers to providers, Group 2	Default	0.833	
		Split	0.833	
	List allowed transit paths from providers to customers, Group 2	Default	0.0	
		Split	0.667	
14	Check if routes with invalid RPKI value are blocked, Format instruction 1	Default	1.0	Yes-no
		Split	0.667	
	Check if routes with invalid RPKI value are blocked, Format instruction 2	Default	0.667	
		Split	0.333	

Table 5.6: Results for the routing policy questions. Scores are averaged by mode.

Question 11: Pick preferred neighbour (inbound traffic)

Question formulation "Here are the routers of the entire autonomous system {this_as}. AS {this_as} is the customer/provider of AS {prov_1}/{cust_1} and AS {prov_2}/{cust_2}. To which provider/customer is AS {this_as} advertising a more desirable route?"

Format instruction "Provide your final answer in the last line of your response in the format: AS {Number} example (unrelated to the configs in question): AS 19"

Results for Group 1 are significantly better than for Group 2. Group 1's configurations contain AS path prependings. GPT recognizes prepended paths, if they are present. This is not the case for Group 2, as they misconfigured their route maps. GPT falls back to the `localpref` values of the *outbound traffic* route maps in many answers. In two out of the six answers, it chose Group 2's AS number. One can conclude that if the route maps do not contain any AS prepending, GPT answers arbitrarily.

Question 12: List transit paths from neighbour X

> **Question formulation** "These routers make up the entirety of AS {this_as}. To which neighboring ASes does AS {this_as} propagate AS {neighbor_as}'s advertisements?"
>
> **Format instruction** "List these allowed paths like this in the last lines of your response, excluding the middle AS: `X -> Y`
> This example path would mean that advertisements from AS X would be propagated to AS Y. This example answer is unrelated to the current scenario.
> If no such path is allowed, end your answer with a "None" in your last line."

GPT performed poorly in this question. It often used the inverse order for the format, as illustrated in the Figure 5.3. In the question formulation, the AS numbers were replaced with X and Y, since the actual numbers used in this instruction corresponded to neighbours of Group 1. As such, GPT might have taken the example directly without processing the explanation. The scores for Group 2 were better, but still not fully passable. The example in the format instruction was the same, i.e. it did not correspond to any of Group 2's neighbours.

```
 # GPT completion sample     # Solution
[...]                       {Neigh X} -> {Cust 1}
{IXP} -> {Neigh X}          {Neigh X} -> {Cust 2}
{Cust 1} -> {Neigh X}
{Cust 2} -> {Neigh X}
{Prov 1} -> {Neigh X}
{Prov 2} -> {Neigh X}
```

Figure 5.3: Sample answer and solution for the advertised paths from X question.

Question 13: List transit paths between customers and providers

> **Question formulation** "These routers make up the entirety of AS {this_as}. On one side, we have the ASes {neigh_3} and {neigh_4}. On the other side, we have the ASes {neigh_1} and {neigh_2}. Between them sits AS {this_as}. To which of the ASes {neigh_3}/{neigh_4} does AS {this_as} route advertisements from AS {neigh_1}/{neigh_2}"
>
> **Format instruction** "List these allowed paths like this in the last lines of your response, excluding the middle AS: `X -> Y`
> This example path would mean that advertisements from AS X would be propagated to AS Y. This example answer is unrelated to the current scenario.
> If no such path is allowed, end your answer with a "None" in your last line."

In this question, the customer to provider transit worked better than the inverse. For Group 1, the reason could be traced back to the example numbers matching some of their neighbours. One answer is simply wrong, but for the other two, the order of numbers is swapped, similar to the previous question (Figure 5.3). For Group 2 it is different. They did not set up the provider to customer transit properly. As seen in Figure 5.4, GPT confidently presented a wrong answer.

```
 # GPT completion sample    # Solution
[...]                      None
{Prov 1} -> {Cust 1}
{Prov 1} -> {Cust 2}
{Prov 2} -> {Cust 1}
{Prov 2} -> {Cust 2}
```

Figure 5.4: Sample answer and solution for the provider -> customer transit question.

Question 14: RPKI question

Question formulation “These routers comprise one entire AS. Considering that, does this AS accept incoming routers with an ‘invalid‘ RPKI value?”

Format instruction “Make sure that the last line of your response contains a plain “Yes” or “No” as the answer to the question.”

Format instruction 2 “’Answer with either “Yes” or “No”.”

This question is not difficult and GPT manages to answer it satisfyingly. The only error in this yes-no question is unnecessary periods after the “Yes” or “No”, especially with format instruction 2. This is consistent with the results for other yes-no questions.

Scores for format instruction 1 have been modified to reflect its contents more accurately.

5.3 Costs

Taking into account the price reduction with GPT-4 Turbo and how we reduced the prompt size, we arrived at the following total costs, excluding all previous experimentations:

- Total tokens fed (input): 2,554,142
- Total tokens generated by GPT (output): 263,438
- Total costs in US $: 33.43

We managed to significantly reduce the token count for each prompt by extracting only necessary snippets. For example, the routing policy questions require close to 4500 tokens. That is for the snippets for each of the eight configurations. For reference, the full configurations of Group 1 amount to 6500 tokens in total. This corresponds to a near 50 % reduction for this case.

Considering that many of the experiments were not run more than three times, the costs are still significant for any future large-scale operation.

5.4 Interpretation

The wrong answers show that it is not easy to make GPT reason with networks in the context of the routing project. Many of the results underline the difficulty in finding good question formulations.For example, in some routing policy questions, we provided an example in the format instruction that was too close to the Group's AS numbers. Even after explaining the example and explicitly stating that it is unrelated, GPT answered in a wrong format. It shows that for more complex questions, we need to be more and more precise, ultimately giving away excessive information exclusive to the given network context.

GPT tends to answer more consistently for smaller scoped questions. The configuration extraction questions are largely answered without issue; Only question 3 had low scores due to a too natural formulation.The iBGP questions for the individual routers perform noticeably better than the AS-encompassing counterparts as an example. This can also be interpreted as a limit to its reasoning capabilities.

Letting GPT "think" or reason before answering improves the answer quality. The *Split* mode facilitates this since GPT can answer in its own way before we demand it in format. This leads to lengthy texts of reasoning before we reply with the format instruction. As such, it outperforms other modes in many cases. Further, the *Example* mode performs poorly, where employed. Only with the configuration extraction questions it works as expected. With more complex questions this strategy fails, perhaps because GPT is forced to answer without working out the answer. In some cases it overrides the formulation instruction's goal to make it reason.

Lastly, GPT has problems with following instructions fully. With yes-no questions, it tends to put some text around or a period after it. This is not entirely consistent with the format instructions. Further, listing multiple items in format continues to be a challenge for GPT. It might ignore the instructions more readily, as seen in the OSPF shortest paths question (Figure 5.2).

Chapter 6

Conclusion

We have asked GPT-4 Turbo a variety of questions with different settings. These included different approaches to prompting (modes) and different format instructions. We find that approaches, where GPT can reason and answer in its own way before putting it in format, tend to yield better results. However, it is not always consistent; Perhaps due to GPT's nondeterministic nature, the reliability of answers varies greatly even within a few runs. It tends to not respect the format instruction consistently and reasoning can often vary as well. All this means that much work is required to make it answer networking questions consistently in a parsable format. Since it requires such precision for more complex questions, we conclude that GPT-4 Turbo cannot understand networks well without the user providing extensive context around the router configurations.

6.1 Outlook

If one were to utilize this thesis and its materials (codebase etc.) as a starting point, they would have to repair flaws in the scoring functions. As of right now, some results needed to be corrected manually, since the format instructions do not fully match the functions any more. As such, they will need to be updated, particularly the yes-no questions. Furthermore, most of them do not yet handle wrong formats with a score of -1. This would have to be updated as well. After changing and improving the codebase, we could explore other ideas.

One idea would be to explore more prompting approaches. First, we could rework the prompting modes such that each of the four current modes can be combined arbitrarily. For example, the feedback loop could start as soon as the score is too low, regardless of mode. In general, the feedback loop could be improved. We could introduce more complex feedback messages, e.g. by varying the formulation every time or repeating the format instructions. In its current state, the loop is repetitive and variance could help retrieve correct answers more quickly. Additionally, the simulated mode could be extended by providing more examples with more variance. We could even create wrong answers and then explain why they are wrong. This way, GPT would be more trained than in our experiments.

Next, we could introduce a way to evaluate reasoning. In the incoming traffic preference question, initial formulations resulted in correct answers, but with the wrong reasoning. Since we had few runs per variation in this thesis, it was simple to evaluate the reasoning and correct the formulation.

However, if in the future, a variation is run many more times, this might become difficult. Therefore, we could make GPT output a reason, e.g. as a keyword. The scoring would take it into account and we could see it in the output, letting us make more informed decisions.

One could also make the LLM choice more modular in the code. Other LLMs could yield equal results without us knowing. Right now, the functions and modes are directly bound to GPT's prompt structure. To evaluate other models, one would have to change this part in particular. This way we could compare the performance of different LLM's.

It might be necessary to introduce proper statistics at some point. In this thesis, the results and scorings have been simple. This is acceptable at the current scale. If we run all experiments on a larger scale, the current score might be too one-dimensional. We could make more meaningful changes and adjustments with proper statistics.

The field of large language model is relatively novel and changing at a fast pace. As seen, there are many possibilities to explore for research still. Particularly in combination with networking, there is potential to uncover new avenues to utilize it. The previously discussed actions could be a stepping stone toward realizing this potential.

Bibliography

[1] BASTIAN, M. Gpt-4 has more than a trillion parameters - report. `https://the-decoder.com/gpt-4-has-a-trillion-parameters/`, 2023. Accessed: 02.01.2024.

[2] BROWN, T., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J. D., DHARIWAL, P., NEELAKANTAN, A., SHYAM, P., SASTRY, G., ASKELL, A., ET AL. Language models are few-shot learners. *Advances in neural information processing systems 33* (2020), 1877–1901.

[3] CHEN, M., TWOREK, J., JUN, H., YUAN, Q., PINTO, H. P. D. O., KAPLAN, J., EDWARDS, H., BURDA, Y., JOSEPH, N., BROCKMAN, G., ET AL. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[4] DALE, R. Gpt-3: What's it good for? *Natural Language Engineering 27*, 1 (2021), 113–118.

[5] FRROUTING. Frrouting project. `https://frrouting.org`, 2023. Accessed: 03.01.2024.

[6] GAO, L., AND REXFORD, J. Stable internet routing without global coordination. *IEEE/ACM Transactions on Networking 9*, 6 (2001), 681–692.

[7] GOOGLE. Bard, a conversational ai tool by google. `https://bard.google.com/?hl=en`, 2023. Accessed: 03.01.2024.

[8] GROUP, N. S. Communication networks. `https://comm-net.ethz.ch/`, 2023. Accessed: 03.01.2024.

[9] MONDAL, R., TANG, A., BECKETT, R., MILLSTEIN, T., AND VARGHESE, G. What do llms need to synthesize correct router configurations? In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (2023), pp. 189–195.

[10] NGUYEN, N., AND NADI, S. An empirical evaluation of github copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories* (2022), pp. 1–5.

[11] OPENAI. New models and developer products announced at devday. `https://openai.com/blog/new-models-and-developer-products-announced-at-devday`, 2023. Accessed: 03.01.2024.

[12] OPENAI. Pricing. `https://openai.com/pricing`, 2023. Accessed: 03.01.2024.

[13] OpenAI. Prompt engineering. `https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results`, 2023. Accessed: 03.01.2024.

[14] OpenAI. Text generation models. `https://platform.openai.com/docs/guides/text-generation/chat-completions-api`, 2023. Accessed: 03.01.2024.

[15] Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615* (2022).

[16] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).