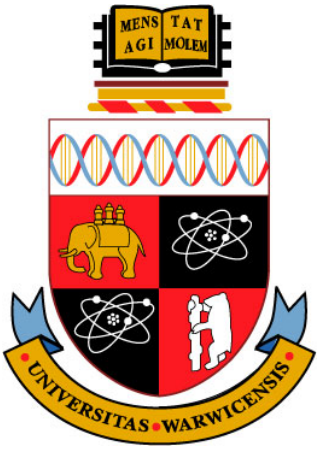


Evaluating Factuality of Biomedical Text Simplification Using Large Language Models

Kailai He

CS907

Supervisor: Dr. Gabriele Pergola

Department of Computer Science

University of Warwick

September 2025

Abstract

Text simplification of biomedical text refers to producing a simpler version of a typically jargon-filled medical article. Unlike text summarisation, where the goal is to generate an abstract of the original text, text simplification aims to provide a sentence-by-sentence simplified version. This helps lay readers better understand the content. However, producing simplified biomedical texts manually is both costly and inefficient. Large Language Models (LLMs), which have demonstrated strong capabilities across many NLP tasks, including text simplification, offer a potential solution.

It is essential, however, that the output of LLMs remains factual. The simplified text should not omit or alter information from the original, nor should it introduce additional information that is absent from the source. Human evaluation of LLM outputs for factuality is also costly and inefficient, so our aim is to leverage LLMs themselves to evaluate the factuality of simplified texts compared to their originals.

In this project, we tested several reasoning and non-reasoning LLMs under different prompt and inference settings, using both the PlainFact and Cochrane datasets, followed by a systematic analysis. We found that while recent reasoning models show strong capabilities in many reasoning-related tasks, they do not always outperform non-reasoning models. Furthermore, the output type specified in the prompt can significantly affect model performance. For example, if the model is instructed to answer "Yes" when the simplified text is factual and "No" when it is not, most models display a "Yes" bias, simply preferring to answer "Yes" regardless of whether it indicates factuality or non-factuality.

Acknowledgements

I would like to especially thank my supervisor, Dr. Gabriele Pergola, for his invaluable support. He helped me build this project, patiently answered my questions, and provided guidance on how to proceed.

Contents

List of Figures

List of Tables

1 Introduction

Automatic Text Simplification (ATS) is a Natural Language Processing (NLP) task aimed at producing an easier-to-read version of a given text for lay readers. In the biomedical domain, this task is particularly meaningful, as biomedical texts often contain specialised jargon that is difficult for the general public to understand. However, in this domain, ensuring that the simplified text remains factual is crucial. Any non-factual text introduced by mistake from automatic approachs, such as "you got cancer" when the original text is "you did not get cancer", is highly misleading and can have serious consequences. In this project, we aim to test large language models' (LLMs) ability to evaluate the factuality of simplified biomedical text.

In Table 1.1, we show two examples of simplified text alongside their original texts. We can see that the original texts are full of technical terminology, whereas the simplified versions omit such terms. These examples demonstrate the complexity of text simplification, which involves more than merely reducing lexical complexity: it often requires restructuring sentences into simpler forms to improve the overall readability.

Dataset	Original text	Simplified text
Cochrane	Analysis showed a higher rate of weight gain in the high-volume feeds group: mean difference 6.20 g/kg/d (95% confidence interval 2.71 to 9.69). There was no increase in the risk of feed intolerance or necrotising enterocolitis with high-volume feeds, but 95 % confidence intervals around these estimates were wide.	Very low birth weight infants who receive more milk than standard volumes gain weight more quickly during their hospital stay. We found no evidence suggesting that giving infants high volumes of milk causes feeding or gut problems, but this finding is not certain.
Medical Parallel English Wikipedia Corpus	Lowered glucose levels result both in the reduced release of insulin from the beta cells and in the reverse conversion of glycogen to glucose when glucose levels fall.	This insulin tells the cells to take up glucose from the blood.

Table 1.1: Examples from Cochrane [12] and Medical Parallel English Wikipedia Corpus [22].

It is worth highlighting the relation of automatic text simplification with another relavant task in NLP: *Text Summarisation*. The aim of summarisation is to produce a condensed version of the text that captures the main idea of the passage. While text simplification also needs to preserve the core idea of the original text, it differs in that it should retain the detailed information of the text, rather than discard it. From Table 1.1, we observe that the simplified text is shorter than the original text, but length is not a criterion of simplification. A simplified text does not necessarily need to be shorter than the original; it may even be longer if this improves readability and accessibility.

This intrinsic complexity of task may beneift from sofisticate reasoning abilities of linguistic models: as text simplification involves not only linguistics skills in rewriting the content, but also accurate understanding in preserving the original and correct meaning, assessing whether more or less knowledge is needed to ease the comprehension, For this reason, in this project, we aim at studying two kinds of large language models: reasoning and non-reasoning large language models. Reasoning LLMs models typically generate longer outputs, often including intermediate steps and explanations, with the purpose of deriving more accurate answers. Such reasoning ability is especially useful in tasks such as question answering, where outlining the thought process can help the model arrive at the correct conclusion. Here, we investigate whether leveraging reasoning capabilities can enhance the ability of language models to evaluate factuality.

This thesis makes three main contributions. First, it provides a systematic audit of the PlainFact-Summary dataset, a recently developed dataset, uncovering previously undocumented artifacts: factual and non-factual plain-language summaries are paired in a fixed offset pattern, and the non-factual versions are generated through a limited set of operations such as numeric swaps, antonym substitutions, entity changes, and negations. We quantify the prevalence of these operations and highlight their implications for evaluation. Second, we design and carry out a bias-robust evaluation of LLMs for factuality assessment. In addition to testing models under the original *"yes/no" prompt*, we introduce reversed-label and "0/1" variants to probe lexical sensitivity, showing that some families flip their behaviour depending on prompt polarity. Third, we conduct a model agreement analysis, using pairwise outcome comparisons and Cohen's κ to reveal complementarities and tensions across model families. While the question-answering pipeline employed for comparison is an adaptation from prior work, these empirical analyses,

i.e., dataset auditing, bias stress tests, and agreement mapping, constitute the contributions of this dissertation.

2 Background

There has been work investigating how modern LLMs perform on different NLP tasks in the biomedical domain [4]. For different models, a range of prompting strategies were tested, such as zero-shot and few-shot prompting. They also fine-tuned BERT for specific tasks. Among the NLP tasks covered, two relevant ones are text simplification and question answering.

It should be noted that this work was conducted in 2023, so the models evaluated are not the most recent ones (e.g., GPT-3.5, GPT-4, T5, BERT, and BART). The main findings are that GPT-4 performs best on tasks involving reasoning, such as question answering, whereas for most tasks the best-performing model was a version of BERT fine-tuned for that specific task. This may be because the models used for comparison are relatively old, and more recent models such as GPT-OSS could potentially perform better given their stronger reasoning abilities.

When it comes to the evaluation of factuality, existing automatic metrics have been shown to be unreliable [18]. The authors investigated whether superficial attributes of simplified or generated texts are sufficient to evaluate "factuality". They trained a supervised model using only shallow features (listed below), and its performance was comparable to that of state-of-the-art factuality evaluation metrics. Furthermore, they found that some state-of-the-art factuality metrics do not exhibit a positive correlation between correct samples and their scores, and that certain metrics can be easily manipulated.

The shallow features list:

- lexical overlap
- entity overlap
- semantic similarity
- sentence novelty ratio – the proportion of words and sentences in the summary not in the source
- conciseness ratio – the ratio of source length to summary length

State-of-the-art large language models are *reasoning models*, where the idea is to generate longer responses that allow the model to "think" more before producing a final answer.

To enable models to generate longer responses, two main types of methods are used.

The first type involves approaches such as reinforcement learning (RL), supervised fine-tuning (SFT), or a combination of RL and SFT. These methods increase training-time compute costs [19].

The second type increases inference-time compute, and this class of methods has drawn significant attention since the release of DeepSeek R1.

Unlike the first type, these methods do not require modifying model parameters. Instead, they use strategies such as adding a "wait" token to prompt the model to re-evaluate its responses [14], or defining a reward model that guides the model to refine its answers [11].

By design, reasoning models generate more tokens, so whether training, fine-tuning, or performing inference, they require more computational power than standard models. Thus, using these models necessitates powerful compute resources.

2.1 PlainQAFact

The PlainQAFact framework [25] is specifically designed for evaluating the factuality of simplified texts, particularly in the biomedical domain. It combines a question–answering (QA) system with a retrieval-augmented generation (RAG) module to assess whether the simplified text is faithful to the original.

The concept of **elaborative explanation** is introduced in this paper, describing cases where new, external content—such as definitions, background information, or explanations—is introduced into the simplified text but is absent from the original scientific abstract. The claim is that the presence of elaborative explanations makes the evaluation of factuality in simplified text more challenging. The framework therefore incorporates strategies to address elaborative explanations, with the goal of improving the overall evaluation of factuality.

The framework consists of five main components: Classifier, Answer Extraction, Retrieval Source, Answer Overlap, and Granularity Level. The Classifier and Answer Extraction components are implemented using language models. The design is model-agnostic, meaning that any language model can be used in these roles, making it straightforward to swap different models to identify the best-performing configuration. In their experiments, the authors found that the best-performing combination was a custom learned classifier and LLaMA-3.1-8B Instruct for answer extraction.

This novel approach significantly outperforms existing factuality metrics at both the sentence and paragraph levels. Unlike other

metrics, which only account for limited factors, this approach takes a broader perspective. However, at the paragraph (summary) level, the LLaMA-3.1-8B model itself outperformed all metrics, including PlainQAFact. The authors suggest that this limitation is primarily due to their implementation of the RAG component: while RAG is effective in principle, the specific implementation in PlainQAFact was not optimised.

The paper also introduces a novel dataset, the *PlainFact* dataset, which is highly valuable and relevant for our project.

2.2 medval

The MedVAL framework [2] is a novel approach designed to fine-tune models for performing factual evaluation on biomedical text across six tasks: medication question answering, patient query summarization, radiology report summarization, radiology impression simplification, hospital course translation, and doctor–patient dialogue summarization. The paper defines each of these tasks. For example, the definition of the radiology impression simplification task is: "produce patient-friendly rewrites that preserve the clinical meaning while being understandable to a layperson." This aligns closely with the goal of text simplification, suggesting that models fine-tuned on this task can be effectively applied to simplification.

The ultimate aim of MedVAL is to enable fine-tuned models to evaluate the factuality of processed text relative to the original text, producing a factuality score that corresponds to a *risk level*, while also identifying the type of factual error, if present.

There are four risk levels: "no risk", "low risk", "moderate risk", and "high risk", which correspond to risk levels 1, 2, 3, and 4 respectively. Processed text with no more than level 2 risk is considered safe to deploy, meaning it is suitable for public use. In addition, four types of factual error are defined: "Hallucinations", "Omissions", "Certainty Misalignments", and "Other". The detailed error taxonomy is shown in Table 2.1.

The main part of the MedVAL framework is composed of two steps and two roles. The two steps are *synthetic text generation* and *data filtering*. The two roles are a *generator* and a *validator* (the validator can be viewed as a classifier). As the paper points out, there are limited resources of high-quality annotated biomedical data, and collecting such data is costly. The framework is designed to bypass these obstacles.

In step one, synthetic text generation, biomedical text is fed into the generator, which then applies perturbations to the text with

Error category	Error
Hallucinations	Fabricated claim
	Misleading justification
	Detail misidentification
	False comparison
	Incorrect recommendation
Omissions	Missing claim
	Missing comparison
	Missing context
Certainty Misalignments	Overstating intensity
	Understating intensity
Other	Other

Table 2.1: Error taxonomy by category and type.

some probability. The purpose of perturbation is to produce a non-factual version of the input text. The perturbed or unperturbed text is then fed into the validator, which assigns a factuality score and identifies the type of factual error, if any. By randomising whether or not to perturb the text, the validator cannot learn to rely on patterns in the perturbation process itself.

The output of the validator is then used as training data to fine-tune the validator itself. As the paper notes, this constitutes a form of self-training.

An advantage of this framework is that it can be applied to any language model. For all the models tested, there was significant improvement across all tasks after fine-tuning with this framework. The models tested include LLaMA 3.1, LLaMA 3.2, LLaMA 3.3, GPT-4o Mini, GPT-4o, Qwen 3, MedGemma, Gemma 3, Gemini 2.0, and Claude 4.

The paper also introduces a dataset called *MedVAL-Bench*. This dataset consists of 840 physician-annotated outputs across the six tasks. Unfortunately, it contains only factual examples, making it an unbalanced dataset.

Overall, the framework is novel, but due to time and resource constraints we were unable to fine-tune a model for this project. Fortunately, the authors published their best-performing fine-tuned model, which is based on Qwen3_4b, and we use this model in our project.

2.3 Factual Inconsistency Taxonomy

Work has been done to define different types of factual inconsistency [6]. With a clear taxonomy, it becomes easier to identify, categorise, and analyse the performance of models. By understanding which types of factual errors a model struggles with, we can then focus on improving the model in those specific areas.

In Table 2.2, we show examples of factual errors defined according to a taxonomy of factual inconsistency [6]. This is not the only taxonomy that exists, and it is difficult to determine which is the most appropriate or comprehensive.

This taxonomy categorises factual inconsistencies into three types: *information insertion*, *information deletion*, and *information substitution*. *Information insertion* occurs when the non-factual text contains information not present in the original text. *Information deletion* occurs when the non-factual text omits information that is present in the original text. *Information substitution* occurs when the non-factual text introduces information that differs from the original text.

Type	**Definition**	**Example**
Information Insertion	Contains information not present in the original text.	The cat is on the mat → The cat is on the mat in the living room
Information Deletion	Does not include information that is present in the original text.	The cat is on the mat in the living room → The cat is on the mat
Information Substitution	Contains information that differs from the original text.	The cat is on the living room mat → The cat is on the carpet in the living room

Table 2.2: Types of factual errors

For each type of factual inconsistency, different levels of severity can be defined, as shown in Table 2.3.

A severity level of 0 indicates no or trivial changes. Level 1 indicates non-trivial changes that do not alter the main meaning, such as the examples shown in Table 2.2. Level 2 indicates changes that alter the main meaning. For example: "The cat is on the mat in the living room, and the dog is on the sofa", where new information ("the dog") is introduced that is absent from the original text. Finally, level -1 indicates gibberish or text that is not understandable. This often occurs when weaker models are used, where the

output may become repetitive or nonsensical.

Level	Description
0	No or trivial changes.
1	Nontrivial changes that do not change the main meaning.
2	Changes that change the main meaning.
-1	Gibberish; text that is not understandable.

Table 2.3: Levels of Severity

Note that with taxonomy like this, we say that we are testing the model's faithfulness, which is different than the model been factual. But in our context, since we are given an scientific abstract to the model, and we assume the abstract is factual, then if the model is faithful when generating simplified text of the abstract, it is equivalent to the model is been factual.

2.4 Models

in this section, we will go through the models that we will use for this project.

2.4.1 Reasoning Models

Qwen3 Qwen3 [20] is a reasoning mixture-of-experts (MoE) model developed by Alibaba. The reasoning ability of the model can be toggled on or off when using Hugging Face's `transformers` package. This feature allows us to compare model performance with and without reasoning enabled.

Qwen3 is available in three sizes: 4B, 30B, and 235B. Due to limitations in time and computational resources, we use the 30B version to balance performance and efficiency. We refer to this model as **Qwen_think** when reasoning is enabled, and as **Qwen_nothink** when reasoning is disabled.

MedVAL We introduced the MedVAL framework in Section 2.2, where the authors published their best-performing model fine-tuned using this framework. We employ this model to compare the performance of a fine-tuned model with other unmodified models. The base model of the fine-tuned version is Qwen3_4b, and we refer to this fine-tuned version of Qwen3 as **MedVAL**.

Since the base model Qwen3 can toggle its reasoning ability on and off, MedVAL also supports both reasoning and non-reasoning modes. We refer to MedVAL with reasoning enabled as

MedVAL_think, and MedVAL with reasoning disabled as **MedVAL_nothink**.

Note that the MedVAL framework is designed to work with six different tasks, and a specially designed prompt template was used during fine-tuning. Therefore, to properly set up the model, we must use the original prompt template. Furthermore, since the model was fine-tuned on six tasks, it requires a `task_description` variable so that the model knows the current task is *text simplification*. The prompt template and the task name are provided in Appendix A.

GPT-OSS The GPT-OSS model [17] is a mixture-of-experts (MoE) reasoning model recently published by OpenAI. The model comes in two sizes: 20B and 120B. Unlike Qwen3, we cannot toggle the reasoning ability of GPT-OSS on or off. Instead, OpenAI provides three levels of reasoning effort: "low", "medium", and "high". At lower reasoning effort, the model produces less detailed reasoning and requires less inference time. At higher reasoning effort, the model generates deeper and more detailed reasoning chains, but at the cost of increased inference time.

For this project, we used GPT-OSS both via Ollama and Hugging Face. While Ollama provides a simple and convenient interface, Hugging Face offers more flexibility and options for controlling the model. In particular, Ollama does not currently support setting the reasoning level, which can only be adjusted using Hugging Face's `transformers` library. Furthermore, Ollama is not designed for use in batch systems such as the Warwick HPC servers, and integrating it into the HPC workflow proved impractical. Therefore, we used GPT-OSS via Hugging Face on the HPC servers, while Ollama was used only for local runs on our own machines.

We were unable to run the 120B version of GPT-OSS via Hugging Face on the HPC servers due to resource limitations. Thus, the 120B model was only run locally using Ollama.

Since Ollama provides no explicit instructions on role settings, we experimented with both a `developer` + `user` role setup and a single `user` role.

Throughout this report, we refer to GPT-OSS as **OSS**. The 20B and 120B versions are denoted as **OSS_20** and **OSS_120**, respectively. When accessed via Ollama's API, we denote them as **OSS_20_o** and **OSS_120_o**. To indicate different reasoning efforts, we add suffixes to the Hugging Face model names: **OSS_20_low**, **OSS_20_med**, and **OSS_20_high**. Since Ollama does not allow modification of reasoning effort, no such suffixes are applied to its

model names.

Marco-O1 Marco-O1 [26] is another reasoning model built by Alibaba. It is a 7B-parameter model inspired by OpenAI's O1. The aim is to replicate O1-like reasoning characteristics, though its performance still falls short of OpenAI's original O1. Unlike Qwen3, Marco-O1 does not provide an option to disable reasoning, so we cannot compare its performance with and without reasoning enabled.

We refer to this model as **Marco_O1**.

deepseek-r1 The DeepSeek-R1 [5] model is a reasoning model trained with Reinforcement Learning (RL) and Supervised Fine-Tuning (SFT).

Its performance is comparable to the GPT-O1 model. The model is available through both Hugging Face and Ollama. However, we were only able to successfully run the model via Ollama's API on our own laptop, and failed to run it on the DCS compute nodes. This occurred during the early stage of the project, when we lacked experience with running large language models that require multiple GPUs to share memory.

The model is released in different sizes. We used the 7B version, which at the time was the newest. Although an 8B version has since been released, we did not have time to test it. We also experimented with the 32B version to investigate whether a larger model would yield better performance.

Since we accessed the model through Ollama, disabling the reasoning ability was not supported. We therefore refer to the DeepSeek-R1 7B version as **DS_7**, and the 32B version as **DS_32**.

Open-R1 The Open-R1 [7] model is a replication of the DeepSeek-R1 model. It was built by the Hugging Face community, based on Qwen2.5-Math-7B [24] and fine-tuned on Mixture-of-Thoughts, a dataset containing reasoning traces distilled from DeepSeek-R1. The aim is to enable Qwen2.5 to replicate the reasoning process of DeepSeek-R1. Qwen2.5 itself is designed to solve mathematical problems using Chain of Thought (CoT) reasoning and Tool-Integrated Reasoning (TiR) in both English and Chinese.

Open-R1 is a 7B model and can fit on a GPU with 24GB of RAM. However, the model is not stable. During testing on the Cochrane dataset, we observed frequent repetition and gibberish in its outputs. At times it produced long passages that were incoherent.

This instability made it difficult to generate structured answers, extract model outputs, and evaluate performance.

The instability is likely due to the fact that the model is community built and therefore not as well-tuned as other models. To avoid spending excessive time adjusting the model, we decided not to use Open-R1 in our project.

2.4.2 Non Reasoning Models

Llama We used three models from the Llama-3 family: Llama-3-8B, Llama-3-8B-Instruct [3], and Llama-3.1-8B-Instruct [1]. The Llama series models are developed by Meta, and the Llama-3 family comes in two sizes: 8B and 70B. Due to limitations in time and computational resources, and to maximise efficiency, we used the 8B versions. The Llama-3 series only supports English.

The Llama-3-8B model is the only model we used that does not support multiple roles in its chat template, so we had to place all inputs into a single prompt and feed it directly to the model.

Llama-3-8B-Instruct is the instruction-tuned version of Llama-3, designed to function as an assistant-like chatbot. Its aim is to improve dialogue performance and alignment compared to the base model. However, this improvement comes at the cost of reduced helpfulness, though the exact quantised trade-off has not been reported.

The Llama-3.1 series comes in three sizes: 8B, 70B, and 405B. Unlike Llama-3, the Llama-3.1 series is multilingual, supporting eight languages in total. We used the 8B instruction-tuned model, which is also intended as an assistant-like chatbot with stronger alignment.

For convenience, we refer to Llama-3-8B as **Llama3**, Llama-3-8B-Instruct as **Llama3_I**, and Llama-3.1-8B-Instruct as **Llama3.1_I**.

We also attempted to run models from the Llama-4 family, specifically Llama-4-Scout and Llama-4-Maverick. These are recent MoE reasoning models, with Scout containing 109B parameters and Maverick 400B. However, due to our lack of experience (leading to frequent out-of-memory errors) and the large model sizes, we were unable to use them. For example, the Llama-4-Scout model has a file size of over 200GB, which means that if we want to use it on Blythe, we have just enough storage for this one model, and if we want to test a variant or a different model, we have to delete the previous model then download the new one. And if we need to use the deleted model sometime later, we need to re-download the model, which is very time consuming and inefficient. As discussed in Sec-

tion 4.1, we also encountered downloading issues with the Hugging Face API, and therefore we ultimately did not include the Llama-4 family models in this project.

Mistral The Mistral-7B-Instruct-v0.3 [9] model was developed by the Mistral.ai team. It is a 7B-parameter model instruction-tuned on the Mistral-7B-v0.3 base model. This version improves upon the original Mistral-7B-v0.1 model, with updates including a larger vocabulary size, a newer tokenizer, support for function calling, a larger context window, and the removal of sliding-window attention. Although the original Mistral-7B-v0.1 is relatively old, benchmarking shows that it was already more capable than Llama-2-13B.

We refer to Mistral-7B-Instruct-v0.3 simply as **Mistral**.

3 Methodology

To evaluate the models' ability to assess the factuality of simplified biomedical text, we ran inference with LLMs under a zero-shot setting on the PlainFact dataset, allowing the LLMs to act as classifiers that output the factuality of the PLS.

We also built a QA system, again in a zero-shot setting, to evaluate the factuality of simplified text using both the Cochrane and PlainFact datasets.

3.1 Question Answering Pipeline

In this section, we discuss the QA system designed to evaluate whether simplified text is factual based on LLMs. In this system, we used the DeepSeek-R1 model as the base model, and applied a series of instructions to guide the model in identifying non-factual content in the simplified text. The QA pipeline is summarised below, and its structure is shown in Figure 3.1.

1. Instruct the model to generate a list of facts from both the simplified text and the original text.
2. Instruct the model to compare the two lists of facts, and record all facts that appear in the simplified text but not in the original text.
3. Instruct the model to read the record. If there are facts listed, this indicates that the simplified text contains information not supported by the original text, and therefore the simplified text is non-factual. In this case, return "yes"; otherwise, return "no".

The idea of listing facts from the original and simplified texts is adapted from another QA system [21], and we used the exact prompt from that paper for our QA system. The full prompt is provided in Appendix B.

3.2 PlainFact Analysis

As the PlainFact summary dataset provides only limited data, we conducted a manual analysis of the dataset.

In Table 3.1, we present some basic statistics about the PlainFact sentence dataset, as reported in the original paper. The *Explanation* and *Simplification* columns represent two types of sentences in the dataset: explanations introduce elaborative content, often adding

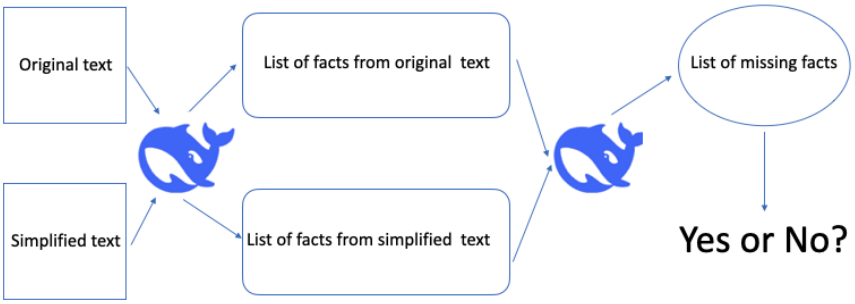


Figure 3.1: The QA pipeline structure.

new ideas, while simplifications are purely simplified versions of the original sentences from the scientific abstracts (see Section 4.2.1). The *Total* column is the sum of these two types.

It should be noted that the total vocabulary size may not be entirely meaningful, since words appearing in both explanation and simplification sentences may overlap, resulting in duplication in the overall vocabulary count.

	Explanation	**Simplification**	**Total**
# of Sentences	1,213	1,527	2,740
Average Length (token)	29	28	57
Vocabulary Size	4,230	4,046	8,276

Table 3.1: Overview of the PlainFact sentence dataset.

In Table 3.2, we report similar statistics for the PlainFact summary dataset, comparable to those in Table 3.1. The total number of summaries, or examples, is 400, with an average length of approximately 298 tokens—about five times longer than the average length of sentences in the sentence-level dataset. The vocabulary size is smaller than the total vocabulary size of the sentence dataset, which is expected since explanation and simplification sentences share overlapping terms. However, when comparing the vocabulary size of the summary dataset to that of the explanation and simplification subsets individually, the summary dataset is slightly larger, as it essentially represents the concatenation of both types of sentences.

Note that the factual and non-factual PLS corresponding to the same scientific abstract are separated into two columns. By manually inspecting the *Original Abstract* column, we observe that the first half of the dataset (indices 0–199) consists entirely of factual PLS, while the second half (indices 200–399) contains non-factual

Table 3.2: Overview of the PlainFact Summary dataset.

Metric	Value
Number of Summaries	400
Average Length (tokens)	298.7825
Vocabulary Size	5639

PLS. The pairing follows a consistent pattern: index 0 is paired with index 200, meaning that the PLS at index 0 is factual and the PLS at index 200 is non-factual, and both are derived from the same scientific abstract. This pattern continues for the entire dataset.

Below we provide a manual analysis of the PLS at indices 0 and 199. An interesting pattern emerges when comparing the two. We have highlighted the inconsistent parts between the two PLS in **bold** font. One PLS is factual, and the other is non-factual. The differences are subtle but systematic: for instance, "five" is changed to "two" (a change in quantity); "estrogen" to "testosterone" (a change in hormone type); "small" to "large" (a change in size); "flexible" to "rigid" (a change in physical property); "drugs" to "chemicals" (a change in substance type); and "could be" to "could not be" (a negation that reverses the original meaning).

Factual PLS (index 0)	Non-factual PLS (index 200)
The skin patch and the vaginal (birth canal) ring are **two** methods of birth control. Both methods contain the hormones **estrogen** and progestin. The patch is a **small**, thin, adhesive square that is applied to the skin. The contraceptive vaginal ring is a **flexible**, lightweight device that is inserted into the vagina. Both methods release **drugs** like those in birth control pills. These methods **could be** used more consistently than pills because they do not require a daily dose.	The skin patch and the vaginal (birth canal) ring are **five** methods of birth control. Both methods contain the hormones **testosterone** and progestin. The patch is a **large**, thin, adhesive square that is applied to the skin. The contraceptive vaginal ring is a **rigid**, lightweight device that is inserted into the vagina. Both methods release **chemicals** like those in birth control pills. These methods **could not be** used more consistently than pills because they do not require a daily dose.

Table 3.3: Manual comparison of factual and non-factual plain-language summaries (PLSs) derived from the same scientific abstract. Highlighted tokens indicate the inconsistent parts: left = factual (index 0), right = non-factual (index 200).

This appears to be a common pattern underlying the differences between factual and non-factual PLS, which also helps explain how

the authors created the non-factual versions. They may have employed rule-based methods to identify keywords representing quantities, sizes, hormones, or properties, and occasionally introduced a negation into the original PLS.

In Table 3.4, we report the average number of tokens for factual and non-factual PLS, as well as the average difference between them. We observe that the mean difference in token length is 1.6, which is approximately the number of tokens required to add a negation, such as inserting the word "not". This observation is further supported by the difference in the number of words between factual and non-factual PLS, which is only 0.81.

Table 3.4: Number of Token Statistics

Metric	**Value**
Average number of token (factual)	298.08
Average number of token (nonfactual)	298.75
Average absolute difference in number of token per pair	1.6
Average # of words (factual)	298.38
Average # of words (nonfactual)	299.19
Average absolute difference in # of words per pair	0.81

In Figure 3.2, we illustrate the difference in word counts between factual and non-factual PLS pairs. This difference is computed as the number of words in the non-factual PLS minus the number of words in the corresponding factual PLS. We observe that, in most cases, non-factual PLS contain more words than their factual counterparts. This is consistent with our earlier observations: for example, non-factual PLS may include additional words such as "not", and in some cases multiple insertions or other lexical additions are introduced.

In Figure 3.3 and Table 3.5, we present the number of differing words between each factual and non-factual PLS pair. On average, the difference is 20.03 words, with no pair differing by more than 46 words. From the figure, we observe that most pairs contain between 10 and 25 differing words, which is considerably smaller than the average total length of both factual and non-factual PLS (approximately 299 words). This further suggests that the differences between factual and non-factual PLS are relatively minor, and that the majority of content is shared between them.

To further investigate the PlainFact dataset, we defined four types of word swaps between factual and non-factual PLS: **Negation**, **Numeric**, **Antonym**, and **Keyword**. Examples of these can be seen in the PLS pairs discussed above: "could be" → "could not

Table 3.5: Number of Different Words per Pair

Statistic	Value
Mean	20.03
Median	19
Minimum	5
Maximum	46

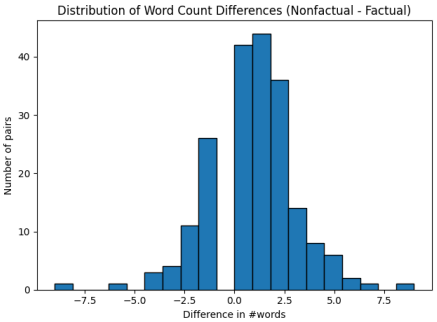


Figure 3.2

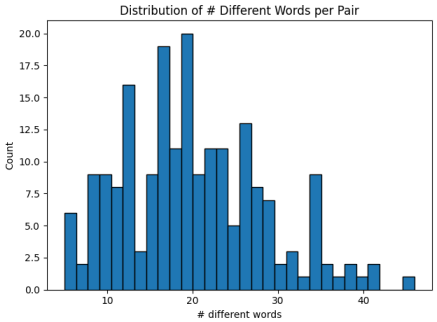


Figure 3.3

be" is a negation swap; "two" → "five" is a numeric swap; "small" → "large" is an antonym swap; and "estrogen" → "testosterone" is a keyword swap.

For antonym swaps, we used WordNet to help identify candidate pairs. WordNet is a lexical database in which words are grouped into *synsets* (sets of synonyms) according to their meanings. It functions similarly to a dictionary, but also encodes conceptual relations between words. These relations include hyponymy, hypernymy, and others. WordNet currently contains over 117,000 synsets, each linked to other synsets through semantic relations. Each synset is accompanied by a **gloss**, i.e., a brief definition that may encompass multiple senses of a word. For instance, the word "chump" is linked with "fool$_2$", meaning that "chump" is associated with the second sense of the word "fool".

However, WordNet also presents certain challenges. First, when constructing a tree based on synsets, the same word with different senses may appear at different levels of the hierarchy. Second, social bias may be reflected in some of the relations encoded in the resource.

For negation swaps, we defined a list of common negation prefixes, which we used to identify unique words that may carry negated meanings. Here, *unique words* are those that appear in only one of the PLS in a pair. Inspired by our observations of the PLS pairs above, we also considered explicit additions of the word "not" in the

non-factual PLS. Whenever such a unique "not" was detected, we counted it as a negation swap.

$$NEG\_PREFIXES = (un, in, im, il, ir, non, dis)$$

For numeric swaps, we simply checked whether the unique words in the factual and non-factual PLS were both numbers; if so, we counted this as a numeric swap.

For keyword swaps, we used four named entity recognition (NER) models from SciSpaCy [16] to identify biomedical terms in the PLS. We then extracted unique terms that appeared only in the factual or the non-factual PLS, and treated these as keyword swaps. The list of NER models we used is provided below, each of which recognises a different set of entity types.

```
models = {
    "BC5CDR": spacy.load("en_ner_bc5cdr_md"),
    "CRAFT": spacy.load("en_ner_craft_md"),
    "JNLPBA": spacy.load("en_ner_jnlpba_md"),
    "BioNLP13CG": spacy.load("en_ner_bionlp13cg_md"),
}
```

In Table 3.6, we list the NER models from SciSpaCy that we used, along with the entity types they can recognise. Each model specialises in different types of biomedical entities, and by combining all four models, we are able to cover a broad range of biomedical terms.

Table 3.6: Biomedical NER Models and Supported Entity Types

Model	**Entity Types**
en_ner_craft_md	GGP, SO, TAXON, CHEBI, GO, CL
en_ner_jnlpba_md	DNA, CELL_TYPE, CELL_LINE, RNA, PROTEIN
en_ner_bc5cdr_md	DISEASE, CHEMICAL
en_ner_bionlp13cg_md	AMINO_ACID, ANATOMICAL_SYSTEM, CANCER, CELL, CELLULAR_COMPONENT, DEVELOPING_ANATOMICAL_STRUCTURE, GENE_OR_GENE_PRODUCT, IMMATERIAL_ANATOMICAL_ENTITY, MULTITISSUE_STRUCTURE, ORGAN, ORGANISM, ORGANISM_SUBDIVISION, ORGANISM_SUBSTANCE, PATHOLOGICAL_FORMATION, SIMPLE_CHEMICAL, TISSUE

Note that this part of the analysis is rather preliminary. We used WordNet to help identify antonyms, but WordNet is not per-

fect and may miss some cases. Negation swaps were identified only through predefined prefixes and unique instances of "not", which may overlook other forms of negation. Keyword swaps were identified using NER models, which may not capture all biomedical terms. Moreover, our four categories of swaps were designed based on observations from a single PLS pair, and there may be other types of swaps that we did not consider.

In Table 3.7 and Figure 3.4, we present the exact counts and a graphical representation of the number of pairs containing each type of swap. The dataset contains 200 pairs in total, and for each swap type we detect approximately 150 pairs. Among the categories, numeric swaps occur least frequently, followed by antonym swaps, then negation swaps, while keyword swaps are the most common. From this we can conclude that our four types of swaps are present in almost every PLS pair.

Furthermore, most pairs involving a negation swap include an additional "not" in the non-factual PLS: specifically, 133 out of 151 such pairs. As shown in Table 3.7, the average number of additional "not" tokens per pair is 1.23 when considering all pairs, and 1.84 when considering only the pairs with an extra "not". This indicates that, unlike the example PLS pair discussed earlier, most pairs contain more than one instance of "not" in the non-factual PLS. Therefore, multiple swaps of the same type can occur within a single pair of PLS.

Table 3.7: Swap Type Counts

Swap Type	**Count**
Pairs with numeric swap	142
Pairs with antonym swap	146
Pairs with negation swap	151
Pairs with keyword swap	163
Pairs with extra 'NOT' in nonfactual	133
Average extra NOTs per pair (all pairs)	1.23
Average extra NOTs per affected pair	1.84

In Table 3.8 and Figure 3.5, we present the most frequent antonym swaps observed in the PLS pairs. The most common antonym swap is "large" $\leftrightarrow$ "small", which occurs 23 times, followed by "less" $\leftrightarrow$ "more", which occurs 22 times. However, when compared with the total of 200 PLS pairs, no single antonym swap appears in more than 12% of the dataset. This indicates that while antonym swaps are relatively common overall, they are highly diverse: each specific antonym pair is found in only a small number of PLS pairs, and no single antonym swap dominates.

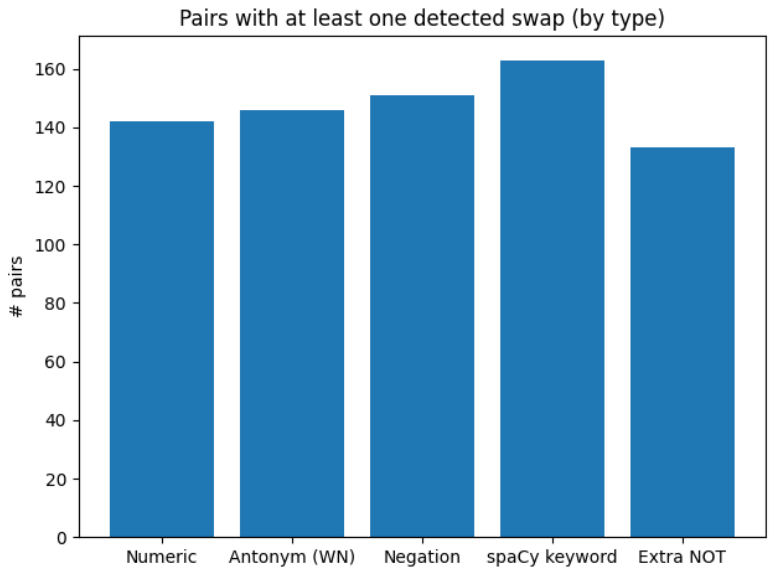


Figure 3.4: Here spaCy keyword represent keyword swaps, and Extra NOT is part of Negation swaps.

Most antonym swaps relate to sizes, quantities, or comparisons, such as "large" ↔ "small", "less" ↔ "more", and "decreased" ↔ "increased". Notably, these antonym words are simple and frequently used terms, which makes them likely to appear in PLS. This also suggests that identifying and altering such words in PLS is straightforward, and even these small lexical changes can substantially affect the factuality of a statement.

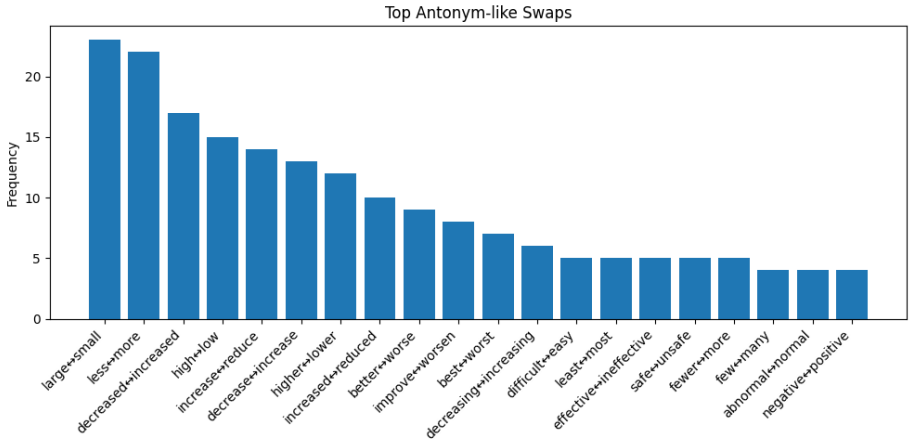


Figure 3.5: Top Antonym swaps for all pairs.

In Table 3.9, we present the most frequent numeric swaps and

Antonym Pair	Number of Pairs
large ↔ small	23
less ↔ more	22
decreased ↔ increased	17
high ↔ low	15
decrease ↔ increase	13
increased ↔ reduced	10
better ↔ worse	9
improve ↔ worsen	8
best ↔ worst	7
decreasing ↔ increasing	6
difficult ↔ easy	5
least ↔ most	5
effective ↔ ineffective	5
fewer ↔ more	5
few ↔ many	4
abnormal ↔ normal	4
negative ↔ positive	4
long ↔ short	4
decrease ↔ increases	3
painful ↔ painless	3

Table 3.8: Top antonym-like swaps

negation swaps observed in the PLS pairs. Numeric swaps by themselves are not particularly interesting, since in principle any number can be swapped for another. However, upon closer inspection, we see that the swapped numbers usually differ only slightly. For example, the most common numeric swap is "10" ↔ "15", a difference of 5, and the second most common swap is "12" ↔ "15", a difference of 3. Such small differences make it more difficult to identify which PLS is factual and which is non-factual. For instance, if we report an average heart rate, which normally ranges from 60 to 100 beats per minute, swapping to a value in the range of 50–110 may still appear plausible. By contrast, swapping to a value in the range of 500–700 is clearly impossible, making the non-factual PLS easy to identify.

Regarding negation swaps, the most common case is the insertion of an additional "not" in the non-factual PLS, which occurs 133 times. Other negation swaps are much more diverse. Adding an extra "not" is a straightforward way to create a non-factual PLS, as it can alter the factuality of a statement without changing any other words. For example, inserting "not" before "effective" to form "not effective" changes the factual meaning, without requiring a replacement such as "ineffective". As shown in Table 3.9, adding

"not" is more common than replacing words with antonyms, since many words lack a clear antonym, or the antonym may be a rare or less natural alternative.

Top Numeric Swaps		**Top Negation Swaps**	
Pair	**Count**	**Pair**	**Count**
10 ↔ 15	14	extra_not ↔ ∅	133
12 ↔ 15	11	effective ↔ ineffective	5
15 ↔ 20	9	safe ↔ unsafe	5
15 ↔ 18	8	common ↔ uncommon	3
2 ↔ 3	8	able ↔ unable	3
3 ↔ 5	8	healthy ↔ unhealthy	3
20 ↔ 25	8	insufficient ↔ sufficient	3
10 ↔ 12	7	dissimilar ↔ similar	2
5 ↔ 7	7	dissatisfaction ↔ satisfaction	2
18 ↔ 25	6	known ↔ unknown	2
15 ↔ 5	6	advantage ↔ disadvantage	2
2 ↔ 25	6	appropriate ↔ inappropriate	2
15 ↔ 25	6	expensive ↔ inexpensive	2
4 ↔ 6	6	happy ↔ unhappy	1
12 ↔ 25	5	imperfect ↔ perfect	1
14 ↔ 18	5	explained ↔ unexplained	1
18 ↔ 20	5	likely ↔ unlikely	1
2 ↔ 5	5	frequent ↔ infrequent	1
18 ↔ 24	5	aware ↔ unaware	1
24 ↔ 30	5	block ↔ unblock	1

Table 3.9: Top numeric swaps and top negation swpas

In Table 3.10, we observe once again that the swaps are quite diverse, with the most frequent pair appearing in only 7 out of the 200 total pairs. From the table, we notice that most swaps involve simply removing a biomedical term, such as "ibuprofen" → ∅ or "antibiotics" → ∅. Only a few swaps replace one biomedical term with another, for example, "carbon dioxide" ↔ "oxygen" or "dizziness" ↔ "vomiting". Thus, the type of swap observed in the example PLS pair where "estrogen" was changed to "testosterone" is relatively uncommon in the dataset.

In Table 3.11, we present the most frequent biomedical term swaps identified using the BC5CDR and CRAFT models, and in Table 3.12, we show the top biomedical term swaps identified using the JNLPBA and BioNLP13CG models. Across all four models, the most frequent swaps occur in only 2 out of the 200 total pairs, indicating that keyword swaps are highly diverse across different types of biomedical terms.

In Figure 3.6 and Table 3.13, we show the number of PLS pairs containing at least one keyword swap identified by each NER

spaCy Keyword Swap	Count
ibuprofen ↔ ∅	7
antibiotics ↔ ∅	6
amoxicillin ↔ ∅	4
aspirin ↔ ∅	4
men ↔ ∅	3
parkinson's disease ↔ ∅	3
drug ↔ ∅	3
food ↔ ∅	3
abdominal ↔ ∅	2
carbon dioxide ↔ oxygen	2
vitamin d ↔ ∅	2
methotrexate ↔ ∅	2
insulin ↔ ∅	2
dizziness ↔ vomiting	2
steroids ↔ ∅	2
helium ↔ ∅	2
vasopressin ↔ ∅	2
anxiety ↔ ∅	2
dbt ↔ ∅	2

Table 3.10: Top spaCy-keyword swaps (entity diffs across models)

model. Most keyword swaps are detected by the BC5CDR and BioNLP13CG models, with 69 and 68 pairs respectively. The CRAFT model identifies about half as many (31), while the JNLPBA model identifies the fewest, with only 9 pairs.

BioNLP13CG covers the largest number of biomedical entity types among the four models, so it is expected to identify the most keyword swaps. Many swaps are also detected by BC5CDR, which covers the "DISEASE" and "CHEMICAL" entity types. This suggests that most biomedical terms in the PLS are related to diseases and chemicals, likely because PlainFact is sourced from the Cochrane Library, which primarily focuses on the healthcare domain.

CRAFT ranks second, but still identifies only about half as many swaps as BC5CDR or BioNLP13CG. The JNLPBA model detects the fewest swaps, likely because it covers only 5 entity types ("DNA", "CELL_TYPE", "CELL_LINE", "RNA", "PROTEIN"), which are less common in the healthcare domain.

In Figure 3.7 and Table 3.14, we show the total number of entity words involved in keyword swaps identified by each NER model. The pattern is similar to that observed for the number of pairs with keyword swaps: BC5CDR and BioNLP13CG each account for 319 entity words across all PLS pairs, followed by CRAFT with 164 and

Table 3.11: Top 10 Biomedical Keyword Swaps using BC5CDR and CRAFT models

BC5CDR Model		**CRAFT Model**	
Pair	**Freq.**	**Pair**	**Freq.**
carbon dioxide ↔ oxygen	2	carbon dioxide ↔ oxygen	2
dizziness ↔ vomiting	2	estrogen and progestin ↔ hormones testosterone	1
estrogen ↔ testosterone	1	chemicals ↔ estrogen and progestin	1
obesity ↔ overweight or obesity	1	hormones ↔ hormones testosterone	1
non-hormonal ↔ overweight or obesity	1	estrogen and progestin ↔ testosterone	1
malariae ↔ plasmodium ovale	1	chemicals ↔ hormones	1
plasmodium vivax malaria ↔ vivax	1	hormones ↔ testosterone	1
p. malariae ↔ plasmodium vivax	1	plasmodium parasite ↔ plasmodium vivax	1
malariae ↔ vivax	1	plasmodium ↔ plasmodium parasite	1
p. vivax malaria ↔ plasmodium vivax malaria	1	plasmodium ↔ plasmodium vivax	1

Table 3.12: Top 10 Biomedical Keyword Swaps using JNLPBA and BioNLP13CG models

JNLPBA Model		**BioNLP13CG Model**	
Pair	**Freq.**	**Pair**	**Freq.**
six-capsule ↔ three-rod	1	carbon dioxide ↔ oxygen	2
non-drug ↔ two-thirds	1	estrogen ↔ testosterone	1
cd4+ cell ↔ cd4+ cell	1	mouth ↔ nose	1
five- ↔ ten-day	1	juice ↔ water	1
eight-day ↔ ten-day	1	falciparum ↔ falciparum line	1
calcium molecules ↔ small muscle protein	1	falciparum ↔ plasmodium	1
fgf ↔ vascular endothelial growth factor	1	falciparum line ↔ falciparum line	1
fibroblast growth factor ↔ vascular endothelial growth factor	1	falciparum line ↔ plasmodium	1
colony stimulating factors ↔ gm-csfs	1	bacillus calmette-guérin ↔ doxorubicin	1
ors $\leq$ ↔ ors $\leq$ 270 mosm/l	1	doxorubicin ↔ mitomycin	1

JNLPBA with only 55. Since there are only 200 PLS pairs in total, this indicates that multiple entity words across different entities are

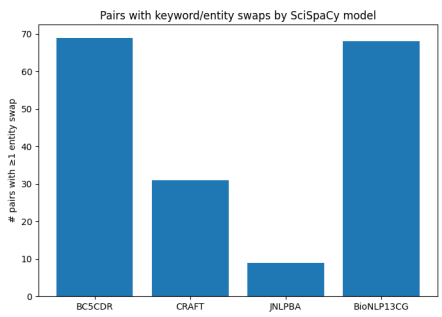


Figure 3.6

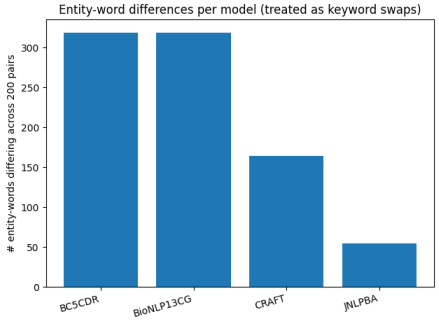


Figure 3.7

Table 3.13: Entity swap counts across biomedical NER models

Model	**Pairs with Entity Swap**
CRAFT	31
JNLPBA	9
BC5CDR	69
BioNLP13CG	68

often swapped within a single pair.

This observation is further supported by the "Avg Words / Affected Pair" column in Table 3.14. For all four models, the average number of entity words per affected pair is at least 1.4, and for BC5CDR and BioNLP13CG it exceeds 2.3. This demonstrates that in most PLS pairs containing keyword swaps, multiple entity words are exchanged between the factual and non-factual versions.

Table 3.14: Per-Model Entity-Words-Difference Summary

Model	**Number of Words**	**Average Words / Pair**	**Average Words / Affected Pair**
BC5CDR	319	1.595	2.435
BioNLP13CG	319	1.595	2.363
CRAFT	164	0.820	1.929
JNLPBA	55	0.275	1.410

In Table 3.15, we present the top 20 words that are unique to factual and non-factual PLS, respectively. We observe that there are more unique words in the non-factual PLS compared to the factual PLS: the most frequent unique word in the non-factual PLS appears 47 times, whereas the most frequent unique word in the factual PLS appears only 25 times.

Many of the unique words in both factual and non-factual PLS are numbers, such as "two", "four", "six", "10", "12", and "15" in

factual PLS, and “five”, “15”, “three”, “25”, “eight”, and “seven” in non-factual PLS. This is expected, as there are numerous numeric swaps between factual and non-factual PLS.

In addition, some words such as “large”, “high”, and “reduce” appear in both factual and non-factual PLS. These correspond to the antonym swaps identified earlier, as shown in Table 3.8.

Overall, while this table does not provide fundamentally new insights, it further supports our previous findings and reinforces the validity of our classification of swap types.

Table 3.15: Top 20 words unique to factual vs. nonfactual Target_Sentence

Factual		**Nonfactual**	
Word	**Freq.**	**Word**	**Freq.**
two	25	five	47
four	25	not	43
six	21	cannot	34
10	20	increase	33
reduce	18	15	32
common	16	three	30
three	16	25	26
more	16	eight	24
small	15	seven	23
can	15	six	22
less	15	less	22
five	15	large	21
eight	14	ten	21
increased	14	decrease	18
12	13	decreased	18
15	13	does	18
low	13	worse	17
2	12	increased	17
4	12	30	17
high	12	high	16

Note that here we are comparing factual and non-factual PLS without directly considering the original scientific abstract. However, since the factual PLS is generated from the scientific abstract, swaps between factual and non-factual PLS also imply swaps between the non-factual PLS and the abstract. Therefore, when using this dataset to evaluate the factuality abilities of models, we are essentially testing whether the model can identify these swaps between the PLS and the scientific abstract.

In Table 3.16 and Figure 3.8, we report the average number of words and tokens in the `Target_Sentence` (PLS) and the `Original_Abstract`, as well as the average differences between them. Since the difference in word and token counts between factual and

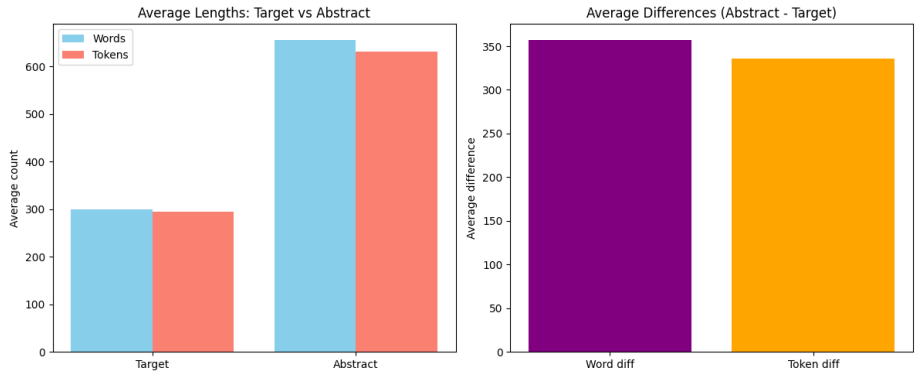


Figure 3.8: Average word and token counts for Target_Sentence and Original_Abstract, along with their differences.

non-factual PLS is small, we ignore it here.

In text simplification, the simplified text is not necessarily shorter than the original. However, since the PlainFact dataset is derived from the Cochrane Library, which was not specifically designed for text simplification, the relative lengths of the PLS and the abstracts may not follow typical simplification patterns. From Table 3.16 and Figure 3.8, we observe a significant difference: the average number of words in the scientific abstract is 656.32, more than double the 298.78 words in the PLS. Similarly, the average number of tokens in the abstract is 630.63, more than double the 294.60 tokens in the PLS. This shows that PLS are systematically shorter than scientific abstracts, suggesting that much of the complex phrasing in the abstracts can be simplified into more concise alternatives. As illustrated in Table 1.1, using an example from Cochrane (the source of PlainFact), numerical details that appear in the scientific abstract are omitted in the corresponding PLS.

Table 3.16: Averages and Differences between Abstract and Target

Metric	**Value**
Averages	
Target_Sentence (words)	298.78
Target_Sentence (tokens)	294.60
Original_Abstract (words)	656.32
Original_Abstract (tokens)	630.63
Differences (Abstract - Target)	
Average word difference	357.54
Average token difference	336.04

In the original PlainFact paper, when selecting the 200 instances from the Cochrane library, the authors only mentioned that they

used simplicity metrics to select the most readable examples.

We conducted a small investigation into the areas of the biomedical domain covered by PlainFact. Specifically, we used the same NER models from SciSpaCy listed in Table 3.6. We searched the scientific abstracts from PlainFact for entities identified by these models. If an abstract contained only one type of entity, we assumed that the example in PlainFact was focused on that entity. The entities themself represent different aspects of the biomedical domain. For example, BC5CDR defines two entity types: *Disease* and *Chemical*. Thus, if an abstract contained only the *Chemical* entity type, we interpreted that example as being primarily about chemicals.

Table 3.17: Primary Class Distribution Across Four SciSpaCy NER Models

CRAFT		**JNLPBA**	
Primary Class	**Count**	**Primary Class**	**Count**
more than one	146	more than one	141
none	31	PROTEIN	26
SO	10	none	22
CHEBI	6	DNA	7
GGP	3	CELL_LINE	3
TAXON	2	CELL_TYPE	1
GO	1	–	–
CL	1	–	–
BIONLP13CG		**BC5CDR**	
Primary Class	**Count**	**Primary Class**	**Count**
more than one	200	more than one	192
–	–	DISEASE	4
–	–	none	3
–	–	CHEMICAL	1

In Table 3.17, we show the number of scientific abstracts classified into different entity groups by different NER models. The row labelled "more than one" represents the number of scientific abstracts that contain more than one type of entity according to that model, while "none" represents the number of scientific abstracts that did not contain any entities from that model.

Here, we see that for BIONLP13CG and BC5CDR, almost all of the scientific abstracts contain more than one entity. For CRAFT and JNLPBA, approximately $\frac{3}{4}$ of the scientific abstracts contain more than one entity.

This result is discouraging, as it means we cannot properly use these models to provide a finer classification of the examples from

PlainFact. Consequently, we cannot use this analysis to identify which biomedical topics represent weaknesses for the models.

In Table 3.18, we show the average, median, and maximum number of each type of swap per pair of factual and non-factual PLS. In Figure 3.9, we present a graphical representation of the average number of swaps per pair. We observe that, on average, each pair has significantly more numeric swaps (about 8 per pair) and the fewest negation swaps (about 1 per pair). Since numeric swaps are the easiest to perform, simply changing a number will often flip the factuality of the sentence.

We also note that the difference between the maximum and minimum number of swaps is quite large, with the highest difference being 90 (for numeric swaps). This variation highlights that the characteristics of factual errors differ substantially across pairs.

Table 3.18: Swap Counts per Pair

Type of Swap	**Average**	**Median**	**Max**	**Min**
Numeric Swaps	7.92	1.00	91	0
Antonym Swaps	1.49	1.00	7	0
Negation Swaps	0.97	1.00	5	0
Keyword Swaps	3.29	2.00	24	0
Total Swaps	13.66	9.00	94	0

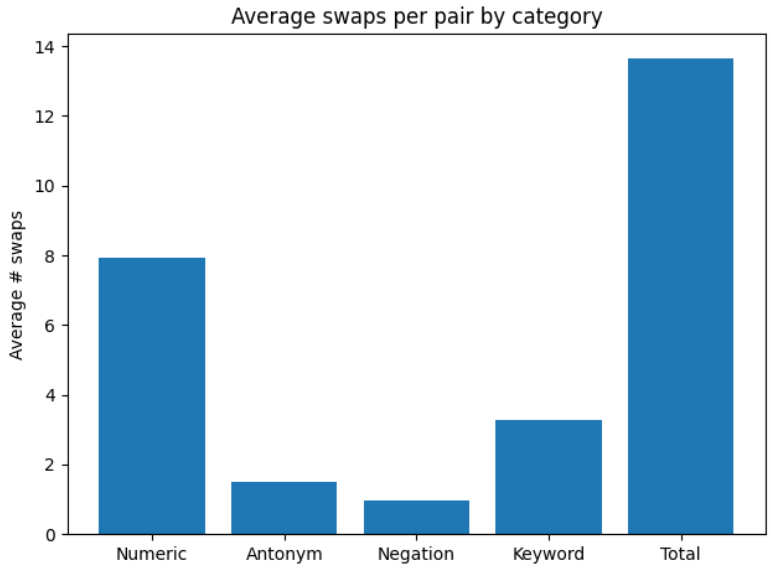


Figure 3.9

4 Experiment

4.1 Computational Resource and API

We had access to three computational resources for this project: Avon and Blythe from Warwick's HPC cluster, and the DCS Kudu server.

The most powerful server is **Blythe**, which provides three NVIDIA L40 GPUs, each with 48GB of RAM, based on the NVIDIA Ada Lovelace architecture.

The second is the **Kudu** server from DCS, which has three partitions available. *Gecko* is the most powerful partition, offering three NVIDIA A10 GPUs, each with 24GB of RAM, based on the NVIDIA Ampere architecture (one generation behind the L40). *Falcon* provides a single NVIDIA A5000 GPU with 24GB of RAM, also Ampere-based, while *Edge* offers a single NVIDIA RTX 2080Ti GPU with 11GB of RAM. For this project, we only used the Gecko partition, as Falcon and Edge provide only one GPU each, which was insufficient for most models. Additionally, we were only permitted to use one partition at a time, so there was no advantage to switching between them.

The third resource is the **Avon** server, which provides three NVIDIA Quadro RTX 6000 GPUs, each with 24GB of RAM, based on the NVIDIA Turing architecture. The GPUs on Avon are four years older than the L40s on Blythe and are two generations behind, with each GPU offering only half the RAM of an L40.

It is also important to note that each server provides different storage capacity. Since we run LLMs locally, the models must first be downloaded, which requires substantial disk space. **Avon** provides the largest storage capacity with 2TB, **Blythe** offers 300GB, and **Kudu** provides only 100GB. Both Blythe and Kudu use dual-storage systems, requiring us to manually specify the model download and reading paths.

Finally, compute resources are limited on each server. When usage is high, compute jobs are queued, sometimes resulting in long wait times. **Avon** is the most crowded server, and we often had to wait in queue. **Blythe** and **Kudu** were generally less congested, with little or no queuing. To maximise efficiency, we relied primarily on Blythe and Kudu.

API We used two libraries to help set up and run the models: Ollama and Hugging Face. Ollama is primarily designed for model inference on local machines and is not well suited for running jobs

on batch systems such as the Warwick servers. Hugging Face, on the other hand, is more versatile and works well across different platforms. Therefore, we primarily relied on Hugging Face's `transformers` API for using the models.

According to Hugging Face's documentation, there are different ways to download models. Since we needed to manually specify the download location, we used the `snapshot_download` method to fetch models and load them from a local directory. This approach worked until August, after which downloads often stalled partway. We suspect this was due to high demand on Hugging Face's servers, which slowed or limited the snapshot method. As downloading via the `transformers` command line interface (CLI) continued to work reliably, we switched to using the CLI for the remainder of the project.

Additionally, while `transformers` provides a `device_map="auto"` option to automatically distribute models across multiple GPUs, in practice this often led to out-of-memory (OOM) errors. For example, even relatively small models such as Qwen3_30B (30B parameters) would trigger OOM errors on the Gecko server with 3 × 24GB GPUs. To address this, we had to manually configure the memory allocation of models across multiple GPUs.

Here we provide an example of how we set up Qwen_think on Avon. For all models on Huggingface uses the same procedure, so we only show one example. We used the `AutoTokenizer` to load the tokenizer for Qwen_think, which is also required later when switching off the reasoning ability to use Qwen_nothink. The model was loaded with `AutoModelForCausalLM`, and we applied the preset `max_memory`. Since Avon provides GPUs with 24 GB of RAM each, in theory we could set `max_memory` to 22 GB (≈ 23 GB), but this sometimes leads to out-of-memory (OOM) errors. To provide headroom, we instead set it to 20 GB. We use the same `max_memory` setting on Kudu as well. On Blythe, we increase `max_memory` to 44 GB, since each L40 GPU has 48 GB of RAM.

We then used the `pipeline` function to wrap the model and tokenizer together. The advantage of using `pipeline` is that the model's output is automatically returned in a structured form. If we were to use `model.generate` directly, we would still need to reapply the tokenizer to obtain structured output. Even if the input prompt is pre-processed with the tokenizer, the `pipeline` method recognises this and avoids re-tokenisation.

For Qwen_think, setting `max_new_tokens` to 512 is sufficient. However, for some reasoning models such as OSS_20_high, much larger values are required to generate complete answers, and the

exact setting must be checked manually.

We directly read PlainFact from Hugging Face's repository and convert it into a dataset, allowing us to process it in batches to maximise efficiency.

The results of the models are stored in a CSV file, which can then be easily processed using Pandas. This allows straightforward analysis with the DataFrame's convenient tabular structure.

```
model_name = "Qwen/Qwen3-32B"
cache_dir = "/dcs/large/u2151813/cache"

tokenizer = AutoTokenizer.from_pretrained(
    model_name,
    cache_dir=cache_dir,
    trust_remote_code=True
)

n_gpus = torch.cuda.device_count()
max_memory = {i: "20GiB" for i in range(n_gpus)}
max_memory["cpu"] = "40GiB"

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    cache_dir=cache_dir,
    dtype="auto",
    device_map="auto",
    trust_remote_code=True,
    max_memory=max_memory,
)

pipe = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    device_map="auto",
    dtype="auto",
    trust_remote_code=True,
)

PlainFact = pd.read_csv("hf://datasets/uzw/PlainFact-summary
    /summary_level.csv")
dataset = Dataset.from_pandas(PlainFact)

texts = [
    tokenizer.apply_chat_template(
        msgs,
        tokenize=False,
        add_generation_prompt=False,
        enable_thinking=False
    )
outputs = pipe(texts, max_new_tokens=512)
```

Here we show how we used Ollama to run inference with DS_7.

We note that Ollama provides far fewer options for customisation compared to Hugging Face, and there is no `enable_thinking` parameter in the `chat` method. However, Ollama is very user-friendly, and we used it for its simplicity and quick implementation.

To obtain results, we only need to call the `chat` method from Ollama, after which the model output can be accessed via `response.message.content`.

```
from ollama import chat, ChatResponse

model = "deepseek-r1:7b"

response: ChatResponse = chat(model=model, messages=[{'role'
    : 'user', 'content': list_prompt}])
response_content = response.message.content
```

4.2 Dataset

Here we will discuss the dataset that we used for this project.

4.2.1 PlainFact

The PlainFact dataset [25] is a sentence-level dataset consisting of over 2000 pairs of scientific abstracts and plain language sentences (PLS). The dataset is derived from the CELLS corpus [8], which is the largest dataset of scientific abstracts paired with PLS, all written by the same authors. CELLS is sourced from 12 biomedical journals, while the PlainFact dataset specifically uses data from the Cochrane Database of Systematic Reviews (CDSR) [12]. The Cochrane Library contains systematic reviews that support medical decision-making in the healthcare domain [15]. These reviews represent the highest level of evidence for clinical decision-making, which helps ensure that the dataset remains factually reliable. To further narrow the scope, the authors selected the top 200 abstract–PLS pairs with the highest readability, as measured by several metrics, including the Flesch–Kincaid Grade Level (FKGL) [10].

It is important to note that the original abstracts and PLS from the Cochrane Library are written at the *paragraph level*. In contrast, the PlainFact dataset contains paragraph-level scientific abstracts paired with sentence-level PLS. The authors did not explicitly describe how they segmented the plain language summaries into sentences. We assume they may have used a straightforward method, such as regular expressions, to split paragraphs into sentences. Thus, while only 200 abstract–PLS pairs were selected from the Cochrane Library, segmenting the plain language summaries into sentences resulted in over 2000 sentence-level abstract–PLS pairs.

Once the plain language summaries were segmented, the authors employed annotators to manually align each plain language sentence to its corresponding abstract sentence. They also annotated the *functional purpose* of each plain language sentence (e.g., background, definition), as well as whether it represented a *direct simplification* or an *elaborative explanation*, which we discuss further in Section 2.1. However, in the official PlainFact Hugging Face repository, there is no information about functional purposes or sentence-level alignments. Only the annotation distinguishing direct simplifications from elaborative explanations is preserved.

Another important detail is that each pair in PlainFact contains both a *factual* and a *non-factual* plain language sentence. The factual PLS is directly sourced from the Cochrane Library, but the method by which the non-factual PLS was created is not described by the authors.

Since our goal is to evaluate the factuality of LLMs on paragraph-level (i.e., summary-level) data, the sentence-level PlainFact dataset does not fully meet our needs. Fortunately, the authors also provide a *summary-level* dataset based on PlainFact, called "PlainFact-Summary". In this dataset, each plain language summary is constructed by concatenating multiple plain language sentences from the original PlainFact dataset, thereby recreating the original plain language summaries from the Cochrane Library. Like PlainFact, PlainFact-Summary also contains one factual and one non-factual plain language summary for each abstract.

However, concatenation destroys the sentence-level annotations of direct simplification and elaborative explanation. As a result, the PlainFact-Summary dataset contains only four columns:
`Original_Abstract`, which corresponds to the scientific abstract from Cochrane; `Target_Sentence`, which contains the plain language summary; `Factual`, which indicates whether the summary is factual; and `Target_Summary_ID`. There are 400 rows in total, because the factual and non-factual summaries are stored separately, effectively doubling the 200 pairs from the Cochrane Library. Thus, each abstract corresponds to two rows, one containing the factual PLS and the other containing the non-factual PLS.

For convenience, throughout this report we will refer to the PlainFact-Summary dataset simply as *PlainFact*, and PLS will denote *plain language summary*.

4.2.2 Cochrane

The Cochrane dataset is a large corpus of biomedical text, but it is unbalanced: every scientific abstract has a corresponding factual simplified text, while there are no examples of non-factual simplified texts paired with the abstracts.

This dataset is sourced from the Cochrane Database of Systematic Reviews, as part of the CELLS corpus [8], where a wide range of clinical topics are covered. We can see that it shares the same source as PlainFact, but the construction method is different. The authors of the Cochrane dataset observed that some summaries from the Cochrane library are equivalent to simplified versions of the scientific abstracts, preserving the claims and content while differing in structure. These summaries were then used as simplified texts to construct the dataset.

However, the Cochrane library's plain language summaries (PLS) were not written for the purpose of creating parallel simplified versions of scientific abstracts. As a result, the quality of the dataset heavily depends on the authors' selection method for summary abstract pairs, which is critical to ensure that the summary can indeed be considered a valid simplification of the abstract. The authors employed a simple rule-based method for selecting the simplified examples, resulting in 4459 pairs.

Nonetheless, investigations into the quality of this dataset reveal issues. Using a factual error taxonomy [6] and manual inspection of the first ten samples, all ten examples exhibited different types of factual inconsistencies. This demonstrates that the rules defined for pair extraction are not sufficiently reliable for our purpose of evaluating factuality in simplified biomedical text.

In addition, this dataset is unbalanced: all 4459 examples contain only factual PLS paired with the scientific abstract. This imbalance reduces the reliability and effectiveness of using the dataset to evaluate model performance.

4.3 QA Result

In Table 4.1, we present the results obtained from the QA pipeline. The "yes" and "no" columns represent the number of examples from the dataset classified as non-factual and factual, respectively. Note that we used the first 250 examples from the dataset when running the DS_7 model, and only 200 examples for DS_32, since the larger model takes significantly longer to run. As these two models were executed on our own laptop, we limited the sample size to save time.

Here, we compare DS_7 and DS_32 as base models to investigate the impact of model size on the performance of the QA system. The Cochrane dataset was used in this experiment.

model	yes	no	unknown
DS_7	193	52	5
DS_32	192	8	0

Table 4.1: Classification result from the pipeline with two model size

From the results, we can see that the two models produced a similar number of "yes" instances but differed in the number of "no" instances. This difference is due to the unequal number of test examples used for the two models, which makes direct comparison between them less rigorous. Nevertheless, the results still show the trend that both models classified the majority of examples as non-factual. Since the Cochrane dataset contains only factual simplified texts, this indicates that the performance of our QA system is very poor.

However, as discussed in Chapter 4.2.2, manual inspection revealed that all of the first ten examples contained factual errors. This suggests that the dataset itself is unreliable, which means the results in the table are also unreliable and do not accurately reflect the performance of our QA system.

In addition, the system is rather inefficient. We currently need to run inference five times per example, whereas efficiency could be greatly improved by reconstructing the prompts and combining different stages of the system. By doing so, we could design a much more efficient system that requires only a single inference to execute the full evaluation. Below, we show the prompt for this new, more condensed version of the pipeline, where inference is performed only once.

```
list_prompt = "List all the facts we explicitly know
from the statement_1, then list all the facts we
explicitly know from the statement_2.
Make each fact as atomic as possible. \n Then compare
the two lists of fact from statement_1 and statement_2.
If there are facts that is in Statement_2 but not in
Statement_1, answer 'Yes', esle 'No'. \n
Statement_1: {org} \n
Statement_2: {tag}".format(
org = df.iloc[i]["Original_Abstract"],
tag = df.iloc[i]["Target_Sentence"])
```

In Table 4.2, we show the results of our QA system on PlainFact, using one reasoning model and one non-reasoning model. Here, DS_7

is the reasoning model, and Llama3.1_I is the non-reasoning model.

Since the dataset is balanced, accuracy is a suitable metric and provides an intuitive way to compare results. We observe that both models achieve similar performance on the QA system, with results close to random guessing (56.00% for DS_7 and 56.48% for Llama3.1_I). This shows that our system is not effective, and that the reasoning ability of the models does not improve system performance.

We also observe that both models make significantly more No to Yes errors, indicating that they tend to classify non-factual simplified text as factual. This suggests that the system is not effectively capturing the factuality swaps in the dataset. The problem may stem from the models failing to reliably extract facts from the original and simplified texts, or from difficulties in comparing the two fact lists to identify mismatches.

Finally, we note that almost $\frac{1}{4}$ of the examples processed by Llama3.1_I resulted in invalid outputs, where the model failed to answer with either "yes" or "no". This indicates that the reasoning model (DS_7) demonstrates better alignment than the non-reasoning model.

Table 4.2: Classification result from the pipeline with two reasoning and non reasoning models

Metric	**DS_7**	**Llama3.1_I**
Correct Predictions	224	183
Accuracy (%)	56.00	56.48
Yes→No	38	22
No→Yes	138	119
Invalid/Missing	0	76

4.4 Evaluation of Factuality

In this section, we compare the performance of recent reasoning models with non-reasoning models in evaluating the factuality of simplified text. We report results from two reasoning models and three non-reasoning models.

The reasoning models include **Marco-O1** and **Qwen_think**, while the non-reasoning models are **Qwen_nothink**, **Llama3**, and **Mistral**.

Since the Cochrane dataset was shown to be less reliable than PlainFact, we use only the PlainFact dataset for the following experiments.

For evaluation, we used the following prompt: the model was asked whether the simplified text is factual. If it is factual, the model should answer "Yes"; otherwise, it should answer "No". The same prompt was applied to all models. Here, the variables "o" and "t" in the prompt represent the original text (scientific abstract) and the simplified text (PLS), respectively.

```
f"Original Abstract: {o}\n"
f"Simplified Abstract: {t}\n"
"Request: Is the Simplified Abstract a factual
    simplification of the "
"Original Abstract? "
"If so, answer Yes; otherwise, answer No.\n"
"Answer:"
```

In Table 4.3, we show the results of the models. We use accuracy as the evaluation metric, since the dataset is perfectly balanced, making this metric appropriate.

We observe that the two non-reasoning models, Mistral and Llama3, achieve near-chance performance, both with an accuracy of around 50%. This shows that the performance of these two non-reasoning models is very poor, as they are unable to identify the factual errors and swaps between the PLS and the scientific abstract.

The two reasoning models demonstrate greater capability than Llama3 and Mistral. Qwen_think performs slightly better than Marco-O1 (84.75% compared to 78.75%).

We also report the performance of Qwen_nothink, which is the best-performing model in our evaluation, achieving 91% accuracy. This result contradicts the trend we observed in the other four models, where reasoning models outperformed non-reasoning models. In addition, Qwen_nothink achieves better results than Qwen_think. This suggests that reasoning models may not actually perform better than non-reasoning models due to their reasoning ability, but rather due to their inherent model strength. Reasoning ability may not always be helpful, and in some cases, it can even lead to worse answers.

The concept of *UnderThinking* [23] describes the phenomenon whereby, during the reasoning step, a model tends to deviate from one chain of reasoning to another before the reasoning path is fully developed. This can cause a potentially correct reasoning path to be discarded prematurely. A similar issue may also be present in the reasoning traces of Qwen_think, leading to its reduced performance compared to Qwen_nothink.

In Table 4.4, we present the confusion matrices of the five models alongside their F_1 scores.

Model Type	Model	Correct Classify	Mis-Classify	Accuracy (%)
Reasoning	Qwen_think	339	61	84.75
	Marco_O1	315	85	78.75
Non-Reasoning	Qwen_nothink	364	36	91.00
	Mistral	203	197	50.75
	Llama3	200	200	50.00

Table 4.3: Results from the models.

From the table, we can see that Mistral and Llama3 have high false-positive rates, meaning that these models often predict "yes" when the label is actually "no", compared to the reasoning models. They also show high true-positive counts, indicating that they tend to classify most examples as factual. As a result, these two models struggle to distinguish between factual and non-factual PLS.

We also observe that both Qwen models perform well at detecting non-factual PLS, achieving a 0% false-positive rate. The drop in accuracy for Qwen_think can be attributed to its higher false-negative rate, where more factual instances are classified as non-factual.

Model Type	**Model**	**Actual Yes**		**Actual No**		**F_1 Score**
		Pred. Yes	**Pred. No**	**Pred. Yes**	**Pred. No**	
Reasoning	Qwen_think	139	61	0	200	0.820
	Marco_O1	175	25	60	140	0.805
Non-Reasoning	Qwen_nothink	164	36	0	200	0.901
	Mistral	200	0	197	3	0.670
	Llama3	168	32	168	32	0.627

Table 4.4: Overview of confusion matrices and F_1 scores for all models

We also attempted to use DeepSeek on this task, but we were unable to run the model on the Kudu server due to limited experience with model memory allocation. As an alternative, we tried Open-R1, the community replication of DeepSeek-R1. We used the same prompt as with the other five models and applied it to PlainFact. At this stage, however, we discovered the instability of the model and ultimately abandoned its use.

We also examined the possibility of data contamination. PlainFact is sourced from the Cochrane Database of Systematic Reviews, which was originally published in 2016. It is possible that the Cochrane library is present in the training sets of some models, potentially inflating their performance on this benchmark.

Since we are using open-sourced language models, we can directly calculate the perplexity of the models on PlainFact.

Perplexity is a measure of how well a language model predicts

text. Formally, given a language model, perplexity captures, on average, how difficult it is for the model to predict the next word. A good language model should assign higher probabilities to the actual next word, and thus a lower perplexity is preferred. Perplexity is inversely related to probability: higher probability corresponds to lower perplexity.

In short, perplexity measures how "good" a language model is, on average per word, when presented with a text.

Perplexity of a sequence of N words W, PP(W):

$$\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\
\text{(Chain rule)} \quad &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P\big(w_i \mid w_1, \dots, w_{i-1}\big)}} \\
\text{(Bigrams)} \quad &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P\big(w_i \mid w_{i-1}\big)}} \,.
\end{aligned}$$

For our use of perplexity, the interpretation is as follows: a lower perplexity indicates that, when given an initial token from a passage in PlainFact, the model assigns a high probability to generating the passage exactly as it appears in PlainFact. This suggests that the model may have encountered the passage during training, implying potential data contamination.

In Table 4.5, we report the perplexity of all models on both the PLS and the scientific abstracts from PlainFact. Since our models are tasked with predicting the factuality of the PLS, we focus on the perplexity of the PLS. If a model has seen the PLS during training, it would effectively "know" whether the PLS is factual or not.

From the table, we observe that for all models except Qwen_think, the perplexity on PLS is around 7, which is very low compared to Qwen_think, whose perplexity is above 30,000. A similar pattern is found for the scientific abstracts, where Qwen_think again has much higher perplexity than the other models. This suggests that all models except Qwen_think are likely to have seen the PlainFact data during training. However, despite this potential data contamination, most models still perform poorly at classifying the factuality of the PLS. This demonstrates that even if a model has encountered the dataset, it does not necessarily mean it will perform well at

classifying factuality in simplified biomedical text under our experimental setup.

Model	Perplexity on scientific abstract	Perplexity on PLS
Marco_O1	4.48	7.86
Lamma3	4.26	7.38
Mistal	3.82	6.66
Qwen_think	3741.61	37486.91

Table 4.5: Perplexity of the models

We cannot examine how the reasoning ability of `Marco_O1` affects model performance in the same way as we did with Qwen. This is because no method is provided to disable the reasoning ability of `Marco_O1`, whereas for Qwen3 we can turn off reasoning by setting the parameter `enable_thinking` to `False` in the `apply_chat_template` function.

In table 4.6, we show the distribution of correctly answered questions of the models. Here we see that for all the questions, there is at least one model who answered correctly. This show that there is no "impossible" questions for the models. However, there are some "easy" questions. We see that out of total 400 questions, there are 54 questions which the model all answered correctly.

Models Correct	**Count**
1	22
2	95
3	130
4	99
5	54

Table 4.6: Agreement distribution across models

Here we show the list of question ids which all the models answered correctly.

```
0, 3, 5, 8, 11, 16, 18, 20, 21, 26, 35, 46, 47, 51, 62,
65, 80, 83, 85, 88, 89, 92, 99, 100, 101, 102, 104, 105,
109, 111, 116, 120, 121, 123, 125, 126, 128, 129, 133,
136, 139, 143, 146, 147, 150, 152, 158, 163, 164, 172,
187, 190, 193, 199
```

We observe that all of the questions which every model answered correctly come from the first half of the dataset, where all examples are factual. This indicates that all models have a tendency to answer "yes". In other words, the models tend to assume that

the PLS is factual. However, as we will see later, this behaviour reflects a general bias towards answering "yes", regardless of the actual truthfulness of the PLS.

In Table 4.8, we show the top ten model pairs with the highest agreement scores. We report two metrics: *Agreement*, which is the percentage of questions on which two models gave the same answer, and *Kappa*, which is Cohen's kappa score between the two models. Cohen's kappa is a statistical measure of inter-rater reliability, showing the agreement between two classifiers that categorise N items into C mutually exclusive categories. The formula for Cohen's kappa is:

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the relative observed agreement between two models, and p_e is the hypothetical probability of chance agreement (i.e., the expected agreement by chance).

The kappa score ranges from 0 to 1, where 1 indicates perfect agreement and 0 indicates no agreement. A more detailed interpretation of kappa values is provided in Table 4.7. Note that in practice, kappa scores can also be negative, as observed in Table 4.8. Although this case is not explicitly covered in Table 4.7, negative values can be interpreted as disagreement, meaning that the observed agreement is worse than expected by chance [13].

Value of Kappa	**Level of Agreement**	**% of Data that are Reliable**
0–.20	None	0–4%
.21–.39	Minimal	4–15%
.40–.59	Weak	15–35%
.60–.79	Moderate	35–63%
.80–.90	Strong	64–81%
Above .90	Almost Perfect	82–100%

Table 4.7: Interpretation of Cohen's Kappa values [13]

We observe from Table 4.8 that while the top pairs achieve relatively high percentage agreement—with the top five pairs all exceeding 50%—the kappa scores tell a different story. Only the first pair, Llama3 and Mistral, shows moderate agreement ($\kappa = 0.68$), whereas all other pairs show no meaningful agreement, and the bottom five pairs even show disagreement, with negative κ values. This indicates that, apart from Llama3 and Mistral, which genuinely produce similar predictions, the apparent high percentage agreement among other pairs is largely due to chance, rather than reflecting consistent agreement on the factuality of the PLS.

It is important to note that the PlainFact dataset we used is perfectly balanced. Therefore, the high percentage agreement cannot be attributed to class imbalance in the dataset.

Table 4.8: Model Similarity (Top Pairs)

Model 1	**Model 2**	**Agreement**	**Kappa**
Mistral	llama3	0.840	0.680
Marco_O1	Qwen_nothink	0.758	0.082
Marco_O1	Mistral	0.588	0.175
Marco_O1	llama3	0.562	0.125
Qwen_nothink	Qwen_think	0.548	0.134
Marco_O1	Qwen_think	0.480	-0.008
llama3	Qwen_nothink	0.435	-0.130
llama3	Qwen_think	0.412	-0.175
Mistral	Qwen_nothink	0.410	-0.180
Mistral	Qwen_think	0.382	-0.235

This shows that different models have significantly different performance on the PlainFact dataset. Another way to interpret this is that Llama3 and Mistral capture patterns in the dataset in a similar way. From Table 4.3, we also see a significant gap in accuracy relative to the ground truth between Llama3 (accuracy = 68.72%) and Mistral (accuracy = 50.25%). This suggests that the two models make the same mistakes on some examples in the dataset. Meanwhile, the pairs with negative κ appear to emphasise different features than the others.

In Table 4.9, we show the number of examples from PlainFact that only a given model answered correctly (`Only_Model_Correct`) and the number of examples that only that model answered incorrectly (`Only_Model_Wrong`). We observe that `Qwen_nothink` is the most outstanding model, with significantly more unique correct answers than the others. By contrast, `Qwen_think` has significantly more unique wrong answers, which is consistent with its low overall accuracy. This demonstrates that the reasoning ability of Qwen3 did not provide an advantage in this task, as the version without reasoning (`Qwen_nothink`) significantly outperformed the version with reasoning
(`Qwen_think`), both in overall accuracy and in unique correct answers.

From Chapter 3.2, we know that PlainFact separates the factual and non-factual PLS for a single scientific abstract into two rows. In order to compare model performance on factual and non-factual PLS side by side, we group the PLS pairs together and then compare how the models perform on the factual and non-factual examples.

Table 4.9: Unique Model Strengths/Weaknesses

Model	Only_Model_Correct	Only_Model_Wrong
Marco_O1	0	2
Mistral	3	12
llama3	0	11
Qwen_nothink	19	3
Qwen_think	0	71

In Table 4.10, we show how each of the five models performed when we paired the factual and non-factual PLS together. We list four possible outcomes for each pair:

- `both_correct`: the model answered correctly for both the factual and non-factual PLS. Intuitively, this means the model correctly identified the factual PLS as factual and the non-factual PLS as non-factual.
- `both_wrong`: the model answered incorrectly for both. Intuitively, this means the model incorrectly labelled the factual PLS as non-factual and the non-factual PLS as factual.
- `only_no_correct`: the model answered correctly only for the non-factual PLS. Intuitively, this means the model classified both the factual and non-factual PLS as non-factual.
- `only_yes_correct`: the model answered correctly only for the factual PLS. Intuitively, this means the model classified both the factual and non-factual PLS as factual.

Note that we cannot directly compare the counts in Table 4.10 with the overall accuracy results in Table 4.3, since the total number of pairs is 200, whereas the total number of examples is 400. Nevertheless, the overall trends in model performance should be consistent between the two tables.

The most desirable model is one with high `both_correct` and low `both_wrong`, as this corresponds to high overall accuracy. Here, we see that `Marco_O1` and `Qwen_nothink` perform best, with `Marco_O1` achieving 115 `both_correct` and 0 `both_wrong`, and `Qwen_nothink` achieving 164 `both_correct` and 0 `both_wrong`.

This table also confirms the presence of a "yes"-bias in some models. For example, Mistral and Llama3 have significantly higher `only_yes_correct` than `only_no_correct`. Mistral takes this to the extreme: for all 200 pairs it classifies every example as "yes", resulting in 200 `only_yes_correct` and 0 `only_no_correct`. This is consistent with its overall accuracy (50.25%), as half of the examples

are factual; by always answering "yes", it achieves 50.25% accuracy. Llama3 also demonstrates a "yes"-bias, with 141 `only_yes_correct` and only 5 `only_no_correct`.

Among the reasoning models, `Marco_O1` shows a slight "yes"-bias, with 60 `only_yes_correct` and 25 `only_no_correct`. By contrast, `Qwen_think` shows a slight "no"-bias, with 23 `only_yes_correct` and 70 `only_no_correct`. However, the reliability of the two models is very different: `Marco_O1` achieves 0 `both_wrong`, while `Qwen_think` produces 59 `both_wrong`, making `Marco_O1` the most balanced and reliable model overall.

Comparing `Qwen_think` and `Qwen_nothink`, we see that the reasoning ability of Qwen3 negatively affects performance. From Table 4.3, `Qwen_nothink` has higher accuracy than `Qwen_think` (91% vs. 84.75%). From Table 4.10, we also see that `Qwen_nothink` achieves 0 `both_wrong` compared to 59 for `Qwen_think`, and significantly more `both_correct` (164 vs. 48). This shows that reasoning did not help in evaluating factuality. Both `Qwen_nothink` and `Qwen_think` show a tendency to answer "no", but reasoning actually made this bias worse: `Qwen_think` has more `only_no_correct` than `Qwen_nothink` (70 vs. 36). Moreover, reasoning also introduced 23 `only_yes_correct` for `Qwen_think`, whereas `Qwen_nothink` has none. This further demonstrates that the reasoning ability of Qwen3 degrades performance in this task.

Table 4.10: Pair outcome counts per model (BC = both_correct, BW = both_wrong, ONC = only_no_correct, OYC = only_yes_correct).

Outcome	**Marco O1**	**Mistral**	**Llama 3**	**Qwen nothink**	**Qwen think**
BC	115	0	27	164	48
BW	0	0	27	0	59
ONC	25	0	5	36	70
OYC	60	200	141	0	23

If we combine this with the observations made in Chapter 3.2, we know that the non-factual PLS in PlainFact are created using four types of swaps. Using this dataset therefore essentially amounts to testing whether the models can identify these swaps. From this perspective, we can conclude that `Marco_O1` is the best model at identifying such swaps.

By contrast, models with a "yes"-bias, such as Mistral and Llama3, perform poorly at this task, as they tend to assume that the PLS is factual regardless of its actual faithfulness. Meanwhile, models with a "no"-bias, such as `Qwen_think` and `Qwen_nothink`, may be overly

sensitive to swaps, tending to classify PLS as non-factual regardless of their true status, and thus also frequently misidentifying factual PLS.

4.5 Yes, No Bias

4.5.1 Different Prompt Settings

To investigate the "yes"- and "no"-bias of the models, we designed two new sets of prompts. In the original prompt (introduced in Chapter 4.4), the model is instructed to answer "yes" when the PLS is factual and "no" when the PLS is not factual.

In the first new prompt, we reverse this instruction: the model should answer "no" when the PLS is factual and "yes" when the PLS is non-factual. The hypothesis is that if poor performance is caused by a bias towards answering "yes" or "no", then reversing the instruction should lead to a significant change in performance. This prompt is inspired by the PlainQAFact paper [25], where the original prompt was used with GPT-4o acting as the classifier. We refer to this as the "reversed prompt", and the one from Chapter 4.4 as the "original prompt".

Note that this prompt involves two roles: the *system* role and the *user* role. Using roles requires tokenizer support, and some models do not support roles, so in those cases we concatenate all instructions and inputs into a single string. Moreover, the functionality of roles can differ across models, and the available documentation is often vague. During our experiments, we observed that some models ignored instructions in the system role, so in those cases we placed the system instructions into the *developer* role instead.

We can see this prompt below:

```
messages = [
    {"role": "system",
     "content": "Annotate whether a sentence or summary
         includes information not present in the original
         abstract.\n"
                "The sentence or summary contains external
                    information that is not explicitly
                    mentioned, paraphrased, or implied in the
                     original abstract will be labeled as '
                    Yes'.\n"
                "The sentence or summary contains
                    information that is explicitly stated or
                    closely paraphrased from the original
                    abstract will be labeled as 'No'."},

    {"role": "user", "content":
                f"Sentence or summary: {Tag}\n"
```

```
                f"Original abstract: {Org}"}
]
```

In the second new prompt, we remove the use of "yes" and "no" entirely, so that any potential bias towards "yes" or "no" will not affect model performance. This setup can also provide further evidence for whether certain models exhibit such biases.

We keep the prompt style largely the same as in the previous case, but instruct the model to answer with "0" and "1" instead of "yes" and "no". In this way, we eliminate the influence of lexical bias towards "yes" or "no". We refer to this as the "01 prompt".

The prompt is as below:

```
messages = [
    {"role": "system",
     "content": "Annotate whether a sentence or summary
         includes information not present in the original
         abstract.\n"
                "The sentence or summary contains external
                    information that is not explicitly
                    mentioned, paraphrased, or implied in the
                     original abstract will be labeled as
                    '1'.\n"
                "The sentence or summary contains
                    information that is explicitly stated or
                    closely paraphrased from the original
                    abstract will be labeled as '0'."},

    {"role": "user", "content":
                f"Sentence or summary: {Tag}\n"
                f"Original abstract: {Org}"}
]
```

4.5.2 Issue with OSS_120

One thing to note is that we tested `OSS_120` only with *medium* reasoning effort, and only under the reversed prompt. This is because `OSS_120` is too large to run on Blythe without running out of memory, even with three L40 GPUs (total VRAM = 3×48 GB). This limitation stems from the quantisation used by the OSS models: MXFP4, a new 4-bit floating-point format designed to run on 80 GB H100 GPUs. In principle, this should fit on Blythe; however, MXFP4 is relatively new and is only fully supported on Hopper and Blackwell architectures, while the GPUs across Avon, Blythe, and Kudu are at most Ada Lovelace (one generation older than Blackwell). There is an open pull request on the GPT-OSS repository for older-architecture support, but it is not yet implemented. Consequently, `oss_120` falls back to 16-bit precision, which three L40s

cannot handle.

We attempted to dequantise the MXFP4 model and quantise it to a lower-precision format (e.g., FP4/FP8), but this required substantial time and VRAM, and could only be attempted on Blythe. Given the time constraints and Blythe's dual-storage setup (which complicated our workflow), we ultimately abandoned this route. We also evaluated quantised variants on Hugging Face: Unsloth provides a GGUF build of `OSS_120`, but we still encountered OOM errors, which appeared to be a bug. A separate 4-bit `OSS_120` (bnb quantised) from Unsloth could be made to run on Blythe, but its outputs were largely gibberish, so we discontinued that line as well. Unsloth appears to be a library that focuses more on fine-tuning and training rather than inference. The Unsloth-specific API and documentation are primarily geared towards fine-tuning, while inference relies on `llama.cpp`, which we are not familiar with. We attempted to use the model through Hugging Face's `transformers` API. Although the code runs successfully, the Unsloth-specific models are probably not properly configured for `transformers`, which explains the numerous obstacles we encountered with Unsloth models.

Our only success was running locally on a laptop with 128 GB of unified memory. A relevant discussion on Reddit suggests that Apple's Metal stack can treat MXFP4 as FP32, which should be *more* memory-hungry than FP16, implying it should not fit under normal assumptions. It appears that Ollama may apply additional optimisations for MXFP4 on macOS. A downside of using Ollama is that we cannot set the *reasoning effort*; this control is available on Hugging Face but not in Ollama. By default, `OSS_120` on Ollama uses *medium* reasoning, with no further documentation. Due to runtime constraints, we tested `OSS_120` only with the reversed prompt.

We will refer `OSS_20` and `OSS_120` on Ollama with and without developer role as: `OSS_120_Dev`, `OSS_20_NoDev`, `OSS_120_NoDev` and `OSS_20_Dev`.

Table 4.11: Pairwise agreement (%) and Cohen's kappa

Model 1	**Model 2**	**Agreement (%)**	**Kappa**
OSS_20_NoDev	OSS_20_Dev	98.25	0.965
OSS_20_Dev	OSS_120_NoDev	98.25	0.965
OSS_20_NoDev	OSS_120_NoDev	98.00	0.960
OSS_120_NoDev	OSS_120_Dev	98.00	0.960
OSS_20_NoDev	OSS_120_Dev	97.00	0.940
OSS_20_Dev	OSS_120_Dev	96.75	0.935

Table 4.12: OSS Models Performance on Ollama

Model	Correct	Wrong	Acc	Invalid
OSS_120_Dev	217	183	54.25	0
OSS_20_NoDev	214	186	53.50	0
OSS_20_med	214	186	53.50	0
OSS_120_NoDev	213	187	53.25	0
OSS_20_Dev	212	188	53.00	0

Table 4.13: Pair outcome counts per Ollama OSS model, BC for both_correct, BW for both_wrong, ONC for only_no_correct and OYC for only_yes_correct.

Model	BC	BW	ONC	OYC
OSS_120_Dev	17	0	183	0
OSS_20_NoDev	14	0	186	0
OSS_20_med	14	0	186	0
OSS_120_NoDev	13	0	187	0
OSS_20_Dev	12	0	188	0

Table 4.14 also shows that role settings (with and without developer role) have little impact for `OSS_20` and `OSS_120`; accuracy and the four category counts (Table 4.13) are similar. Accordingly, we standardise on the developer-role setting for `OSS_20` on Hugging Face for consistency with other runs. Moreover, the similar accuracies and outcome distributions for `OSS_20b` via Ollama and `OSS_20_med` on Hugging Face suggest that Ollama's `OSS_20` likely uses medium reasoning effort.

From Table 4.12 and Table 4.13, Both `OSS_120_NoDev` and `OSS_120_Dev` performs similarly to `OSS_20_NoDev` and `OSS_20_Dev`, and `OSS_20` at medium reasoning, with comparable accuracy (54.25% for `OSS_120_Dev` vs. 53.50% for `OSS_20_med`) and similar counts for all four outcomes. In Table 4.11, pairwise percentage agreement and Cohen's κ between `OSS_120` and `OSS_20` across role settings are both above 0.9, indicating *almost perfect* agreement. This supports our assumption that `OSS_120` with *low* and *high* reasoning would behave similarly to `OSS` with *low* and *high* reasoning. For these models perform similarly, we will only show `OSS_20_med` and `OSS_120` in the full table, Table 4.14 and Table 4.15.

4.5.3 Full Result

In Table 4.14, we compare the performance of reasoning, non reasoning models in Original, Reversed and 01 settings. "01" as prefix in category represents results from the new prompt that uses "0"

and “1” instead of “yes” and “no”, which is the 01 prompt. For results in the categories “Reversed” as prefix, the prompt used is the reversed prompt, where the instruction for “yes” and “no” was reversed.

Here we have also added an “Invalid” column, which represents the number of examples where we could not find a valid “yes”, “no”, “0”, or “1” in the model output. There are two possible cases for this: (1) the model did not follow the instructions and failed to output the required label, or (2) the model produced an incomplete output because the maximum token limit was reached before the final answer was generated. The latter case is more common in reasoning models, since their reasoning process consumes more tokens. This issue is not represented in the table, since we reran the models multiple times with progressively larger maximum token limits until most examples produced complete answers.

During the experiment, the GPT-OSS models were released, and since they are reasoning models, we applied both new prompts to them. From the original PlainQAFact paper, we know that `Llama3.1_I` performed best on PlainFact, so we also included two additional models at this stage: `Llama3.1_I` and `llama3-instruct`. Note that in the original paper, the output of `Llama3.1_I` was a factuality score, without a clear threshold to determine when a summary should be labelled “factual” or “non-factual”. To make comparisons easier, we converted its output into a binary classifier output, consistent with the other models.

During this stage of the experiment, we also came across the MedVAL framework. Since the results reported in its paper were promising, and because its output could easily be mapped into binary labels (risk level 1–2 interpreted as factual, risk level 4–5 as non-factual), we included the best MedVAL model in our evaluation. Note that MedVAL uses its own prompt, which we did not modify, and its output is a risk level from 1 to 5 rather than “yes” or “no”, so there should be no lexical bias in its results.

We did not obtain results for these new models under the original prompt, as we did not have enough time to rerun all models with that setting.

In Table 4.14, the Top 5 models by accuracy are highlighted in different colours. Three of these belong to the Original prompt setting and two to the 01 prompt setting. The best-performing model is `Qwen_nothink` from original non-reasoning, with an accuracy of 91.00%. The second-best is `Qwen_think` from original reasoning, with an accuracy of 84.75%. The third-best is `Qwen_nothink` from 01 non-reasoning, with an accuracy of 82.16%. The fourth-best is

Marco-O1 from the original reasoning, with an accuracy of 78.75%. Finally, the fifth-best is Llama3.1_I from 01 non-reasoning, with an accuracy of 72.75%.

Here in table 4.15, We show the four outcomes counts for each model. For each category, the non reasoning models are on top and the reasoning models are at the bottom.

In Table 4.14, Mistral and llama3—the two models exhibiting the strongest "yes" bias—achieve only about 50% accuracy under the reversed prompt. Under the 01 prompt, Mistral reaches 54.39% and llama3 reaches 53.75%, slightly better than their reversed/original results, but still poor.

From Table 4.15, llama3 shows a very large only_no_correct count of 191 under the reversed prompt. Recall that in this setting only_no_correct means the model answered "yes" for both the factual and non-factual PLS of the same abstract. This is strong evidence of a persistent "yes" bias: the model is not detecting swaps that flip factuality, and instead answers "yes" nearly always. Mistral behaves similarly with only_no_correct = 159, reinforcing the same conclusion.

With the original prompt, most models (except Marco_O1 and both Qwen_think and Qwen_nothink) show varying degrees of "yes" bias. Under the reversed prompt, almost all models exhibit a large only_no_correct count. For Qwen models, which showed a mild "no" bias under the original prompt, the reversed prompt still yields large only_no_correct counts: 128 (Qwen_think) and 118 (non-Qwen_nothink), with small only_yes_correct (14 and 15). Both reasoning and non-reasoning variants drop to both_correct = 10 (from 48 and 164 under the original prompt). Accuracy collapses from 84.75% (reasoning) and 91.00% (non-reasoning) to 41.01% and 39.23% under the reversed prompt, i.e., worse than random guessing. Thus, the reversed prompt strongly affects Qwen_nothink and Qwen_think; importantly, reasoning does not mitigate this effect.

This "yes" bias under the reversed prompt is not unique to Qwen models; Table 4.15 shows that most models accumulate large amount of only_no_correct counts, with accuracies hovering near 50%.

The main exceptions are Marco_O1, oss_20b_low, and Llama3_I, with reversed-prompt accuracies of 62.50%, 61.50%, and 68.72%, respectively. The two reasoning models, Marco_O1 and oss_20b_low, still show a tendency toward "yes" (only_no_correct = 126 and 152), but also achieve relatively high both_correct (56 and 47), far higher than other models (next best: oss_120b_med with 17). Llama3_I, a non-reasoning model, is the most balanced under the reversed prompt (with BC = 70, BW = 2, OYC = 61 and ONC

Category	Model	Correct	Wrong	Acc	Invalid
MedVAL					
	MedVAL_nothink	277	120	69.77	3
	MedVAL_think	269	129	67.59	2
Reversed Reasoning					
	Marco_O1	250	150	62.50	0
	OSS_20_low	246	154	61.50	0
	OSS_120	217	183	54.25	0
	OSS_20_med	214	186	53.50	0
	OSS_20_high	202	187	51.93	11
	Qwen_think	162	233	41.01	5
Reversed Non-Reasoning					
	Llama3_I	268	122	68.72	10
	Llama3.1_I_I	214	186	53.50	0
	Llama3	201	192	51.15	7
	Mistral	201	199	50.25	0
	Qwen_nothink	153	237	39.23	10
01 Reasoning					
	OSS_20_Low	272	128	68.00	0
	Qwen_think	225	164	57.84	11
	OSS_20_High	171	133	56.25	96
	Marco_O1	215	184	53.88	1
	OSS_20_Med	202	177	53.30	21
01 Non-Reasoning					
	Qwen_nothink	327	71	82.16	2
	Llama3.1_I	291	109	72.75	0
	Mistral	124	104	54.39	172
	Llama3	215	185	53.75	0
	Llama3_I	186	166	52.84	48
Original Reasoning					
	Qwen_think	339	61	84.75	0
	Marco_O1	315	85	78.75	0
Original Non-Reasoning					
	Qwen_nothink	364	36	91.00	0
	Mistral	203	197	50.75	0
	LLama3	200	200	50.00	0
QA					
	DS_7	224	176	56.00	0
	Lama3.1_I	183	141	56.48	76

Table 4.14: Performance comparison of reasoning, non-reasoning, old, and 01 models, with reasoning/non-reasoning separated in both original and 01 categories. Top 5 models by accuracy are highlighted in different colours.

Category	Model	BC	BW	OYC	ONC
MedVAL					
	MedVAL_nothink	77	0	123	0
	MedVAL_think	65	0	135	0
Reversed Reasoning					
	Marco_O1	56	6	126	12
	OSS_20_low	47	1	152	0
	OSS_120	17	0	183	0
	OSS_20_med	14	0	186	0
	Qwen_think	10	48	128	14
	OSS_20_high	4	2	194	0
Reversed Non-Reasoning					
	Llama3_I	70	2	67	61
	Mistral	18	17	159	6
	Llama3.1_I_I	17	0	183	0
	Qwen_nothink	10	57	118	15
	llama3	4	3	191	2
01 Reasoning					
	OSS_20_low	72	0	125	3
	Marco_O1	35	20	134	11
	Qwen_think	32	7	161	0
	OSS_20_med	10	8	182	0
	OSS_20_high	1	30	169	0
01 Non-Reasoning					
	Qwen_nothink	128	1	71	0
	Llama3.1_I_I	91	0	36	73
	Llama3	18	3	166	13
	Mistral	15	91	64	30
	Llama3_I	9	23	165	3
Original Reasoning					
	Marco_O1	115	0	25	60
	Qwen_think	48	59	70	23
Original Non-Reasoning					
	Qwen_nothink	164	0	36	0
	Mistral	0	0	0	200
	Llama3	27	27	5	141
QA					
	DS_7	52	31	96	21
	Llama3.1_I_I	29	46	112	13

Table 4.15: Pair outcome counts per model (sorted within categories by both_correct), BC for both_correct, BW for both_wrong, ONC for only_no_correct and OYC for only_yes_correct.

= 67.), yielding the highest reversed-prompt accuracy (68.72%).

For `Qwen_nothink` and `Qwen_think`, the gap between reasoning and non-reasoning under the reversed prompt is small (41.01% vs. 39.23%), and both are poor. Table 4.15 shows `both_correct` = 10 (both), with the two largest `both_wrong` counts (48 and 57). This explains the low accuracies and confirms a "yes" bias (large `only_no_correct`).

Comparing reasoning vs. non-reasoning under the reversed prompt, the best model is `Llama3_I` (non-reasoning). Although `Marco_O1` and `oss_20b_low` are close, both results exhibit clear bias. Notably, `OSS_20_low` outperforms higher-reasoning-effort OSS variants in accuracy and `both_correct`, suggesting that *lower* reasoning effort yields *higher* accuracy and more balanced outcomes across the four categories in Table 4.15. Overall, reasoning does not appear to help factuality evaluation here, nor does it reduce "yes"/"no" bias.

For the 01 prompt, we did not include `OSS_120` due to time limits; given its similarity to `OSS_20_med` elsewhere, we assume comparable 01-prompt results. From Table 4.14, the best 01-prompt model is `Qwen_nothink` at 82.16%, followed by `Llama3.1_I` (non-reasoning) at 72.75%, and `OSS_20_low` (reasoning) at 68.00%.

Interestingly, `Qwen_nothink`, which performed worst under the reversed prompt, becomes the best under 01. From Table 4.15, `both_correct` rises to 128 (highest among 01 models), `only_no_correct` drops to 71, and `both_wrong` falls to 1 (from 57 under reversed). Similarly, `Qwen_think` improves from below 50% to 57.84%; `both_correct` increases to 32 (from 10), `only_no_correct` decreases to 161 (from 128; still large), and `both_wrong` drops to 7 (from 48). These results suggest that under the 01 setting, removing lexical "yes"/"no" responses reduces bias and confusion for `Qwen_nothink` and `Qwen_think`, dramatically lowering `both_wrong` and improving accuracy.

By contrast, `Marco_O1` performs worse under 01: accuracy drops from above 60% (reversed) to 53.88%. Although `Marco_O1` shows a mild "yes" bias, the key driver of the drop is the rise in `both_wrong` from 6 (reversed) to 20 (01). `only_no_correct` increases slightly (126 $\rightarrow$ 134), while `only_yes_correct` changes marginally (12 $\rightarrow$ 11). This suggests `Marco_O1` is less adept at catching factuality-changing swaps under the 01 regime.

Llama3_I also degrades markedly under 01: accuracy = 52.84% (lowest among non-reasoning models in 01). `both_wrong` jumps to 23 (from 2), `both_correct` collapses to 9 (from 70), and

only_yes_correct drops to 3 (from 61), driving only_no_correct up to 165 (from 67). Thus, 01 appears to introduce a bias towards "1" (akin to "yes") for Llama3_I, or otherwise destabilises its decision pattern. Multiple test rounds would help diagnose this, but were infeasible. Note also its high invalid count: 48/400 ($\approx$ 12%), indicating weaker instruction-following compared to other models.

Llama3.1_I, on the other hand, improves substantially: from 53.50% (reversed) to 72.75% (01), second-best overall in 01. Under the reversed prompt it showed a strong "yes" bias (only_no_correct = 183). With 01, it becomes much more balanced: both_correct = 91, only_no_correct = 36, only_yes_correct = 73, and both_wrong = 0. Thus, 01 reduces "yes"/"no" bias and boosts performance for Llama3.1_I.

Mistral remains consistently weak across prompts: 50.75% (original), 50.25% (reversed), and 54.39% (01), i.e., near random guess. Under the reversed prompt it still shows a strong "yes" bias, with only_no_correct = 159. Under 01, it attains the highest both_wrong among all settings (91) and large only_no_correct/only_yes_correct (64/30), indicating it fails to capture factuality-changing swaps and remains biased. Its 01 invalid count is also the highest (172/400 $\approx$ 43%), largely due to instruction-following failures and gibberish outputs, suggesting poor alignment.

MedVAL is relatively consistent: 69.77% (MedVAL_nothink) vs. 67.59% (MedVAL_think). From Table 4.15, both_correct is 77 (no reasoning) vs. 65 (reasoning), and only_no_correct is 123 vs. 135; both_wrong and only_yes_correct are 0 in both cases. Thus, both MedVAL_nothink and MedVAL_think slightly improves without reasoning (higher both_correct, lower only_no_correct). There are small invalid counts (3 vs. 2). Despite being fine-tuned for the task, both MedVAL models still misses many non-factual PLS (high only_no_correct and zero only_yes_correct), suggesting insufficient sensitivity to swaps. Nevertheless, with a tailored prompt and fine-tuning, a small model (base Qwen-4b) can be competitive. Overall, reasoning offers no benefit here and may slightly reduce both MedVAL's performance.

Llama3 For Llama3, we observe that with the reversed prompt, it achieves a similar accuracy to the original prompt. It also has comparable accuracy with the 01 prompt, with all accuracies being

around 50%, which is close to random guessing. In both the original and reversed prompts, we can see a strong "yes" bias, suggesting that the model simply prefers to answer with "yes".

However, we also observe this trend in the 01 prompt, where the model classifies most examples as non-factual. This contradicts its behaviour with the original prompt, where it classified most examples as factual. This suggests that the model is quite unstable in its results and lacks the ability to reliably distinguish the factuality of PLS.

In summary, many models exhibit a bias towards simply answering "yes" or "no"; this bias materially harms performance. We can also see a similar bias from out previous QA system. The 01 prompt (removing "yes"/"no") alleviates this for some models (notably `Qwen_think`, `Qwen_nothink` and `Llama3.1_I`), substantially improving accuracy. Reasoning, in contrast, does not improve factuality evaluation in this setting and can even degrade performance. We also observe that a fine-tuned model on this specific task still outperforms most other models, including both reasoning and non-reasoning variants. Moreover, reasoning ability does not appear to be beneficial, even when the model has been fine-tuned.

4.6 Distribution of different swaps and performance

In this analysis, we examine how the number of swaps per pair of factual and non factual PLS affects model performance.

Note here we are not considering oss old 120b nodev, oss old 20b dev and oss old 20b nodev when calculating average over all reasoning models.

In Figure 4.1a, we show the average number of different swaps across outcomes for all models under all prompt settings. We observe that, in total, antonym and negation swaps occur in roughly equal amounts across outcomes, while keyword swaps are slightly more frequent in the *only-no-correct* and *only-yes-correct* cases.

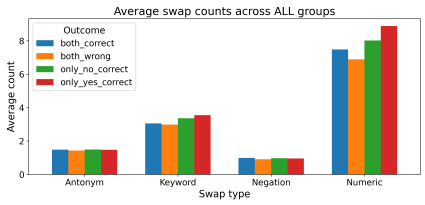
For numeric swaps, it appears that performance decreases when there are fewer numeric swaps, as both wrong shows less numeric swaps. This suggests that it is harder for models to distinguish factual from non-factual pairs when there are fewer numeric differences, while larger numeric differences are easier for the models to detect.

In Figures 4.1b, 4.1c, and 4.1d, we show the distribution of swaps per outcome across the three prompt setups. In all three settings, antonym and negation swaps follow similar trends across outcomes.

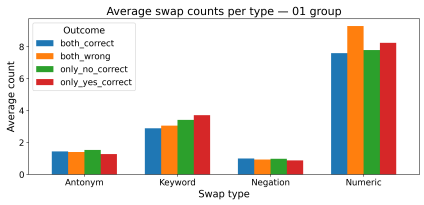
For keyword swaps, the trend is similar for the 01 and reversed prompts, but under the original prompt setup, fewer keyword swaps

are associated with the *both-wrong* outcome. This indicates that in the original setting, a lower number of keyword swaps makes evaluation more difficult.

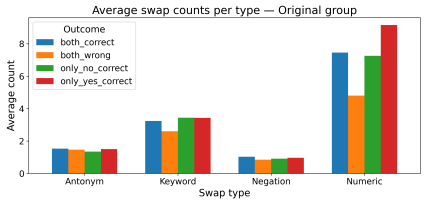
The most significant differences occur in numeric swaps. In the original and reversed setups, fewer numeric swaps are associated with worse performance (*both-wrong* outcomes), whereas in the 01 setting, a higher number of numeric swaps makes evaluation harder. This shows that changing the target outputs from "yes/no" to "0/1" alters the models' sensitivity to numeric information in the text. Moreover, in both the original and reversed setups, we observe that models tend to classify both PLS as factual (answering "yes") when there are more numeric swaps.



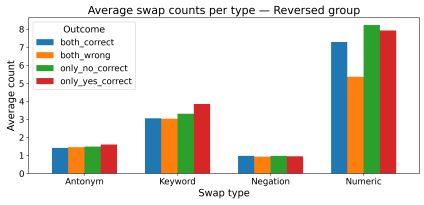
(a) Average swaps for All results.



(b) Average swaps in 01 setup.



(c) Average swaps in Original setup.



(d) Average swaps in Reversed setup.

Figure 4.1: Comparison of average swap counts across different setups.

In Figure 4.2, we compare reasoning and non-reasoning models under the same setup.

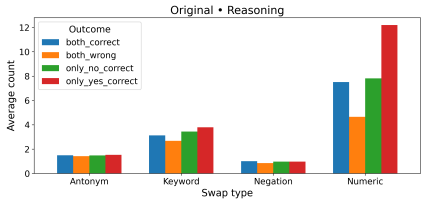
For the original prompt (Figures 4.2a and 4.2b), we observe that with less keyword swaps for *both-wrong* cases increases, making the task harder for the models. Antonym and negation counts remain similar between reasoning and non-reasoning models. The major difference lies in numeric swaps: reasoning models are more likely to classify texts with more numeric swaps as factual, compared to non-reasoning models. Thus, reasoning ability appears to worsen performance when there are many numeric differences.

This finding is further supported by the reversed prompt results (Figures 4.2c and 4.2d), where reasoning models, compared to non-

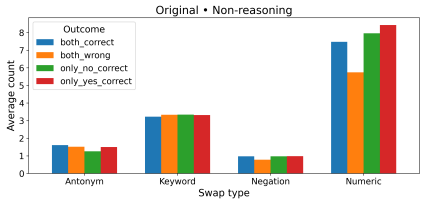
reasoning ones, tend to show more numeric swaps in their *only-no-correct* outcomes.

In the 01 setting ((Figures 4.2e and 4.2f)), however, more numeric swaps cause reasoning models to produce *both-wrong* outcomes compared to non-reasoning models. This contradicts the pattern seen in the original and reversed settings, where fewer numeric swaps are associated with *both-wrong* outcomes for reasoning models. For non-reasoning models, by contrast, fewer numeric swaps consistently lead to more *both-wrong* outcomes.

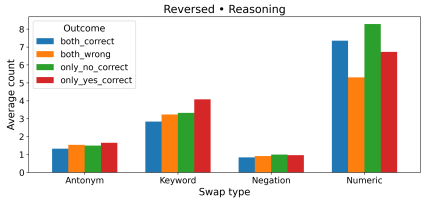
Overall, this suggests that the answer format requested from reasoning models (e.g., "yes/no" vs. "0/1") can influence their performance when numeric swaps are present. This effect, however, is not as strong in non-reasoning models.



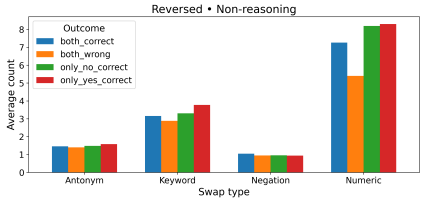
(a) Avg swaps in Original Reasoning



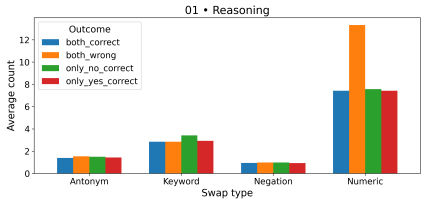
(b) Avg swaps in Original Non Reasoning



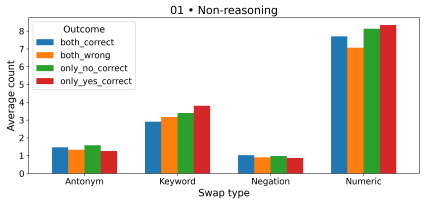
(c) Avg swaps in Reversed Reasoning



(d) Avg swaps in Reversed Non Reasoning



(e) Avg swaps in 01 Reasoning



(f) Avg swaps in 01 Non Reasoning

Figure 4.2

In Figure 4.3, we show the distribution of swaps across different outcomes for Qwen_think and Qwen_nothink.

One noticeable difference appears in the 01 setup (Figures 4.3e and 4.3f). Here, a higher number of numeric swaps confuses Qwen_think, as the number of numeric swaps in the *both-wrong* category is high.

By contrast, for Qwen_nothink there are no *both-wrong* cases with numeric swaps at all. Instead, many numeric swaps appear in the *both-correct* category, suggesting that Qwen_nothink performs well when there are more numeric swaps. However, when numeric swaps are high, Qwen_nothink also tends to classify PLS as non-factual, showing that it is quite sensitive to numeric changes.

Qwen_think shows fewer antonym, keyword, and negation swaps in *both-correct* examples compared to Qwen_nothink, suggesting that the reasoning model is weaker than the non-reasoning model in capturing these types of swaps.

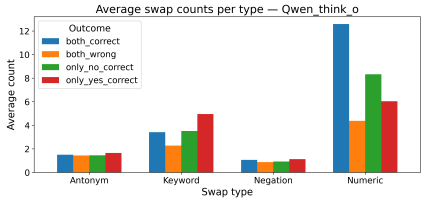
The largest performance gap between Qwen_think and Qwen_nothink occurs when comparing the original and reversed setups. For Qwen_think (Figures 4.3c and 4.3a), the difference in *both-correct* cases is substantial (around 12–13 in the original setting vs. 8–9 in the reversed setting). Qwen_think performs better when there are more numeric swaps. This indicates that numeric swaps are the main driver of performance differences between the original and reversed setups for Qwen_think. In the reversed setup, there are fewer numeric swaps in *both-correct* cases, which may indicate that the model is less sensitive to numeric changes under this configuration. For the other swap types, differences are marginal.

When comparing Qwen_nothink across settings, numeric swaps again show the largest differences. In the reversed setup, there are far more numeric swaps in *both-correct* compared to the original setting. Additionally, the distribution of outcomes differs significantly: in the original prompt, only two categories appear (*both-correct* and *only-no-correct*), while in the reversed prompt all four outcome categories are represented. This demonstrates that the reversed prompt leads to more inconsistent behaviour. The other three swap types remain relatively stable across prompt settings.

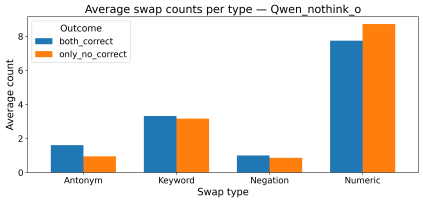
In summary, for both Qwen_think and Qwen_nothink, the biggest changes occur in numeric swaps, and changing the output type alters how the distribution of swaps maps across different outcomes.

In Figure 4.4, we show the distribution of different swaps for OSS_20_low, OSS_20_med, and OSS_20_high in both the reversed and original settings. We do not include OSS_120 here, as it shows very similar performance to OSS_20_med. The exact comparison can be found in Appendix C.

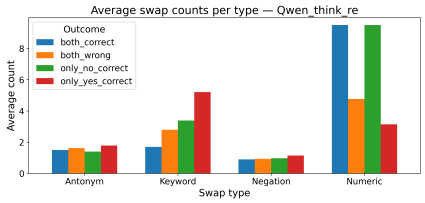
For OSS_20 models under the reversed prompt (Figures 4.4a,



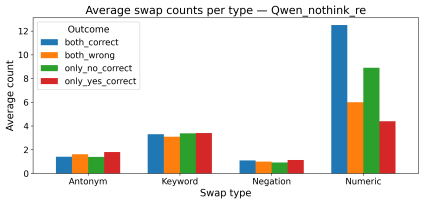
(a) Qwen_think with Original



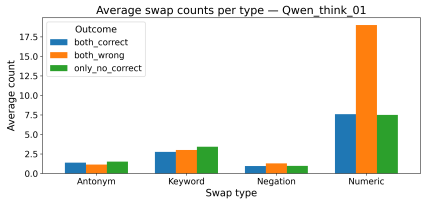
(b) Qwen_nothink with Original



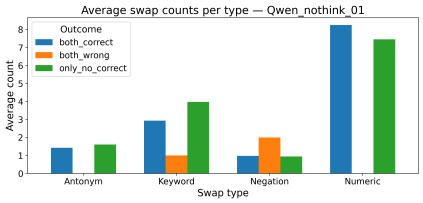
(c) Qwen_think with Reversed



(d) Qwen_nothink with Reversed



(e) Qwen_think with 01



(f) Qwen_nothink with 01

Figure 4.3

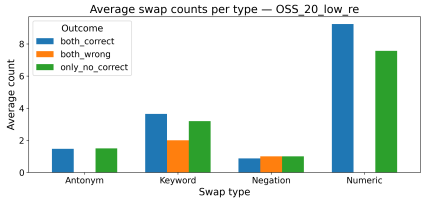
4.4c, and 4.4e), numeric swaps remain the clearest indicator of divergence. With low reasoning effort, more numeric swaps appear in the *both-correct* category, but as reasoning effort increases, this number decreases. At the same time, the overall number of numeric swaps increases with reasoning effort. If the model were able to catch even a single numeric swap, it could determine factuality, but it fails to do so. As a result, performance decreases as reasoning effort increases.

A similar trend is observed for numeric swaps in the *both-wrong* category under the 01 setting (Figures 4.4b, 4.4d, and 4.4f). We notice that from OSS_20_low to OSS_20_med, the number of swaps across all outcomes increases, suggesting that the model loses performance as reasoning effort grows. This is reflected in the fact that both the *both-wrong* and *only-yes-correct* outcomes contain more swaps. In theory, more errors should make it easier to detect at least one, but the models fail to take advantage of this. With OSS_20_high, there are no *both-correct* cases with numeric swaps, and a massive increase is observed in the *both-wrong* category. This indicates that its performance is very poor.

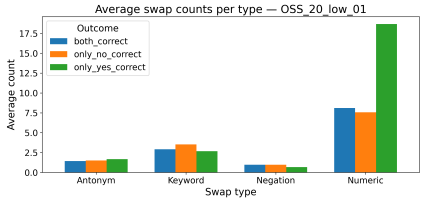
Finally, we can see that the distribution differs substantially between the reversed and 01 settings, again showing that the answer format affects model performance.

In general, we observe that model performance is influenced by the answer type specified in the prompt, though the severity of this effect differs between reasoning and non-reasoning models. Across models, those with less or lower reasoning effort tend to perform better.

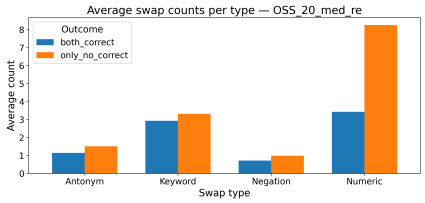
As shown in Table 3.18, numeric swaps occur most frequently among all swap types. They also appear to be the most important factor affecting model performance. If a model can reliably detect numeric swaps, it is likely to perform well on PlainFact.



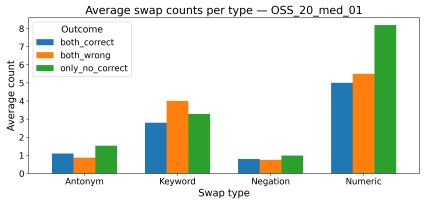
(a) OSS_20_low with Reversed



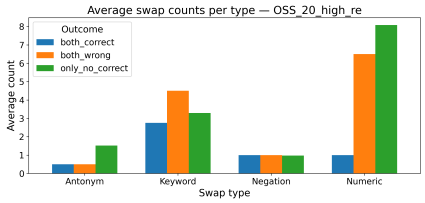
(b) OSS_20_low with 01



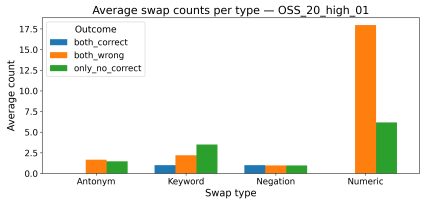
(c) OSS_20_med with Reversed



(d) OSS_20_med with 01



(e) OSS_20_high with Reversed



(f) OSS_20_high with 01

Figure 4.4

5 Project Management

5.1 Limitation

One of the major limitations of this project is our lack of knowledge about recent LLMs. We only studied NLP systematically through CS918, and more advanced model architectures remain largely unknown to us and are too complex for us to analyse in depth. As a result, we may have overlooked important features that explain why some models with similar characteristics perform in comparable ways.

Another limitation is that we are not familiar with medical terminology. Thus, we relied on entities extracted by pre-built NER models to identify keyword swaps. A more nuanced classification of terms could allow for a finer-grained analysis of the dataset and provide deeper insights.

Since LLMs are statistical models, the same model may produce different answers to the same question. Moreover, the reasoning traces of reasoning models can sometimes guide the model towards incorrect answers. It is therefore necessary to run each model multiple times on the same dataset to measure fluctuations in performance and then report the average. This approach would produce more robust results. However, due to limited time, we were only able to run each model once per dataset.

5.2 Time Management

One of the biggest obstacles we faced during the project, and which cost us the most time, was downloading models from Hugging Face using the "snapshot_download" method. This method, described in Hugging Face's documentation, allows downloading to a manually specified path. This feature was critical for running models on Blythe and DCS, since both use dual storage systems with a dedicated path for larger storage capacity, and we needed to directly specify the storage location for the models.

To connect to remote servers, I used VS Code with SSH. However, VS Code on my laptop experienced many connection issues: the connection quality was unstable, sometimes dropping unexpectedly, and at other times failing to connect entirely. This consumed a significant amount of time, and although I attempted various solutions, they were ultimately unsuccessful.

It was not until I learned how to contact support for Blythe and Avon that I was able to partially resolve the problem. Before that,

I was limited to using Kudu and my own laptop. This issue was never completely solved, as there were still occasions when I could not connect to Avon or Blythe and had to attempt several logins before succeeding. This was highly time-consuming, and the wasted time accumulated significantly over the course of the project.

The method worked until early August, when we suddenly observed that some jobs requiring model downloads would continue running until the time limit (two days). At first, we assumed the models were simply too large, but later realised the downloads were stalling and never finishing. We suspected either model size issues or network problems between the HPC servers and Hugging Face. After several attempts, we noticed that downloads consistently stopped at a certain point. We tried different adjustments to the "snapshot_download" parameters, but nothing worked. Eventually, we found a Reddit post reporting the same issue, with the recommended solution being to use Hugging Face's command line interface (CLI) to download models via terminal. This resolved the problem. In total, it took us about one and a half weeks to diagnose and fix this issue, most of which was spent digging through documentation, experimenting with configurations, and waiting for failed jobs to complete. During this period, no progress could be made, as no models were running.

We also spent significant time learning how to properly run models on multiple GPUs. Initially, we set the "device_map" parameter to "auto", which in theory should distribute the model across multiple GPUs. However, this frequently caused out-of-memory (OOM) errors, even on mid-sized models such as Qwen_think. We attempted several methods, none of which worked, until we eventually resolved the issue by manually configuring memory allocation across GPUs. This process also took about a week, as we first attempted to run the models on our own laptop, which further delayed progress.

A substantial amount of time was also spent waiting for model outputs from the servers. Some models, such as MedVAL_think, completed the entire PlainFact dataset in under two hours. By contrast, models such as OSS_20_high often required two full days to complete. In general, reasoning models took significantly longer than non-reasoning models, with higher reasoning effort correlating with longer runtime. There was little we could do during this time, as results were required for subsequent analysis.

Another challenge was that the reasoning process sometimes generated too many tokens. This led to incomplete outputs for certain examples, and in some cases required re-running entire datasets. Although we only needed to re-run the incomplete examples, the

increased token generation per example still extended runtime significantly.

In some cases, models generated extremely long reasoning traces. Typically, setting "max_new_token" to 512 was sufficient for non-reasoning models, and 2048 for reasoning models. However, for OSS_20_high, there were instances where we had to set "max_new_token" to 65536 (32×2048) in order to obtain a complete output. This took a very long time to finish.

While Blythe and Kudu were generally available, at busy times we still had to wait in queues. Avon was the busiest server, where almost every submitted job immediately entered the queue. Moreover, it initially took us time to learn how to correctly set the model storage paths for Blythe and DCS. As a result, we were forced to run larger LLMs on Avon, which both took longer to execute and required queueing, significantly reducing time efficiency.

Finally, since we ran jobs across three servers, each required slightly different configurations. On several occasions, we mistakenly applied the configuration for one server to another, causing jobs to fail to initiate. In some cases, these failures went unnoticed for extended periods, leading to further wasted time.

5.3 Timetable

Here we present the timetable for this project. The original timetable was quite rough, as the project was highly task-dependent. It was difficult to predict the outcomes of the experiments in advance, and different actions had to be taken depending on the results obtained.

Most of the obstacles we faced occurred between 17 July and late August, which contributed to the delayed start of the dissertation report.

5.4 Appraisal and reflection

Trying to get the models to run properly took longer than expected, and when problems arose, troubleshooting requested long sessions of debugging and would have benefit from more systematic approaches in structuring the overall development. However, the project did provide us with valuable experience in working with such systems.

Table 5.1: Original Timetable for Project Management

Time	**Work**
February	Read papers and discuss potential routes with a supervisor.
March – Mid-April	Implement potential methods discussed with the supervisor.
Mid-April – Mid-May	Prepare presentation (exam period requires extra time).
Mid-May – Mid-June	Implement and evaluate potential methods; seek improvements.
Mid-June – 10th July	Draft and polish Interim Report.
10th July – Mid August	Further implement and evaluate methods; seek improvements.
Mid August – 2nd September	Draft and polish Dissertation Report.

Table 5.2: Work schedule.

Time	**Work**
February	Read papers and discuss potential routes with supervisor.
March – Mid-April	Implemented the QA system and getting familiar with tools, such as remote servers and APIs.
Mid-April – 9th-May	Prepare presentation (exam period, we mainly focus on revision).
10th-May – Early June	Revision for examn
Early June – Late-June	Implemented the first part of evaluation of factuality
Late-June – 17th July	Draft and polish Interim Report.
17th July – Late August	Implemented the rest of evaluation of factuality, including more models. Seek ways for analysis of difference in performance of models.
Late August – 9nd September	Draft and polish Dissertation Report.

6 Further Work

We observe that when Open-R1 attempts to replicate DeepSeek-R1, it uses the reasoning traces of DeepSeek-R1 to fine-tune the base model. Thus, the characteristics of the model are reflected in its reasoning traces. It is therefore meaningful to examine the reasoning processes of reasoning models. By analysing these reasoning processes, we can inspect how different reasoning models approach problems and potentially gain insights into the performance differences between them.

We also note that different models show varying levels of agreement with each other. One possible approach is to design a multi-agent or QA system in which multiple models—particularly those with low or negative kappa agreement—compare and evaluate each other's results to decide the factuality of a PLS. A low or negative kappa score suggests complementary behaviour, so combining such models in an ensemble could allow the system to capture different types of errors.

In this project, we only considered the scenario in which non-factuality means that there are facts in the PLS that are different from, or missing in, the scientific abstract. However, another type of error can occur when facts are present in the scientific abstract but missing in the PLS. We did not consider this type of error, as it is more difficult to detect. Nonetheless, as shown in Figure 3.8, there is a significant difference in the average length between the PLS and the scientific abstracts. This suggests that entire claims may be omitted from the PLS simply because it is shorter. Developing methods to detect such errors, and analysing how the distribution of these errors correlates with model performance, could provide valuable insights.

References

[1] meta-llama/llama-3.1-8b-instruct · hugging face, 09 2024.

[2] A. Aali, V. Bikia, M. Varma, N. Chiou, S. Ostmeier, A. Singhvi, M. Paschali, A. Kumar, A. Johnston, K. Amador-Martinez, E. J. P. Guerrero, P. N. C. Rivera, S. Gatidis, C. Bluethgen, E. P. Reis, E. D. Z. van Rilland, P. L. Hosamani, K. R. Keet, M. Go, E. Ling, D. B. Larson, C. Langlotz, R. Daneshjou, J. Hom, S. Koyejo, E. Alsentzer, and A. S. Chaudhari. Medval: Toward expert-level medical text validation with language models, 2025.

[3] AI@Meta. Llama 3 model card. 2024.

[4] Q. Chen, Y. Hu, X. Peng, Q. Xie, Q. Jin, A. Gilson, M. B. Singer, X. Ai, P.-T. Lai, Z. Wang, V. K. Keloth, K. Raja, J. Huang, H. He, F. Lin, J. Du, R. Zhang, W. J. Zheng, R. A. Adelman, Z. Lu, and H. Xu. Benchmarking large language models for biomedical natural language processing applications and recommendations. *Nature Communications*, 16(1), Apr. 2025.

[5] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[6] A. Devaraj, W. Sheffield, B. C. Wallace, and J. J. Li. Evaluating factuality in text simplification, 2022.

[7] H. Face. Open r1: A fully open reproduction of deepseek-r1, January 2025.

[8] Y. Guo, W. Qiu, G. Leroy, S. Wang, and T. Cohen. Retrieval augmentation of large language models for lay language generation, 2024.

[9] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023.

[10] P. Kincaid, R. P. Fishburne, R. L. Rogers, and B. S. Chissom. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. 1975.

[11] Y. Li, X. Hu, X. Qu, L. Li, and Y. Cheng. Test-time preference optimization: On-the-fly alignment via iterative textual feedback, 2025.

[12] C. Library. Cochrane reviews — cochrane library, 2024.

[13] M. L. McHugh. Interrater reliability: the kappa statistic. *Biochemia Medica*, 22:276 – 282, 2012.

[14] N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto. s1: Simple test-time scaling, 2025.

[15] M. H. Murad, N. Asi, M. Alsawas, and F. Alahdab. New evidence pyramid. *Evidence-Based Medicine*, 21:125 – 127, 2016.

[16] M. Neumann, D. King, I. Beltagy, and W. Ammar. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 319–327, Florence, Italy, Aug. 2019. Association for Computational Linguistics.

[17] OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025.

[18] S. Ramprasad and B. C. Wallace. Do automatic factuality metrics measure factuality? a critical evaluation, 2024.

[19] S. Raschka. Ahead of ai — sebastian raschka, phd — substack, 2022.

[20] Q. Team. Qwen3 technical report, 2025.

[21] J. Trienes, S. Joseph, J. Schlötterer, C. Seifert, K. Lo, W. Xu, B. C. Wallace, and J. J. Li. Infolossqa: Characterizing and recovering information loss in text simplification, 2024.

[22] H. Van, D. Kauchak, and G. Leroy. AutoMeTS: The autocomplete for medical text simplification. In D. Scott, N. Bel, and C. Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1424–1434, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics.

[23] Y. Wang, Q. Liu, J. Xu, T. Liang, X. Chen, Z. He, L. Song, D. Yu, J. Li, Z. Zhang, R. Wang, Z. Tu, H. Mi, and D. Yu. Thoughts are all over the place: On the underthinking of o1-like llms, 2025.

[24] A. Yang, B. Zhang, B. Hui, B. Gao, B. Yu, C. Li, D. Liu, J. Tu, J. Zhou, J. Lin, K. Lu, M. Xue, R. Lin, T. Liu, X. Ren, and Z. Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

[25] Z. You and Y. Guo. Plainqafact: Automatic factuality evaluation metric for biomedical plain language summaries generation, 2025.

[26] Y. Zhao, H. Yin, B. Zeng, H. Wang, T. Shi, C. Lyu, L. Wang, W. Luo, and K. Zhang. Marco-o1: Towards open reasoning models for open-ended solutions, 2024.

A MedVAL Prompt Template

Here we show the prompt template for MedVAL. Note the "task_instruction" is to let the model know the task is text simplification.

Listing 1: Prompt Template for MedVAL

```
task_instruction = """
Create a simplified, patient-friendly version of the input.
1. Input Description: The original text containing medical
    terminology.
2. Output Description: The simplified, patient-friendly, and
     easy-to-understand version of the text.
"""

def build_prompt(o: str, t: str) -> str:
    return f"""
Your objective is to evaluate the output in comparison to
    the input composed by an expert.

Instructions:
1. Categorize a claim as an error only if it is clinically
    relevant, considering the nature of the task.
2. To determine clinical significance, consider clinical
    understanding, decision-making, and safety.
3. Some tasks (e.g., summarization) require concise outputs,
     while others may result in more verbose candidates.
    - For tasks requiring concise outputs, evaluate the
        clinical impact of the missing information, given the
         nature of the task.
    - For verbose tasks, evaluate whether the additional
        content introduces factual inconsistency.

Your input fields are:
1. "instruction" (str)
2. "input" (str)
3. "output" (str)

Your output fields are:
1. "reasoning" (str)
2. "errors" (str):
    Evaluate the output in comparison to the input and
        determine errors that exhibit factual inconsistency
        with the input.

    Instructions:
    - Output format: "Error 1: <brief explanation>"
Error 2: ...
    - Each error must be numbered and separated by a newline
         character; do not use newline characters for
        anything else.
    - Return "None" if no errors are found.
    - Refer to the exact text from the input or output in
```

```
        the error assessments.

    Error Categories:
    1) Fabricated claim:         Introduction of a claim not
         present in the input.
    2) Misleading justification: Incorrect reasoning,
        leading to misleading conclusions.
    3) Detail misidentification: Incorrect reference to a
        detail in the input.
    4) False comparison:         Mentioning a comparison not
         supported by the input.
    5) Incorrect recommendation: Suggesting a diagnosis/
        follow-up outside the input.
    6) Missing claim:            Failure to mention a claim
        present in the input.
    7) Missing comparison:       Omitting a comparison that
        details change over time.
    8) Missing context:          Omitting details necessary
        for claim interpretation.
    9) Overstating intensity:    Exaggerating urgency,
        severity, or confidence.
    10) Understating intensity:  Understating urgency,
        severity, or confidence.
    11) Other:                   Additional errors not
        covered.

3. "risk_level" (Literal[1, 2, 3, 4]):
    The risk level must be an integer from 1, 2, 3, or 4.
        Assign a risk level to the output from the following
        options:

    Level 1 (No Risk):       No clinically meaningful
        factual inconsistencies. Any deviations should not
        affect clinical understanding, decision-making, or
        safety.
    Level 2 (Low Risk):      Subtle or ambiguous
        inconsistencies, unlikely to influence clinical
        decisions or understanding.
    Level 3 (Moderate Risk): Inconsistencies that could
        plausibly affect interpretation, documentation, or
        decision-making.
    Level 4 (High Risk):     Errors that could result in
        unsafe or incorrect clinical decisions.

All interactions will be structured in the following way,
    with the appropriate values filled in.

[[ ## instruction ## ]]
{task_instruction}

[[ ## input ## ]]
{o}

[[ ## output ## ]]
```

```
{t}

[[ ## reasoning ## ]]
# TO_BE_FILLED_BY_MODEL

[[ ## errors ## ]]
# TO_BE_FILLED_BY_MODEL

[[ ## risk_level ## ]]
# TO_BE_FILLED_BY_MODEL

[[ ## completed ## ]]
"""
```

B Prompt for QA

```
source_prompt = (
    "Statement: {src\_prompt} List all the facts we
        explicitly know
    from the statement. Make each fact as atomic as possible
        ."
).format(src\_prompt=src\_prompt)

target_prompt = (
    "Statement: {tgt\_prompt} List all the facts we
        explicitly know
    from the statement. Make each fact as atomic as possible
        ."
).format(tgt\_prompt=tgt\_prompt)

compare_prompt = (
    "Source: {src} Target: {tgt} Compare the two lists of
        fact
    and list the fact that is in Target but not in Source."
).format(
    src=source_response_content,
    tgt=target_response_content
)

label_prompt = (
    "Statement: {state} Does the statement agree that there
    are facts presented in the Target list are not reflected
    in the Source list? "
    "Answer with 'yes' or 'no'."
).format(incorrect_facts)
```

The variables “src_prompt” and “tgt_prompt” represent the original text and the simplified text from the dataset, respectively. The variables “src” and “tgt” in “compare_prompt” are generated texts produced using “src_prompt” and “tgt_prompt”, respectively.

Finally, “state” is the generated text obtained by using “compare_prompt” as the input prompt.

C Distribution of different swaps and performance images

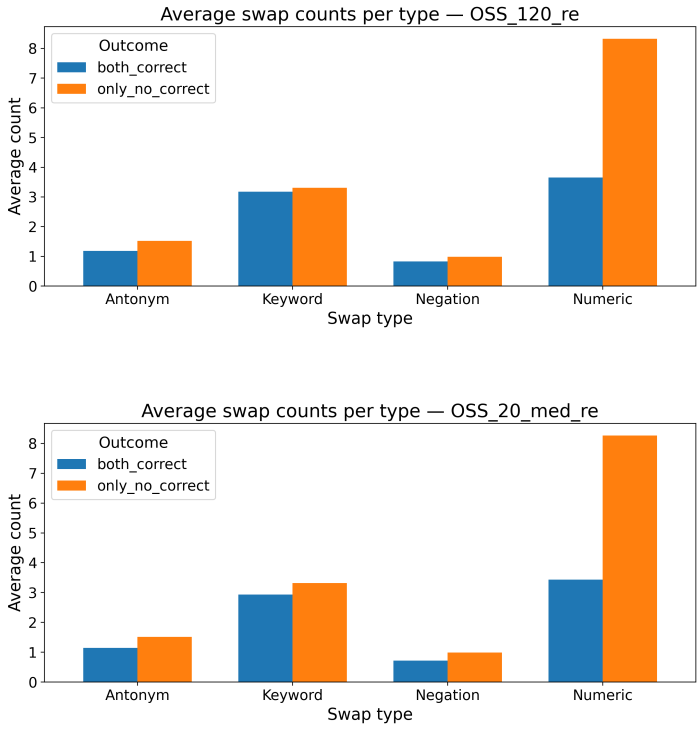


Figure C.1