# Efficient Fact-checking through Supporting Facts Extraction from Large Data Collections

Kevin Nanhekhan

# Efficient Fact-checking through Supporting Facts Extraction from Large Data Collections

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Kevin Nanhekhan
born in The Hague, The Netherlands

Web Information Systems Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

# Efficient Fact-checking through Supporting Facts Extraction from Large Data Collections

Author: Kevin Nanhekhan
Student id: 4959094
Email: K.R.Nanhekhan@student.tudelft.nl

## Abstract

Amidst the rampant spread of misinformation, fact-checking of diverse claims made on the internet has become a pertinent task to mitigate this problem. Manual fact-checking cannot scale up with this demand and is very cumbersome, therefore instead automated fact-checking can be used. However, existing work has primarily focused on the fact-verification part rather than evidence retrieval for large data collections, leading to scalability issues for practical applications. In this study, we address this gap by exploring various methods for indexing a succinct set of supporting facts extracted from large data collections and enhancing the retrieval phase of the fact-checking pipeline. Our evaluation, consisting of measuring the performance and efficiency, is performed on the state-of-the-art claim datasets HoVer and WiCE, where we utilised the English Wikipedia as a large evidence data collection. Overall our results underscore the effectiveness of integrating supporting facts and advanced retrieval techniques for fact-checking pipelines in practical applications. We achieve, through a combination of indexing supporting facts together with Dense retrieval and Index compression, a massive improvement over the original fact-checking pipeline. This is up to a **10.0x** speedup using a CPU-based approach and up to a **20.0x** speedup using a GPU-based approach, while only incurring a modest loss of less than 6 points in accuracy.

**Thesis Committee:**

Chair: Prof. Avishek Anand, Web Information Systems, TU Delft
Supervisor: Venktesh Viswanathan, Web Information Systems, TU Delft
Committee Member: Prof. Pradeep Murukannaiah, Interactive Intelligence, TU Delft

# Preface

Embarking on this thesis journey has been an enriching and transformative experience, one that I could not have undertaken without the support and guidance of several individuals whom I am deeply grateful to acknowledge.

First and foremost, I extend my sincerest gratitude to my thesis advisor, professor Avishek Anand, for providing me with the opportunity to delve into this fascinating research as well as the constructive feedback that has been instrumental in shaping the course of this thesis. Likewise, I am equally indebted to my daily supervisor, Venktesh Viswanathan, whose guidance and expertise have been invaluable throughout every stage of this journey and helped me out in striving for excellence.

Furthermore, I extend my heartfelt appreciation to my family and friends as a constant source of motivation for their unwavering encouragement, patience, and unwavering support. Moreover, I extend my sincere thanks to the Web Information Systems group members and fellow master students who participated in the ELIXER reading group, whose insightful discussions and shared knowledge broadened my understanding of the field. Lastly, I also want to thank professor Pradeep Murukannaiah for being a member of the thesis committee.

In the end, this thesis is the culmination of countless hours of research, collaboration, and reflection, and it is with immense gratitude that I acknowledge the contributions of all those who have played a part in its realization.

Kevin Nanhekhan
Delft, the Netherlands
April 19, 2024

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Research Motivation

Detecting misinformation and disinformation is of critical importance in the digital era. Fact-checking forms a crucial process for confirming the veracity of claims made on the web to detect misinformation. Initially, this was done through manual annotation by a small group of professionals and later on crowd-sourced workers which is expensive and cumbersome [39]. This is primarily because fact-checking involves researching existing evidence, understanding the context, and reasoning about the implications of the evidence. With the huge growth of misinformation growing larger, manual fact-checking can thus not keep up in speed and capacity [12, 37]. Additionally, since fact-checking is done through human annotators, due to differences between individual ideals and opinions their work is still susceptible to human biases.

Therefore, automated fact-checking is critical to mitigating the negative consequences of misinformation. Typically, an automated fact-checking pipeline entails several stages, namely claim detection, retrieval of evidence, re-ranking of the retrieved evidence to get the claim-worthy ones, and performing the verification using a Natural Language Inference model [12, 23]. For retrieval of evidence from a collection, traditional approaches use term-based retrieval approaches [17]. However, this may result in sub-optimal results due to a lack of overlapping terms between passages and the claim. While significant attention has been devoted to fact verification, the retrieval stage has received less focus, particularly in real-world scenarios. Existing works for retrieval often rely on artificially constructed experiment settings that may not accurately reflect real-world conditions [37]. This is due to the assumption that the retrieval of relevant documents is perfect, potentially limiting their applicability in practical settings where this assumption may not hold. Fact-checking in the real world often involves accessing vast data collections such as Wikipedia. However, this process presents significant scalability challenges, primarily due to the extensive resources required for indexing entire Wikipedia articles alongside their corresponding embeddings. Consequently, this approach is computationally expensive and can cause significant latency during inference time [49], which poses limitations for deploying production-grade fact-checking systems. Thus as the retrieval stage forms the bottleneck in fact-checking pipelines, we propose some retrieval approaches aimed at efficiency that apply to real-world applications.

To guide us in research, we investigate the following research questions.

- RQ 1: How does indexing supporting facts improve information retrieval efficiency?
- RQ 2: How does indexing supporting facts affect overall pipeline efficiency and downstream fact-checking performance?
- RQ 3: In what ways does index compression enhance the efficiency of dense retrieval and fact-checking systems?

## 1.2 Scientific Contributions

The main contributions of our work can be summarised in the following:

- We introduce techniques for identifying supporting facts relevant to the claims being fact-checked. These mechanisms are designed to efficiently extract pertinent information from large corpora, enabling more targeted retrieval and verification processes.
- We use and show methods for indexing supporting facts in a manner that optimizes retrieval efficiency. By selectively indexing relevant information, our approach reduces computational overhead and accelerates the retrieval process, thereby enhancing overall pipeline performance.
- Our study includes a comprehensive evaluation of various retrieval approaches, mainly showing the difference in indexing supporting facts versus indexing the entire data corpus. We meticulously analyze performance metrics at each step of the fact-checking pipeline, offering detailed insights into the effectiveness and efficiency of different experimental configurations.
- In addition to our research findings, we contribute to the scientific community by providing access to the supporting facts datasets we constructed and the codebase[1] used in our study. This includes modifications to existing algorithms used in our work, as well as the implementations for identifying, extracting, and processing supporting facts within diverse data collections such that it can be used in a fact-checking pipeline.

## 1.3 Thesis Outline

The report is laid out the following way. Chapter 2 delves into the existing body of literature and research about the challenge of hallucinations in generated content and the field of fact-checking. Chapter 3 lays out the primary methodology of our work, introducing the improvements we aimed to explore and test. The ensuing Chapter 4 establishes everything concerning the various aspects of the executed experiments, ranging from providing information on the used resources to fines implementation details. Following the experiments, Chapter 5 lays out the findings of the experiments coupled with detailed answers to our research questions. Lastly, Chapter 6 serves as the conclusion, summarising our key findings and also putting forward possible research directions for future work.

[1] https://github.com/kevin-rn/Efficient-Fact-checking

# Chapter 2

# Related Work

This chapter discusses previous work that is related to the hallucination problem and a breakdown of definitions and such. Additionally, the current work that aims to mitigate that problem is also laid out, where important works are named and further explained.

## 2.1 Fact-checking

The notion of fact-checking delves into the logic, coherence, and context of claims [39]. In the fact-checking process, fact verification serves as a crucial preliminary step in acquiring and confirming facts, ensuring the trustworthiness of the information under consideration. The surge in demand for automated fact-checking has prompted rapid advancements in the development of tools and systems for it. Successful of these pipelines rely on efficient handling of large document collections, optimal text span granularity for detailed answers, contextual awareness for appropriate text granularity selection, and versatility across domains [1]. Currently, the typical fact-checking process involves three stages: (i) *claim detection* involves identifying salient text spans from a large collection; (ii) *evidence retrieval* focuses on finding sources that either support or refute the claim; and (iii) *fact verification* entails assessing the veracity of the claim based on the retrieved evidence [12].

For the claim detection part, there is no formal definition of what constitutes a claim [23]. Some existing work establishes check-worthiness as a possible concept [12]. It determines a claim when one wants to know the truth of that assertion, which either requires binary classification or an importance-ranking to classify. Another method, used in social media settings, is whether text spans are detected for rumourness. Nonetheless, these two methods are quite subjective as the language understanding and importance of the concepts differs between social groups or even individuals [12, 39]. Furthermore, the information pertaining to the claim can change over time or have been debunked already, no longer necessitating the need for verification. A more objective approach is to classify text as an assertion if it's checkable with available evidence.

Early retrieval systems were typically complex, composed of numerous components [20, 49]. However, recent advancements in reading comprehension models advocate for a simplified two-step approach: initially, a context retriever selects a subset of passages, some of which potentially contain the answer to the query, followed by a thorough analysis by a machine reader to identify the correct answer. Nonetheless, a notable challenge arises concerning the sources from which information is pulled [12]. The task of fact-checking requires access to reliable and trustworthy knowledge sources that have been thoroughly verified. These sources serve as the foundation for retrieving evidence-based information. These include a diverse array of textual sources such as encyclopedia articles, policy documents, verified news articles, and scientific journals, which offer rich information for verifying claims [23, 39]. Additionally, knowledge graphs or fact databases provide structured canonical information about the world, though their limitations must be considered, as not all facts may be present in them. Social media and online content analysis offer insights into the veracity of claims, especially when traditional textual or structured knowledge sources are unavailable.

Lastly, for fact verification, either binary classification using supported/refuted labels or Multi-class labels are utilized [12, 39]. The latter mimicking journalistic fact-checking practices, to include more fine-grained classification schemes or indicate when not enough information could be retrieved. Automated fact-checking primarily relies on supervised text classification methods, often using labelled data from fact-checking agencies [39]. While effective for some tasks, it lacks the broader world knowledge necessary for comprehensive fact-checking. Other approaches include network analysis, Recognizing Textual Entailment, and distant relation extraction. Speaker profiling, such as analyzing the credit history of claim originators, can enhance accuracy but raises ethical concerns. To further improve the verdict interpretations, the justification procedure is important [12]. Particularly in automated fact-checking where black-box models lack transparency. Strategies include highlighting salient evidence, designing understandable decision-making processes, and generating textual explanations.

## 2.2 Efficient Retrieval Mechanisms

Efficient Retrieval Mechanisms themselves play a crucial role in not only major web search engines and recommendation systems but also Natural Language Processing tasks [5, 43]. With the rise of the Web and the development of early search engines, early approaches concerned that of inverted indexes, which efficiently stored term-document relationships, enabling rapid retrieval of documents containing specific terms. Examples are TF-IDF and BM25. However inverted indexes only retrieve text-like keywords, but do not include semantic recognition. Another approach was that of the Vector Space Model, which represented documents and queries as vectors in a high-dimensional space and used similarity measures like cosine similarity for ranking documents by relevance. A notable evolution in retrieval mechanisms is observed in Dense Retrieval, where dense encodings replace traditional methods [20]. Dense encodings, being learnable through adjustable embedding functions, offer enhanced flexibility for task-specific representations.

With the exponential growth of digital data, the deployment of models that utilize this data tends to become extensive in size to achieve cutting-edge results, sometimes even exceeding billions of parameters [6, 51]. This expansion brings about costly repercussions: a larger memory footprint in terms of storage space and prohibitively large energy consumption. Furthermore, traditional retrieval techniques have encountered scalability challenges leading to higher latency in inference time, particularly due to the limitations posed by the needed high-dimensional vectors, known as the dimensionality catastrophe [14, 43]. Therefore, developing techniques or algorithms to create more compact representations of data vectors is crucial [6, 29]. This does come with the possible caveat of introducing some unwanted noise to achieve compressed vectors which results in some performance degradation. The goal, therefore, is to make more efficient representations by reducing the size of data structures without compromising too much of the integrity of the information they encapsulate. This optimization not only contributes to more effective storage and retrieval of data, but also due to lesser data complexity facilitates faster processing and transmission of information [45, 51]. As a solution, vector databases that employ Approximate Nearest Neighbors (ANN) search have emerged as a solution to alleviate complexity and enhance the accuracy of similarity searches for vectors [14, 43, 50].

Early on hash-based approaches [14, 29], such as Locality-Sensitive Hashing, spectral hashing deep hashing, were adapted for ANN in text retrieval. These functions aimed to map similar data points to the same bucket with high probability, enabling efficient approximate retrieval in a reduced space. For faster search, Tree-based structures, like KD-trees and Ball trees, were employed to organize text data for efficient ANN by partitioning the data space into subsets. While these two approaches achieve fast high-recall searches, these do come at the cost of being extremely expensive when handling very large-scale databases. Additionally, there are still problems concerning the lack of sufficient semantic features, slow response times, and information loss on the query side [50, 29]. More recently, word and document embeddings (such as Word2Vec and BERT embeddings) gained prominence as these dense vector representations captured semantic information and enabled efficient ANN-based text retrieval. Metric learning techniques were applied to learn similarity metrics that better captured semantic relationships in text data, further improving the quality of ANN-based retrieval. However, to go one step further, Quantization, which is widely employed in state-of-the-art systems, can be applied as it places a significant emphasis on reducing the precision of numerical representations. This involves transforming what are mostly high-dimensional vectors into low-dimensional ones, enhancing efficiency in terms of storage and computational requirements for retrieval systems [6]. While the aforementioned performance degradation is inevitable, meticulous quantization techniques can achieve significant model compression with minimal loss [51].

A pioneering method within Quantization involves that of Product Quantization (PQ) [14, 18, 29, 45]. PQ is a method that encodes high-dimensional vectors by breaking them into $m$ smaller subvectors known as codebooks each containing $k$ codewords. This effectively partitions the vector space into smaller cells, allowing for fast Euclidean distance computation through precomputed distances. A further improvement is introduced in Optimized Product Quantization (OPQ) [11] which applies orthogonal transformations to vectors as preprocessing, improving the degree of independence and overall performance. In contrast to concatenation, Non-orthogonal quantization methods like Residual Vector Quantization (RVQ), Additive Quantization (AQ), and Composite Quantization (CQ) approximate vectors as sums of codewords, boosting approximation accuracy while maintaining efficient procedures. RVQ iteratively quantizes approximation residuals from previous iterations, while AQ, though slower, offers minimal compression errors without constraints on codewords. Lastly, there is also JPQ [45] that tries to overcome the delayed information retrieval capabilities in existing compression methods. This is by bridging the gap in the separation between encoding and compression training that is present in the existing methods. For our work, JPQ would be the most relevant one as we intend to optimize the efficiency of the index while still achieving similar performance for evidence retrieval using the claim as the query.

## 2.3 Factual Consistency in LLMs

With the huge growth of open source and commercial Large Language Models (LLM), the adoption of these models has increased exponentially. These models deliver impressive performance on a wide range of natural language tasks [31, 32, 41]. Hence, there has been an increase in reliance on LLMs to deliver precise information needs. However, when generating responses, LLMs tend to hallucinate and generate factually inconsistent outputs which deviate from established world knowledge [4, 16, 25]. These errors are imperceptible, as the generated text is seemingly plausible [48]. The limited factual consistency frequently hinders the wide-spread acceptance of these generative LLMs in real-world applications, such as in summarizing patient information in healthcare [16, 36] or question answering in critical scenarios such as healthcare [42] or legal proceedings. The hallucinations raise safety concerns for said applications, the spread of misinformation and disinformation on the web to be further exacerbated, and potential privacy violations.

Studies have demonstrated that LLMs have sufficient parameters to encode large volumes of information, serving as a proxy for knowledge bases [2, 15]. However, with the rate of growth of information, it is not possible for LLMs to encode factual knowledge with temporal aspects where facts change over time [27, 38]. To tackle these limitations, there have been growing efforts to augment LLMs with external knowledge [13, 35, 47], which in general can be divided into two broad categories:

- The first category consists of retrieve-and-generate language models, based on Retrieval Augmented Generation (RAG) [24], which includes a knowledge retriever and a generative model that utilizes gathered evidence for text generation tasks [13, 24]. Some further improved RAG systems involve REALM [13] and RetGen [47].
- The second category involves k-Nearest Neighbor LMs, such as RETOMATON [3] and COG [22], which interpolate the next token probabilities of a pre-trained LM using a K-NN model [21, 26]. The model computes the k-nearest neighbours based on the distance from the representation of the generated prefix. Overall k-NN LMs have been found to have better performance compared to the regular generative task.

Besides incorporating external knowledge, some efforts have also gone to post-hoc attribution and edit methods like RARR [10] and PURR [8]. These aim to retrieve relevant evidence for the output from an LLM followed by an editing approach to ensure the LLM-generated output is factually consistent with the evidence.

To evaluate the factual consistency of these frameworks and LLM-generated texts in general, various benchmarks and metrics have been proposed such as AtrributionScore [44], FactScore [28], HaluEval benchmark collection [25] and LLM-augmenter [33]. However, a common fallacy present in existing work is that most efforts towards mitigating the hallucination problem focus on assessing the fact verification part rather than also focusing on the evidence retrieval part. Here, the aforementioned approaches are expensive as they require fine-tuning multiple large-scale models that are involved in the retriever and the generator components. This leads to the retrieved context also increasing the latency at inference time. Additionally, The system is tightly coupled, as the optimization of the generative model depends on the quality of the output (relevant documents) of the retrieval module [47]. These problems make the use of LLMs that leverage external knowledge susceptible to scalability issues in practical applications. Therefore to close this gap, our contributions involve extending the evaluation process to include measuring the efficiency of the fact-checking process. Consequently, this can help these systems to be more of practical value for real-world use cases.

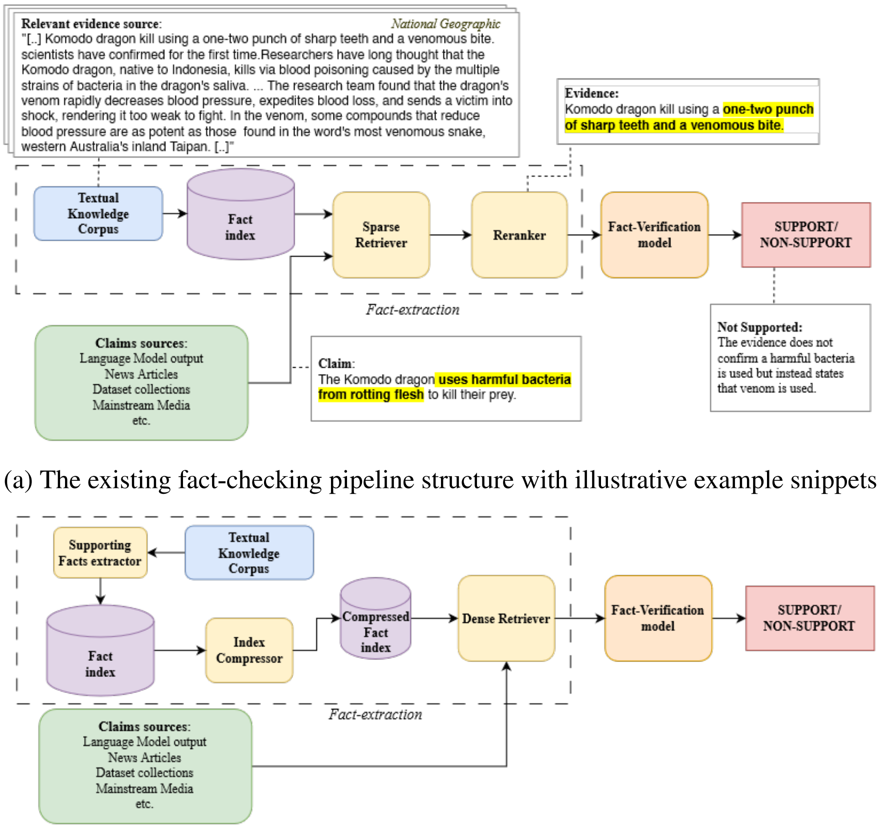
## 2.4 Related work relevancy

In the sections of this chapter, we have explored the main topics related to the different components that make up the general fact-checking process. A further look has been taken at the topic of retrieval mechanisms, a crucial component of the fact extraction phase of the pipeline. Additionally, the factual consistency in Large Language Models has also been explored in this chapter as an emerging application for which fact-checking has gained paramount importance. While our work does not propose any novelties, it does introduce a fresh perspective by combining existing research explored throughout this chapter. Key themes such as the general stages of a fact-checking pipeline, different retrieval methods and their benefits and index compression methods to condense vector embeddings, are all central to the main experiments conducted in this study. Showcasing their improvements in the overall efficiency while still retaining a certain performance output, underscores their significance in addressing the challenges posed by claims that contain misinformation.

# Chapter 3

# Methodology

In Chapter 2, an in-depth exploration of prior research has been presented, laying the groundwork for our study. Taking that knowledge, the main methodology will be explained in this chapter. This will be done by elaborating further upon the key contribution of our work.



**Relevant evidence source:** *National Geographic*
"[..] Komodo dragon kill using a one-two punch of sharp teeth and a venomous bite. scientists have confirmed for the first time.Researchers have long thought that the Komodo dragon, native to Indonesia, kills via blood poisoning caused by the multiple strains of bacteria in the dragon's saliva. ... The research team found that the dragon's venom rapidly decreases blood pressure, expedites blood loss, and sends a victim into shock, rendering it too weak to fight. In the venom, some compounds that reduce blood pressure are as potent as those  found in the word's most venomous snake, western Australia's inland Taipan. [..]"

**Evidence:**
Komodo dragon kill using a **one-two punch of sharp teeth and a venomous bite**.

**Claim:**
The Komodo dragon **uses harmful bacteria from rotting flesh** to kill their prey.

**Not Supported:**
The evidence does not confirm a harmful bacteria is used but instead states that venom is used.

(a) The existing fact-checking pipeline structure with illustrative example snippets.

(b) The proposed fact-checking pipeline structure using just supporting facts in the index to achieve efficiency.

Figure 3.1: Comparison of Existing and Proposed Fact-Checking Pipelines.

## 3.1 Problem Statement

The presence of misinformation in claims poses a significant challenge that needs to be addressed. There are numerous ways to face this challenge. Therefore for our research, we will solely focus on enhancing the efficiency of detecting whether claims are supported or not supported through a fact-checking pipeline. Our approach involves constructing an efficient index of facts sourced from known knowledge repositories to facilitate grounding the generated content. Our emphasis lies on optimizing for efficiency in terms of both time and space consumption, enabling the deployment of end-to-end pipelines on less resource-intensive hardware without compromising performance. Thus our primary contribution lies in refining the index retrieval process, while maintaining comparable fact-checking performance. This refinement allows for faster information retrieval, improving the overall efficiency of the pipeline.

A core part of our research revolves around indexing supporting facts. This is illustrated in Figure 3.1 where we display how we intend to change the general existing state-of-the-art pipeline structure (Figure 3.1a) to one with more efficient components (Figure 3.1b). Here for our study, emphasis is put on how to identify relevant text spans, thus filtering out anything that is not a fact to construct a concise fact database. To guide our research, we formulate the following research questions:

- **RQ 1: How does indexing supporting facts improve information retrieval efficiency?**
  Specifically, we seek to understand how incorporating the supporting facts into the indexing process influences the retrieval latency of relevant information about a claim and the index size on disk. Additionally, this also involves utilising different retrieval methods (sparse and dense retrieval) to more generally see the effects it has on the latency. In the end, we want to see to what extent storing just the supporting facts in the index and performing retrieval on it will degrade performance loss. Additionally we also want to showcase the benefits of this approach over the existing approach of storing the entire document corpus as an index.

- **RQ 2: How does indexing supporting facts affect overall pipeline efficiency and downstream fact-checking performance?**
  The focus here shifts from the information retrieval part towards assessing the broader implications of indexing supporting facts on the overall efficiency of the fact-checking pipeline. Here analysing the overall inference time as well as fact-verification performance becomes important. The aim here is to replace parts of the pipeline to make it more efficient in inference and memory footprint, while trying to achieve approximately the same results compared to the original full-pipeline setting.

- **RQ 3: In what ways does index compression enhance the efficiency of dense retrieval and fact-checking systems?**
  In particular to the dense retrieval part, our objective is to leverage an existing index compression technique to significantly diminish the size of the index stored on disk. Furthermore, we aim to evaluate whether comparable retrieval results can be achieved despite the substantial reduction in index size. By exploring the feasibility of maintaining retrieval effectiveness through a significantly smaller index footprint, we aim to not only optimize storage utilization, but also enable faster query processing, ultimately improving overall system efficiency.

By addressing these questions, we aim to provide insights into optimizing the efficiency of fact-checking processes for existing pipelines, ensuring robustness in terms of performance while minimizing resource requirements.

## 3.2 Improving efficiency

In this section, some concepts for improving efficiency are explained. These are the re-ranking setup that tackles selecting the supporting facts, the use of Dense retrieval compared to Sparse retrieval and lastly, Index compression to further improve Dense retrieval.

### 3.2.1 Extracting Supporting Facts

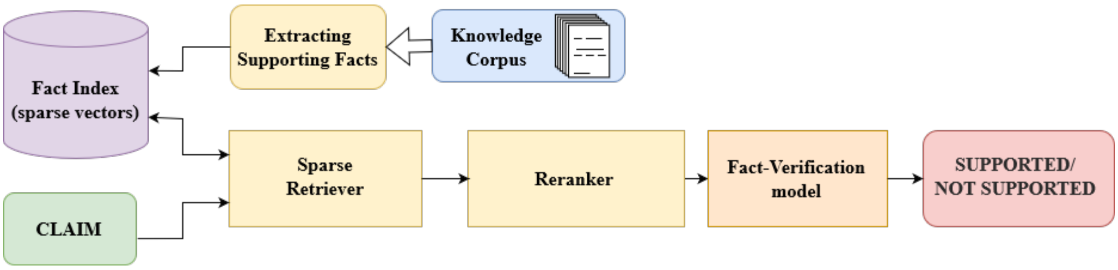


Figure 3.2: The first adjustment to the pipeline using the Re-ranking setup, eliminating the need for fine-grained top-k sentence selection.

A first improvement that could be made is storing only the salient sentences in the data corpus instead of every sentence, as seen in Figure 3.2. As a result, this will reduce the total pipeline runtime drastically and also save disk space for the index. There are various ways to extract supporting facts from a text document from the data corpus. Manually curating the text of the document to only keep the most important ones is the best way to get great performance results. However, this is not only a very time-consuming process, but also does not scale well when using a different data corpus for different use cases.

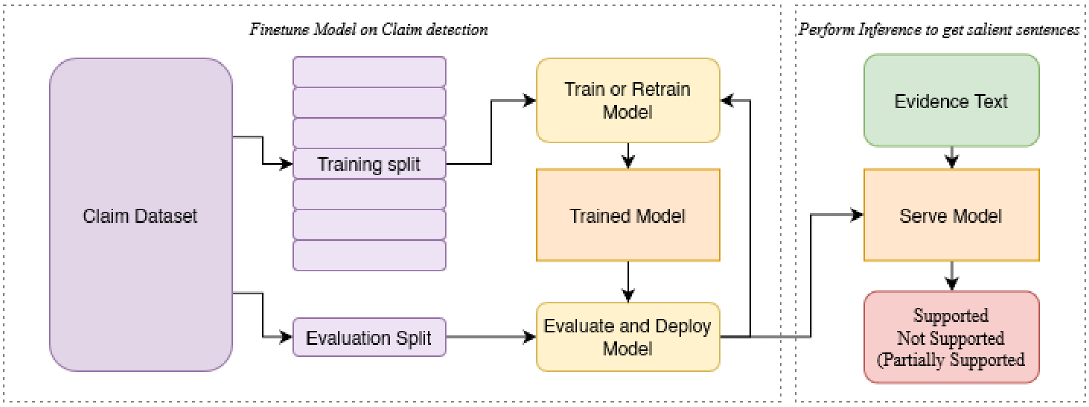
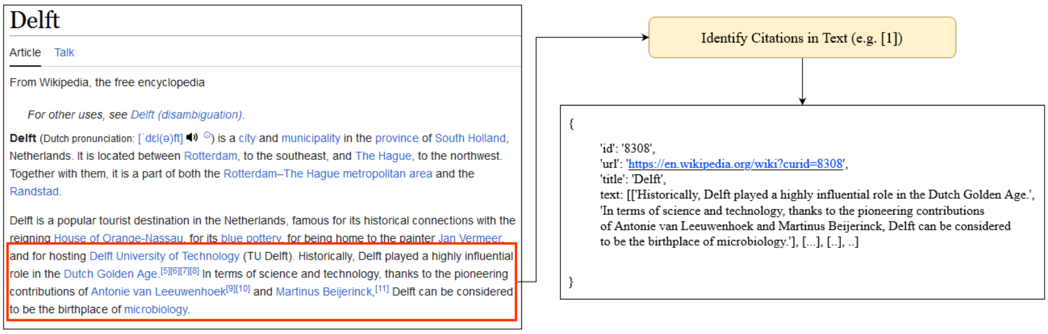


Figure 3.3: A general depiction of claim detection model training and deployment.

**Claim-detection model** As an automated alternative, a claim-detection model can be employed. The idea here is to train a model to specifically detect important sentences out of a text span (see Figure 3.3). Important here is that the model is trained with enough training data to distinguish non-salient from salient sentences. For our experiment, we will be utilising an off-the-shelf pre-trained model as we do not particularly focus on getting the best performance and instead will just be comparing different methods for efficiency. This choice allows us to fully put our efforts towards experimentation and facilitates easy comparison of various methods without the need for extensive training or fine-tuning efforts.



Delft

Article Talk

From Wikipedia, the free encyclopedia

*For other uses, see Delft (disambiguation).*

**Delft** (Dutch pronunciation: [ˈdɛl(ə)ft] 🔊 ⓘ) is a city and municipality in the province of South Holland, Netherlands. It is located between Rotterdam, to the southeast, and The Hague, to the northwest. Together with them, it is a part of both the Rotterdam–The Hague metropolitan area and the Randstad.

Delft is a popular tourist destination in the Netherlands, famous for its historical connections with the reigning House of Orange-Nassau, for its blue pottery, for being home to the painter Jan Vermeer, and for hosting Delft University of Technology (TU Delft). Historically, Delft played a highly influential role in the Dutch Golden Age.[5][6][7][8] In terms of science and technology, thanks to the pioneering contributions of Antonie van Leeuwenhoek[9][10] and Martinus Beijerinck,[11] Delft can be considered to be the birthplace of microbiology.

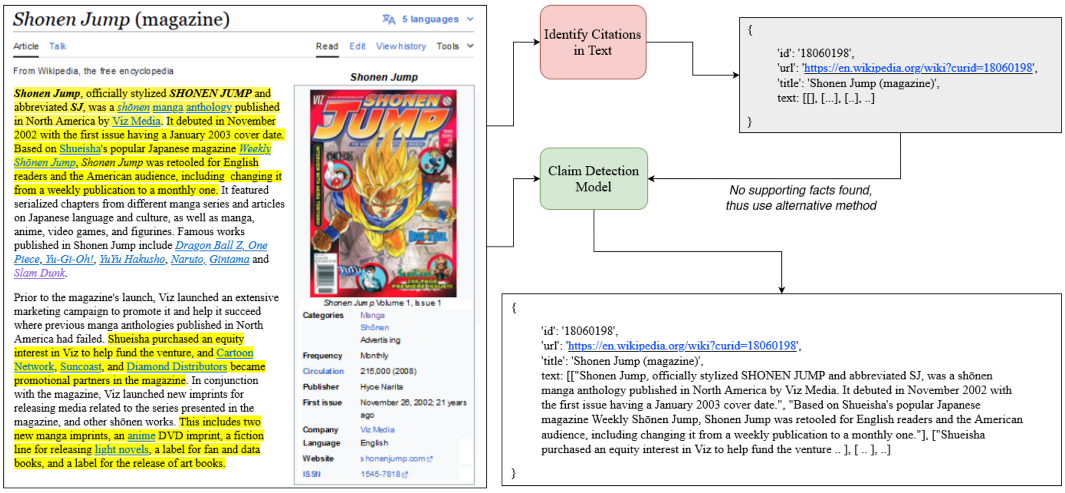
```
{
    'id': '8308',
    'url': 'https://en.wikipedia.org/wiki?curid=8308',
    'title': 'Delft',
    text: [['Historically, Delft played a highly influential role in the Dutch Golden Age.',
    'In terms of science and technology, thanks to the pioneering contributions
    of Antonie van Leeuwenhoek and Martinus Beijerinck, Delft can be considered
    to be the birthplace of microbiology.'], [...], [..], ..]

}
```

Figure 3.4: Citation extraction for the lead section of the Delft Wikipedia page.

**Citation Extraction** An alternative approach for our use case involves sentence extraction based on citations. This concept relies on the underlying assumption that authors include references and citations of reliable sources in parts of the text, that they consider particularly crucial. In other words, citations serve as indicators of claim-worthy sentences within a text. This methodology accelerates the extraction process, eliminating the need to train a model specifically for the task of identifying or selecting claim-worthy sentences. Furthermore, this approach provides a higher level of confidence that the retained sentences are indeed claim-worthy. This stands in contrast to models trained for claim detection or claim extraction, which, despite achieving high accuracy, may still make errors by misclassifying non-salient sentences as salient. A commonly used knowledge data collection is that of Wikipedia, where there are three distinct citation types[1]. These are: 'Inline citations' that place citations close to the text they support, 'In-text attribution' that names the source of a statement, and 'General reference' that is not linked to any particular piece of material. In our use case, as depicted in the Delft Wikipedia page[2] example in Figure 3.4, the strategy involves using the original evidence page and extracting the sentences containing references. These are indicated by its numerical reference notation or footnote. By doing so, we will effectively be able to retrieve the inline and in-text attribution types of citations. However, we will also miss the general reference type as we only extract the closest sentence. In Wikipedia's case, a single cited source can support consecutive sentences, which is indicated by a single citation at the end of the final sentence and thus not for each sentence. In our simplistic automated approach, we will not know what sentences belong to a consecutive sequence and just extract the closest sentence to its citation. Consequently, this will cause our approach to miss the surrounding sentences that may also belong to that citation.

[1] https://en.wikipedia.org/wiki/Wikipedia:Citing_sources

[2] https://en.wikipedia.org/wiki/Delft



*Shonen Jump* (magazine)

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

***Shonen Jump***, officially stylized ***SHONEN JUMP*** and abbreviated ***SJ***, was a *shōnen* manga anthology published in North America by Viz Media. It debuted in November 2002 with the first issue having a January 2003 cover date. Based on Shueisha's popular Japanese magazine *Weekly Shōnen Jump*, *Shonen Jump* was retooled for English readers and the American audience, including changing it from a weekly publication to a monthly one. It featured serialized chapters from different manga series and articles on Japanese language and culture, as well as manga, anime, video games, and figurines. Famous works published in Shonen Jump include *Dragon Ball Z, One Piece*, *Yu-Gi-Oh!*, *YuYu Hakusho*, *Naruto*, *Gintama* and *Slam Dunk*.

Prior to the magazine's launch, Viz launched an extensive marketing campaign to promote it and help it succeed where previous manga anthologies published in North America had failed. Shueisha purchased an equity interest in Viz to help fund the venture, and Cartoon Network, Suncoast, and Diamond Distributors became promotional partners in the magazine. In conjunction with the magazine, Viz launched new imprints for releasing media related to the series presented in the magazine, and other shōnen works. This includes two new manga imprints, an anime DVD imprint, a fiction line for releasing light novels, a label for fan and data books, and a label for the release of art books.

**Shonen Jump**

*Shonen Jump* Volume 1, Issue 1

| | |
|---|---|
| Categories | Manga<br>Shōnen<br>Advertising |
| Frequency | Monthly |
| Circulation | 215,000 (2008) |
| Publisher | Hyoe Narita |
| First issue | November 26, 2002; 21 years ago |
| Company | Viz Media |
| Language | English |
| Website | shonenjump.com |
| ISSN | 1545-7818 |

5 languages
Identify Citations in Text
Claim Detection Model
No supporting facts found, thus use alternative method

```
{
        'id': '18060198',
        'url': 'https://en.wikipedia.org/wiki?curid=18060198',
        'title': 'Shonen Jump (magazine)',
        text: [[], [...], [..], ..]
}
```

```
{
        'id': '18060198',
        'url': 'https://en.wikipedia.org/wiki?curid=18060198',
        'title': 'Shonen Jump (magazine)',
        text: [["Shonen Jump, officially stylized SHONEN JUMP and abbreviated SJ, was a shōnen
        manga anthology published in North America by Viz Media. It debuted in November 2002 with
        the first issue having a January 2003 cover date.", "Based on Shueisha's popular Japanese
        magazine Weekly Shōnen Jump, Shonen Jump was retooled for English readers and the American
        audience, including changing it from a weekly publication to a monthly one."], ["Shueisha
        purchased an equity interest in Viz to help fund the venture .. ], [ .. ], ..]
}
```

Figure 3.5: Fusion setup showing claim detection being used when citation extraction is not possible. Based on the 'Shonen Jump (magazine)' Wikipedia page.

**Fusion of Citation extraction and Claim Detection** While cited sentences are valuable for providing trustworthy support, they may not always be abundant enough to rely on exclusively. While some pages boast numerous well-sourced references, others may have only a handful or even none at all. Additionally, due to the way we extract citations, there may be some surrounding sentences missed that are also part of the citation. In both such instances, citation extraction might overlook valuable claim-worthy information present in the text. Take for example Figure 3.5 showing that the *'Shonen Jump (magazine)'*[3] article does not have any citations in the given text span. Here one can see that while no references are used, there are sentences present that are worthwhile to consider as supporting facts. While preserving the original text could be considered a valid option, it is important to note that for external knowledge sources beyond Wikipedia, citations may only be available for a small percentage of data. As a consequence, this leads to a negligible reduction of the data in those cases. Thus, to optimize efficiency and achieve our end goal, a more effective approach is to utilize claim detection whenever citations have not been extracted. This strategy ensures that entries in the data collection without citations still capture potential supporting facts, while simultaneously reducing the overall corpus size. In this approach, it's important to keep in mind that the inference latency for detecting supporting facts now depends not only on the number of citations present in a corpus, but also on the inference process of the claim detection model.

### 3.2.2 Evidence Retrieval

In addition to employing re-ranking to reduce the corpus size, we can incorporate Dense Retrieval as a more efficient alternative to Sparse Retrieval and re-ranking. This is as depicted in Figure 3.6. Like is mentioned in Section 2.1, the evidence retrieval consists of retrieving data that most likely contains the answer followed by selecting the top relevant ones pertaining to the query/claim. The first step is usually done using a Sparse retrieval setup, as depicted in Figure 3.7a. This setup however is very costly to run, as one would require a module for retrieving

[3] https://en.wikipedia.org/wiki/Shonen_Jump_(magazine)

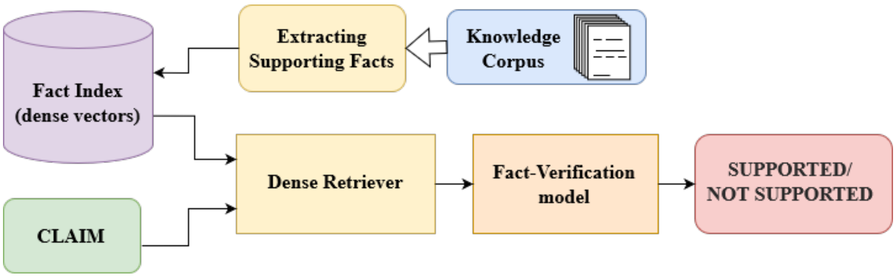
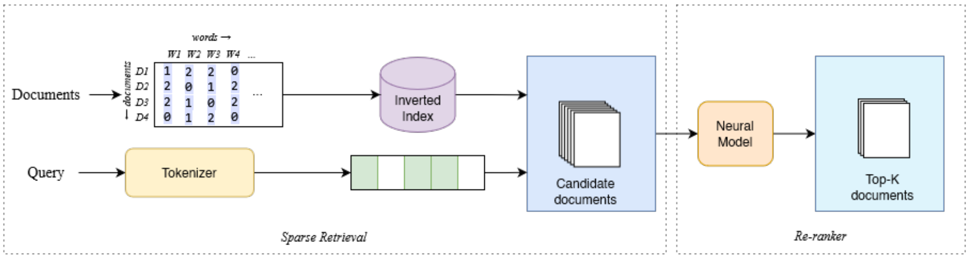
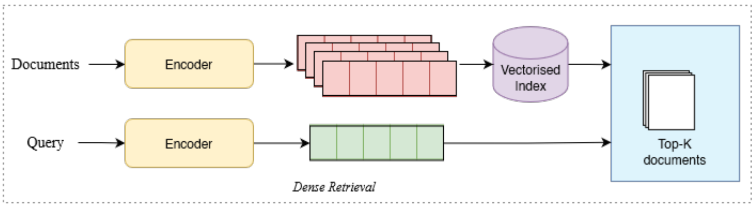


Figure 3.6: The second adjustment to the pipeline in replacing Sparse Retrieval with Re-ranking stages with a single Dense Retrieval module.

and a module for re-ranking. As the re-ranking setup eliminates the need to perform sentence selection to reduce non-relevant sentences, we can essentially directly retrieve the top-k documents from the newly formed data corpus using Dense Retrieval (see Figure 3.7b) This is because we no longer need the other two-step process for iteratively increasing the chances of getting the most relevant documents pertaining to a claim. Additionally, in the current state-of-the-art systems, Neural network-based approaches are used as re-rankers to enhance Sparse retrieval by leveraging techniques like word2vec for semantic similarity assessment, pre-trained models such as BERT for predicting term importance, and term expansion methods to enhance recall. These advancements significantly augment traditional methods by providing more nuanced understandings of query-document relationships and improving retrieval performance from a semantic aspect. Here Dense retrieval can potentially benefit the fact-checking pipeline, as it can immediately in its first stage retrieval give high similarity scores in the to semantically relevant text pairs, even without exact token matching [20].



(a) Sparse Retrieval with Re-ranker



(b) Dense Retrieval

Figure 3.7: Retrieval methods for retrieving top-k relevant documents pertaining to a query.

Although implementing Dense Retrieval offers a more streamlined approach, it is important to also acknowledge its limitations compared to the original two-step retrieval approach. A notable limitation is that the dense representation is typically inferior to the sparse representation of data [20]. To elaborate, the use of Dense Retrieval necessitates the creation of vector embeddings capable of encapsulating the entirety of semantic information within the text. This task becomes very challenging, particularly for extensive textual spans. Thus, while Dense Retrieval enhances overall computational efficiency by simplifying the process, it may not necessarily surpass the original pipeline's retrieval methods due to the complexity of capturing comprehensive semantic representations. However, we do expect the performance to be within a few points difference.

### 3.2.3 Index Compression

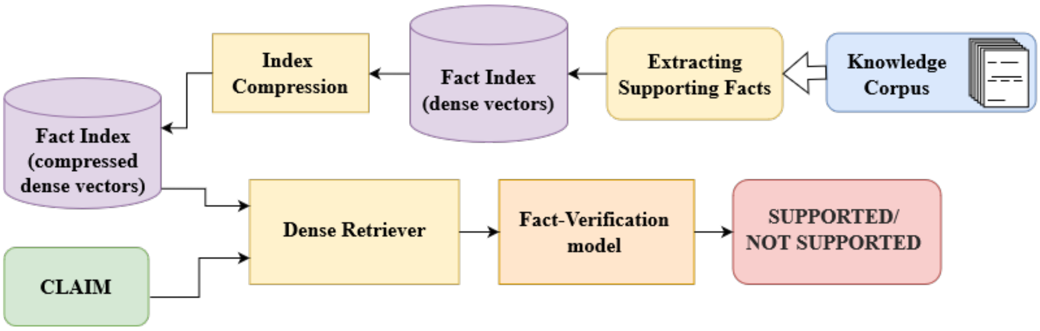


Figure 3.8: The third adjustment to the pipeline with the added Index Compression module to enhance Dense Retrieval.

As described in Section 2.2, leveraging Quantization can significantly enhance efficiency in the Dense Retrieval setup. In our research, we will adopt the JPQ model introduced by Zhan et al. [45] into the fact-checking pipeline (see Figure 3.8). JPQ, short for *Joint optimization of query encoding and Product Quantization*, offers a novel approach by optimizing ranking performance in an end-to-end manner, departing from the traditional encoding-compression two-step process. Conventional compression techniques indeed enhance the efficiency of Dense Retrieval. However, these methods typically rely on a task-independent reconstruction error as the loss function during training. Moreover, the encoders and compressed index are trained separately. As a consequence of these design choices, compression fails to capitalize on supervised information, and the compatibility between the query-doc encoders and the compressed index might not be optimally aligned, overall leading to sub-optimal performance.

To resolve this, JPQ trains the query encoder and PQ index jointly based on three optimization strategies.

1. First, the model employs a ranking-oriented loss to precisely measure the disparity between the PQ index and dual encoders. Computation of this loss involves reconstructing the quantized document embeddings, denoted as $d^{\dagger} = c_{1,\varphi_1(d)}, c_{2,\varphi_2(d)}, \ldots, c_{M,\varphi_M(d)}$. This is then used in embedding similarity computation between the query and document, to derive the relevance scores $s^{\dagger}(q,d) = \langle q, d^{\dagger} \rangle$ utilized by PQ for ranking purposes. Subsequently, these scores are fed into a pair-wise loss function to compute the loss accurately:

$$\ell(s^{\dagger}(q,d^{+}), s^{\dagger}(q,d^{-})) \tag{3.1}$$

2. Second, PQ centroid optimization is used to train the PQ index with the ranking-oriented loss. This would otherwise be non-trivial due to problems related to differentiability and the substantial number of index assignments (proportional to the corpus size) leading to potential overfitting. The PQ centroid optimization approach initializes a small set of PQ centroid embeddings and updates the embedding through gradient descent, defined as:

$$\frac{\partial \ell(s^\dagger(q,d^+), s^\dagger(q,d^-))}{\partial c_{i,j}} = \begin{cases} -\alpha \vec{q}_i, & \text{if } j = \varphi_i(d^+), j \neq \varphi_i(d^-) \\ \alpha \vec{q}_i, & \text{if } j \neq \varphi_i(d^+), j = \varphi_i(d^-) \\ 0, & \text{if } j = \varphi_i(d^+), j = \varphi_i(d^-) \\ 0, & \text{if } j \neq \varphi_i(d^+), j \neq \varphi_i(d^-) \end{cases} \tag{3.2}$$

   These embeddings are differentiable and due to their compact size avoid the problem of overfitting. The PQ centroid optimization strategy benefits from supervised signals guiding updates within the PQ index compared to other existing approaches. Additionally, PQ parameters directly evolve the query encoder through $\vec{q}_i$, which is part of the encoder's output, ensuring efficient refinement and adaptation.

3. Finally, incorporating end-to-end negative sampling can further bolster ranking performance. Previous research by the authors [46] demonstrated that dynamic hard negative sampling contributes to enhancing top-ranking performance. This technique involves penalizing the scores of top-ranked irrelevant documents treated as negatives. The rationale behind this approach lies in the significant impact that top-ranked negatives have on ranking performance, whereas low-ranked documents are often cut-off by truncated evaluation metrics. The process entails the real-time retrieval of negative samples by leveraging current PQ parameters to extract the top-$\hat{n}$ irrelevant documents as negatives at each training step. The incorporation of retrieved negatives ($D_q^{-\dagger}$) helps minimize the top-$\hat{n}$ pairwise errors, thereby aligning with the truncated evaluation metric. The formulation for $D_q^{-\dagger}$, with C being the entire document corpus and $D_{q^\dagger}^{+}$ the labeled relevant documents, can be expressed as:

$$D_q^{-\dagger} = \text{sort}\left(d \in C \setminus D_{q^\dagger}^{+} \text{ based on } s^\dagger(q,d)\right)[:\hat{n}] \tag{3.3}$$

In essence, JPQ jointly optimizes the query encoder and PQ Centroid Embeddings through end-to-end negative sampling and ranking-oriented loss computation. The entire optimization objective of JPQ can thus be formulated as follows:

$$f^*, \{c_{i,j}\}^* = \arg \min_{f,\{c_{i,j}\}} \sum_q \sum_{d^+ \in D_q^+} \sum_{d^- \in D_q^-} \ell(s^\dagger(q,d^+), s^\dagger(q,d^-)) \tag{3.4}$$

Given our reliance on establishing an efficient index for accurate retrieval, the work by JPQ [45] aligns closely with our work. Having optimally compatible trained dual-encoders with a PQ index can help ensure the corpus index footprint in memory is small. This can be seen in the reported compression ratio of *4D/M*, where $D$ represents the vector dimensionality and $M$ denotes the number of codebooks used for compression. Concurrently, this approach facilitates efficient retrieval of supporting facts while maintaining performance levels comparable to those of standard Dense Retrieval setups, albeit without the overhead of a compressed index. This is evidenced by the reported speedup ratio of $(D + \log n)/(M + \log n)$, with $n$ being the total number of documents in the index.

# Chapter 4

# Experiments

This chapter delineates the datasets utilized. Furthermore, it delves into the intricacies of each stage within the end-to-end fact-checking pipeline, elucidates the configuration and setup employed for the experiments, and finally, provides an explanation of the assessment criteria and details the measurements taken during the experiment runs.

## 4.1 Datasets

Our experiments utilize two state-of-the-art comprehensive claim datasets for evaluating the fact-checking pipeline, namely HoVer [17] and WiCE [19]. These datasets pose challenges to textual entailment models in evidence extraction and fact verification due to their intricate multi-hop reasoning requirements. Besides the claim data, also required is the Wikipedia data dumps as a vast source of evidence data collection. The distribution of entailment labels across dataset splits for both HoVer and WiCE, as utilized in our experiments, is illustrated in Table 4.1.

| Claim Data | | Supporting | Non-Supporting | Total |
|---|---|---|---|---|
| HoVer | Train | 11023 (61%) | 7148 (39%) | 18171 |
| | Dev | 2000 (50%) | 2000 (50%) | 4000 |
| | Test | 2000 (50%) | 2000 (50%) | 4000 |
| WiCE | Train | 460 (37%) | 800 (63%) | 1,260 |
| | Dev | 115 (33%) | 234 (67%) | 349 |
| | Test | 111 (31%) | 247 (69%) | 358 |

Table 4.1: Sizes of the claim dataset splits as used in our experiments.

### 4.1.1 HoVer dataset

The work by Jiang et al. [17] will be used. The paper with regards to other datasets states that, while valuable for community engagement, single-hop datasets (e.g. FEVER) and current multi-hop question-answering datasets (e.g. HotPotQA) face limitations related to the number of reasoning steps and word overlap between questions and evidence. Therefore the paper introduces HOppy VERification (HoVer) consisting of 26k claims that require evidence from up to four English Wikipedia articles. Additionally, those claims contain significantly less semantic overlap between the claims and some supporting documents. This is to avoid reasoning shortcuts such as shallowly performing direct semantic matching with only the claims. The data corpus comprises only the first Wikipedia paragraph (the lead section). To stay in line, we will do the same and also add another setup of using the whole Wikipedia article text.

The HoVer dataset is curated using the HotPotQA[1] dataset as a basis, which consists of multi-hop question-answer pairs derived from the 2017 English Wikipedia dump. The creation of the HoVer dataset unfolds in three stages, guided by the collaborative efforts of trained crowd-workers.

1. **Claim Creation:** In the initial stage, question-answer pairs sourced from HotPotQA are rewritten into claims. These claims incorporate information from two distinct English Wikipedia articles. To enhance complexity, extra hops are introduced by substituting an entity with information from another article that pertains to the original entity. To improve readability, many-hop claims are articulated across multiple sentences, interconnected through coreferences. To ensure the quality of the claims, a separate group of crowd workers validate the claims. Here only claims for which at least two out of three annotators agree on a valid statement and cover the same information from the original question-answer pair are kept.

2. **Claim Mutation:** The second stage involves the generation of unsupported claims by modifying those produced in the first stage. This process includes automatic word/entity substitution, followed by human editing. Crowd-workers refine claims to make them either more specific or more general, and some claims are negated. A human validation process ensures the quality of machine-generated claims.

3. **Claim Labeling:** In the third stage, a binary classification system is adopted for claim labels: SUPPORTED and NOT-SUPPORTED. Recognizing the inherent ambiguity in distinguishing between REFUTED and NOTENOUGHINFO, particularly in many-hop claims, the label NOT-SUPPORTED is employed. This decision streamlines the categorization process and addresses challenges in determining refutation or insufficient information.

### 4.1.2 WiCE dataset

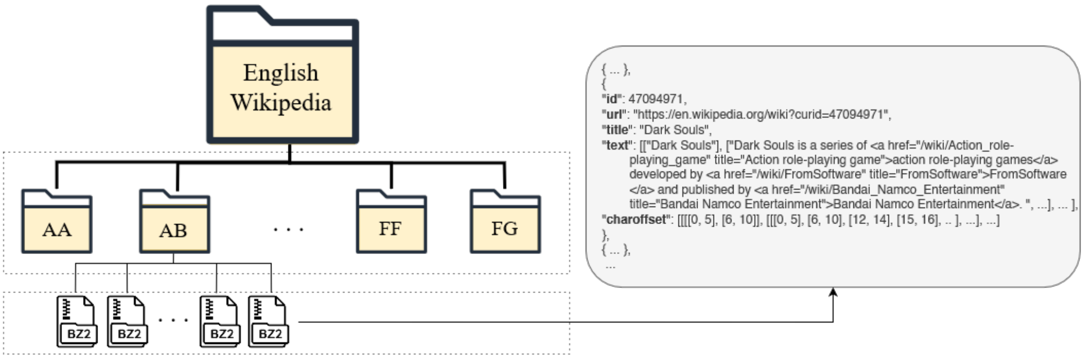
Besides HoVer, we also aim to use the WiCE dataset proposed in the work of Kamoi et al. [19]. WiCE, an acronym for *Wikipedia Citation Entailment* is a dataset consisting of real-world claims from Wikipedia articles. The dataset aims to mitigate the challenges that arise for modern entailment systems. This is due to Natural Language Inference (NLI) datasets having a limited analysis scope due to too much focus on short premises, causing entailment systems to rely solely on local scores or retrieval methods. There are a few exceptions like DocNLI, which has synthetic negatives. However, this introduces a different challenge of spurious correlations and insufficient annotations of claim support due to a lack of ecologically valid negatives hampering the training. These gaps highlight the need for more robust methodologies and diverse training

[1] https://hotpotqa.github.io/wiki-readme.html

data for better inference accuracy.

The WiCE dataset uses the same base claim data as SIDE [34] and consists of sentences constructed from Wikipedia together with the corresponding articles it cites. To help in the annotation process, the authors introduce the Claim-Split method, which decomposes hypotheses into sub-claims using few-shot prompting with ‘GPT-3.5’. By breaking down long claims into short sub-claims, they manage to not only simplify the annotation process, but also simplify the entailment prediction task and enhance the classification performance. Additionally, this method also allows for a more fine-grained view of which parts of the claims are supported. For the annotation process itself, attained are the entailment label, the list of the article’s sentences that support the claim sentence, and tokens in the claim that are unsupported.

For our experiment evaluation, we conducted the pipeline analysis on the WiCE claim dataset rather than their sub-claim dataset. This decision was made because the sub-claims often lack the necessary surrounding context to be considered claims in their own right. This limitation becomes apparent when considering an example claim: *”The couple married in New York in December 2012, and their son, Bear, was born the next year,”* which pertains to the actress Kate Winslet. However, the corresponding sub-claim: *”The couple married in New York in December 2012.”* is highly ambiguous and tends to yield non-relevant documents when used in isolation. Another decision made concerning the dataset is the change of labelling. WiCE utilizes a three-way entailment system, which we adapted into a binary labelling scheme. It’s important to acknowledge that many of the unsupported claims in the dataset are likely true. However, they are labelled as unsupported due to the lack of specific evidence documents. Furthermore, the original partially supported label can be considered ambiguous, because it doesn’t necessarily imply that all parts of the claim are verified by the retrieved evidence documents. Therefore, in line with the principles of the HoVer dataset, we adapted the instances labelled as ‘PARTIALLY-SUPPORTED’ to ‘NOT-SUPPORTED’. This adjustment ensures consistency and clarity in our evaluation process.



```
{ ... },
{
"id": 47094971,
"url": "https://en.wikipedia.org/wiki?curid=47094971",
"title": "Dark Souls",
"text": [["Dark Souls"], ["Dark Souls is a series of <a href="/wiki/Action_role-
      playing_game" title="Action role-playing game">action role-playing games</a>
      developed by <a href="/wiki/FromSoftware" title="FromSoftware">FromSoftware
      </a> and published by <a href="/wiki/Bandai_Namco_Entertainment"
      title="Bandai Namco Entertainment">Bandai Namco Entertainment</a>. ", ...], ... ],
"charoffset": [[[[0, 5], [6, 10]], [[[0, 5], [6, 10], [12, 14], [15, 16], .. ], ...], ...]
},
{ ... },
 ...
```

Figure 4.1: Layout of processed Wikipedia dump used for our experiments.

### 4.1.3 Wikipedia corpus

As both of the datasets contain claims with supporting documents related to Wikipedia articles, we used those Wikipedia dumps as corpus collection to ensure relevant documents were retrieved. For HoVer, as mentioned before, we used the same processed 2017 English Wikipedia dump, retrieved from the HotPotQA website. This dump consists of in total of 5,486,211 Wikipedia articles which in total constitute a total of 94,914,378 sentences. Similarly, the WiCE dataset also uses an English Wikipedia where they re-retrieved relevant articles using Common Crawl. Instead of doing the same and to speed up the process, we used one of the latest available English Wikipedia dumps at the moment (namely 01-Jan-2024). This dump consists of in total of 6,777,401 Wikipedia articles which constitutes a total of 126,533,841 sentences. For processing the corpus to be of the same format as the HotPotQA Wikipedia dump, we utilised the HotPotQA fork[2] of the official Wikiextractor[3] tool. This tool helps format the dump into a folder structure, as seen in Figure 4.1, containing multiple sub-folders holding one hundred Bzip2 files, where each zip file contains multiple Wikipedia articles.

## 4.2 Experimental design

It is crucial to establish a baseline for our experiments amidst the myriad of discussed approaches for improvements and setups. Additionally, we provide an overview of the various ablations and other studies conducted for our experiments, as to offer a more streamlined perspective.

### 4.2.1 Fact-checking Pipeline

The authors of HoVer have developed an initial baseline pipeline system for fact extraction and claim verification, following the state-of-the-art model by Nie et al. [30] on FEVER [40], to highlight the complex multi-hop challenge. This pipeline[4] forms the core part of our work as we will be comparing the performance and efficiency of it, to the improvements we will introduce. The pipeline, depicted in Figure 4.2 consists of the following stages:

1. **Term-based Document Retrieval:** The system employs Chen et al.'s [9] document retrieval component, utilizing cosine similarity on binned uni-gram and bi-gram TF-IDF vectors. This process identifies and returns the k most relevant Wikipedia documents and their probabilities for a given query.

2. **Neural-based Document Retrieval:** The BERT-base model utilizes a single document p from the set Pr and a claim c as inputs. It produces a relatedness score reflecting the connection between p and c. The system selects a set Pn consisting of the top kp documents with relatedness scores surpassing a threshold kp.

3. **Neural-based Sentence Selection:** Another BERT-base model is fine-tuned to encode the claim c and all sentences from a chosen document p in the set Pn. It predicts sentence relatedness scores using the first token of each sentence. A set Sn is then selected, comprising the top sentences from Pn with relatedness scores exceeding a threshold ks.

[2] https://github.com/qipeng/wikiextractor

[3] https://github.com/attardi/wikiextractor

[4] https://github.com/hover-nlp/hover

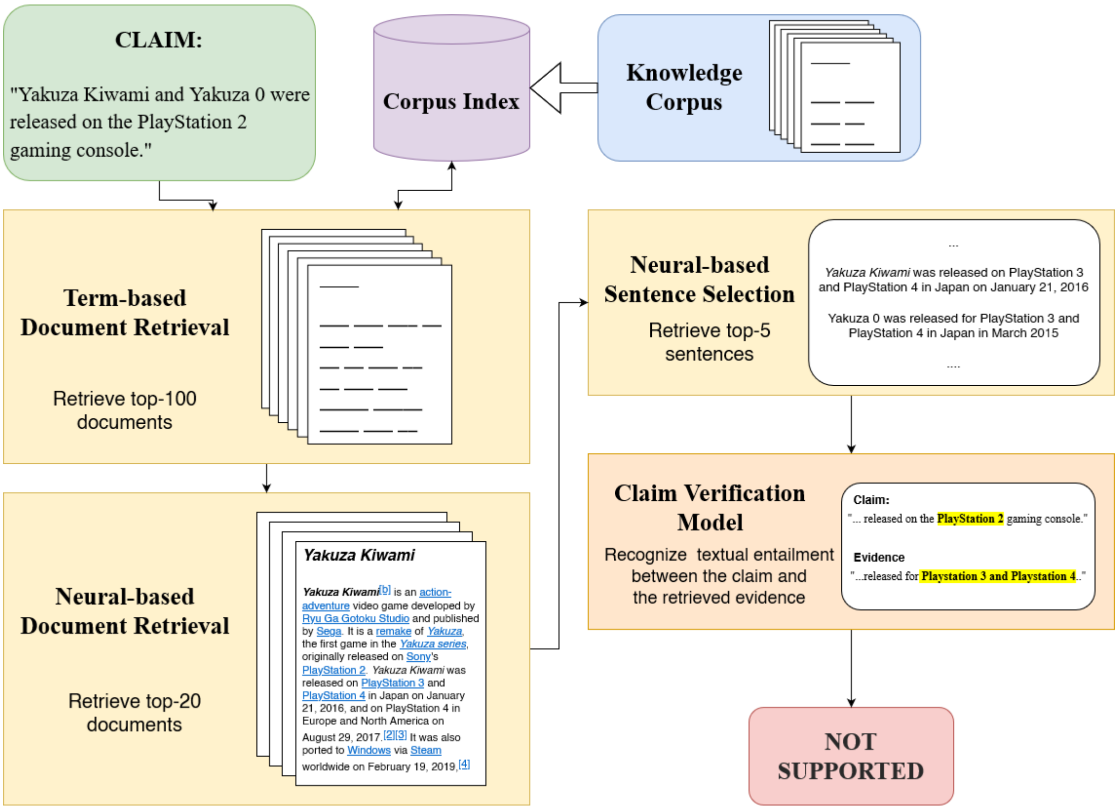

CLAIM:
"Yakuza Kiwami and Yakuza 0 were released on the PlayStation 2 gaming console."
Corpus Index
Knowledge Corpus
Term-based Document Retrieval
Retrieve top-100 documents
Neural-based Document Retrieval
Retrieve top-20 documents
Yakuza Kiwami
Yakuza Kiwami[b] is an action-adventure video game developed by Ryu Ga Gotoku Studio and published by Sega. It is a remake of Yakuza, the first game in the Yakuza series, originally released on Sony's PlayStation 2. Yakuza Kiwami was released on PlayStation 3 and PlayStation 4 in Japan on January 21, 2016, and on PlayStation 4 in Europe and North America on August 29, 2017.[2][3] It was also ported to Windows via Steam worldwide on February 19, 2019,[4]
Neural-based Sentence Selection
Retrieve top-5 sentences
...
Yakuza Kiwami was released on PlayStation 3 and PlayStation 4 in Japan on January 21, 2016
Yakuza 0 was released for PlayStation 3 and PlayStation 4 in Japan in March 2015
....
Claim Verification Model
Recognize textual entailment between the claim and the retrieved evidence
Claim:
"... released on the PlayStation 2 gaming console."
Evidence
"...released for Playstation 3 and Playstation 4.."
NOT SUPPORTED

Figure 4.2: Example of HoVer's fact-extraction and -verification pipeline with the 4-stage architecture

4. **Claim Verification Model:** A BERT-base model is fine-tuned for textual entailment between the claim c and the retrieved evidence Sn. The model takes the claim and evidence, separated by a [SEP] token, as input and conducts binary classification based on the output representation of the [CLS] token at the first position.

### 4.2.2 Overview Experiments

As the main baseline, we will take the original datasets before using any of the aforementioned adjustments (see Section 3.2) and run the HoVer pipeline on it. After having established the baselines on both datasets, for our experiments we aim to compare the three types of improvements for efficiency, as explained in Chapter 3. To provide a streamlined view, we will iteratively go through them and show how each iteration improves the pipeline.

**Retrieval Setups** We distinguish three types of retrieval methods in our experimental setup. The First consists of the Sparse retrieval which is part of the original baseline pipeline. For this, we will use the BM25 retrieval from Elasticsearch. For the second method, as mentioned in Section 3.2.2, the Term-based retrieval stages (Sparse retrieval and Neural-based) in the HoVer pipeline can be replaced by a single Dense Retrieval setup. For our specific use case, we will implement FAISS as our chosen dense retrieval method, aiming to retrieve the top 5 documents similar to the original pipeline. Lastly, there is also the Index compression setup aims to further improve the Dense Retrieval setup. Unlike the previous two improvements, this setup does not replace any component in the HoVer pipeline. Instead, it introduces a new module. Here instead of storing text in its original form in the index, as in previous setups, we utilize a document encoder segment to compress the index into a PQ index, significantly reducing its size compared to the Dense Retrieval Setup. During retrieval, the query encoder encodes the claim with the learned index, which is then utilized for retrieval purposes in the Dense Retrieval process. The retrieval process is expected to exhibit a speedup owing to the smaller size of vector embeddings in the index to search through. Moreover, in terms of performance, it is anticipated that there will be only marginal differences compared to the standard Dense Retrieval setup.

**Re-ranking data** Within each retrieval method, we will execute the same three distinct fact-extraction data settings. These settings, as explained in Chapter 3, involve the utilization of a pre-trained claim-detection model, the extraction of citations, and a fusion of both approaches. The goal of these methods is to bypass the Sentence Selection stage. This means there's no need to train and employ a separate model for identifying claim-worthy sentences in each candidate document, as we pre-compute and store them in the index. This is to essentially eliminate the need for on-the-fly computation, resulting in the amount of resources and time needed to be reduced for the fact-checking pipeline. It's important to highlight that although we still employ a BERT model for claim detection, it's only utilized during the curation of the data corpus, rather than during the inference time of the pipeline. This shift enhances the efficiency of our pipeline since the claim detection step is executed less frequently, resulting in overall improved performance.

**Sentence Retrieval Ablation** Expanding upon our experiments involving the three re-ranking data settings, we will also aim to gauge the impact of Dense Retrieval on the original corpus data across two distinct setups through an ablation study. Firstly, we'll assess its influence in a manner akin to the baseline pipeline configuration, which encompassed the Sentence Selection stage. Secondly, we'll examine its effects in a setup resembling the re-ranking approach, but without the Sentence Selection stage. This comparative investigation will elucidate the degree to which Sentence Selection contributes to the overall pipeline performance. Furthermore, this provides us with a valuable means to compare and contrast the outcomes against those derived from fact-extracted data settings. It also lets us see how much performance degradation these settings experience compared to the original data.

## 4.3 Experiment Setup

To ensure the reproducibility of our work, we provide a comprehensive overview of both our hardware setup and the implementation details of our code. The latter part involves which models are used in our experiments and some design decisions.

### 4.3.1 Hardware configuration

The experiments were run on a dedicated private server equipped with high-performance hardware. This to ensure robust and efficient processing of our experiments could be carried out, without facing problems due to hardware limitations. The server, powered by Arch Linux, consists of a 16-core 2nd Gen AMD EPYC™ 7302 processor coupled with two NVIDIA GeForce RTX 3090 GPUs, providing ample computational resources for our tasks. Complementing this powerful hardware configuration was a substantial 256 GB of RAM, enabling smooth and uninterrupted operation even during intensive computational tasks.

The HoVer pipeline boasts support for multi-GPU usage, yet we encountered some issues in that regard. Consequently, we executed the pipeline and all GPU-enabled models in a single GPU configuration to ensure seamless operation and consistency across the different experiments.

### 4.3.2 Implementation details

**Text Processing** For the re-ranking setup, the corpus files needed to be processed before being able to use them in our experiments. As mentioned before the Wikipedia dumps that we used are processed through the WikiExtractor tool. To speed up the process, we used JobLib to multiprocess the files in parallel. For the re-ranking setup, the claim-detection model used is a pre-trained BERT model [5] that has been finetuned on the ClaimBuster dataset. The model has scored an 82% accuracy score in correctly identifying claims and thus made for a proper model to use for our experiments. Overall processing the entire corpus for Claim detection takes around two to three days on our hardware setup. On the other hand processing for citation extraction takes around eight days to process. This is due to the need to retrieve the html webpages asynchronously as in a multi-processing setup we can miss some pages to be retrieved. Additionally, NLP models are used for sentence splitting. Using a single model instance would not work in multi-processing due to lack of multi-processing support and there not being any shared memory between the processes. This results in no retained text despite articles having citations, therefore the multi-threading option should be used instead. However in Python, as stated in their documentation[6], the 'Global Interpreter Lock' threading has been implemented as a concurrency instead of parallelism way. This has as a consequence that only one thread is to be run at a time instead of multiple ones concurrently. Even more in our case, we noticed it slowed the code even further compared to running it in a serial manner. Therefore to mitigate these issues we use the default multiprocessing and instantiate the NLP model in each process rather than share the model between the processes. Although this adds some overhead in latency and memory for each process, in the end, it significantly speeds up our text processing for citation extraction on the data collection. Finally, it's important to highlight that the initial setup of the HoVer pipeline utilized only the lead section of the Wikipedia corpus in its downloaded database file. However, since Wikipedia's lead sections don't include citations and serve as self-contained summaries of the entire articles, we decided to utilize the full article text for our experiments. Despite this adaptation, we've retained the option in our code to choose whether to use the full text or just the lead section.

**NLP models** For extracting supporting facts in our text processing pipeline, we leveraged several state-of-the-art models to facilitate various tasks. The HoVer pipeline, by default, incorporates StanfordCoreNLP[7] for sentence splitting. However, StanfordCoreNLP requires the Java Runtime Environment to be installed, which may not always be convenient for Python-centric workflows. Additionally, the setup of StanfordCoreNLP acts like a server. This works well in

[5] https://huggingface.co/Nithiwat/bert-base_claimbuster

[6] https://docs.python.org/3/library/threading.html

[7] https://stanfordnlp.github.io/CoreNLP/

the pipeline as it works on smaller amounts of text. Unfortunately, for processing the entire Wikipedia corpus for extracting supporting facts, we noticed it suffered from disconnections as the response was not being received at the server endpoint. To address this, we have integrated instead the *'en_core_web_lg'*[8] model from the SpaCy library as an alternative. This choice offers several advantages, including SpaCy's renowned efficiency and speed, often surpassing that of StanfordCoreNLP. This ensures quicker processing times and optimized resource utilization, which is particularly beneficial for real-time applications. Moreover, Spacy enables the creation of custom NLP pipelines, allowing us to tailor the functionality to our specific needs. For instance, if only sentence splitting is required, we can configure Spacy to load only the Sentencizer component, thereby avoiding unnecessary overhead from other components.

**Sparse Retrieval** The current implementation of the HoVer pipeline includes the utilization of retrieved article files from the Term-based Document Retrieval stage, although it lacks specific setup details. It does reference the use of the DrQA library[9] as an option for processing HoVer data, which could enable a more fair comparison. However, for our specific use case, we plan to replace the TF-IDF Document Retrieval step using BM25 from Elasticsearch[10]. This decision is primarily motivated by the fact that our system already has an Elasticsearch instance set up, making integration with BM25 retrieval more straightforward compared to incorporating DrQA. Additionally, we anticipate that the Sparse Retrieval between the two methods should yield comparable results.

**Dense Retrieval** As mentioned before, for our Dense Retrieval we will employ the FAISS[11] library with GPU support. Here we used the FAISS flat index data structure for a fast similarity search of dense vectors, using dot product against the query as a scoring function. This particular index data structure stores all vectors directly in a single structure, making it simple and efficient for exact searches. However, it may not scale as well to very large datasets compared to more complex index structures of the FAISS library (e.g. IVF). For generating text embeddings, we employed the pre-trained *'all-MiniLM-L6-v2'* model from Sentence Transformers[12], a versatile and effective solution for creating dense representations of textual data. This model has only been used for the Dense Retrieval setup without index compression, as the index compression used the JPQ[13] model to encode the textual data.

**Index Compression** As detailed in the JPQ paper [45], variable $K$ (the number of codewords) is set to 256 to store embedding in one byte, and $M$ (number of codebooks/sub-vectors) can be set to any number of sub-vectors smaller than the embedding size. For our use case, we only ran it for M = 96, to see how much reduction can be achieved in index size, while still getting as high performance as possible from using a relatively high number of sub-vectors to represent the data. The dual-encoders are based on the Roberta model from Huggingface Transformers. The document encoder uses OPQ [11] to learn a linear transformation of embeddings and PQ [18] for compression. Furthermore, for training, the AdamW optimizer has been used with a batch

[8] https://spacy.io/models/en

[9] https://github.com/facebookresearch/DrQA

[10] https://elasticsearch-py.readthedocs.io/en/v8.12.1/

[11] https://github.com/facebookresearch/faiss

[12] https://www.sbert.net/docs/pretrained_models.html

[13] https://github.com/jingtaozhan/JPQ

size of 32, LambdaRank [7] as a pair-wise loss function, and the top-200 irrelevant documents as hard negatives. For the query encoder, the learning rate is set to *5* × *10-6* and the PQ learning rate is set to *1* × *10-4* for our setting (depends on *M*). Creating the index for the original Wikipedia dumps (without re-ranking applied) for our experiments took roughly between *10-12* hours on our setup. Subsequently, the reduced corpus size through re-ranking took *6-7* hours, roughly half the time.

**Fact-checking pipeline** In our custom scripts, we predominantly utilize Python version 3.10.9 due to its compatibility with various existing tools and libraries essential for our experimental work, such as retrieval libraries and the wiki extractor. However, we made an exception for the HoVer codebase, which contains legacy code relying on older dependencies. For this particular codebase, we opted for Python version 3.7.16 to maintain compatibility and avoid disrupting existing functionality. Updating to a newer Python version would have necessitated not only updating dependencies, but also extensive re-implementation of core components, which is beyond the scope of our study. The models employed in different stages of the HoVer fact-checking pipeline are based on pre-trained BERT-base uncased models, with variations arising from their fine-tuning for specific pipeline stages. Fine-tuning involves using a batch size of 16 and a default learning rate of 3e-5 without warmup. The number of training epochs is set to 3 for the Sentence Selection stage and 5 for the remaining stages. For model parameters, kr = 20, kp = 5, p = 0.5, and s = 0.3 are used depending on the memory limitations and performance on the development claim dataset.

## 4.4 Assessment and Metrics

**Performance Evaluation:** In alignment with the original HoVer methodology, we will utilize accuracy as the primary metric to assess the correct identification of claims as either *'Supporting'* or *'Not Supporting'*. However, to provide a comprehensive evaluation, we will supplement accuracy with additional metrics including *F1 score, recall*, and *precision*. Given the uneven distribution of entailment labels in the WiCE dataset, both macro and weighted metrics will be reported for a more nuanced understanding (Table 4.1). Since the HoVer claim data is well-balanced and much larger, the various metrics do not vary by much compared to the WiCE claim data where accuracy is not particularly indicative of how well the experiments score. This as the accuracy can be quite misleading with incorrectly labelling every claim as 'not supported' would already yield a 66% accuracy. Therefore by using the combination of both *weighted and macro* scores, these metrics allow us to take into account class imbalances of the WiCE data, while also ensuring equal considerations between the two labels. As end results, we select and report the model checkpoints with the best predictions. The selection process will prioritize sorting based on taking the five best F1-weighted scores, followed by selecting the one with the highest F1-macro score. This approach ensures the choice of the model checkpoint is less biased towards specific labels. For example, it may occur that a checkpoint scores highly in accuracy and f1-weighted, but perhaps very poorly in f1-macro. To mitigate the risk of selecting such a checkpoint, this selection approach minimizes the chances of under- and over-prediction for any particular class label, causing checkpoints to be selected that have an overall well-balanced score across the different metrics.

**Efficiency Evaluation** The evaluation of retrieval latency will encompass both CPU and GPU implementations of FAISS (with and without Index Compression). As BM25 lacks GPU support, latency measurements will be limited to CPU performance. In addition to the above, the following metrics will be recorded for each retrieval setting: index size on disk, time taken for index creation, and time for document retrieval in both BM25 and FAISS setups. Furthermore, secondary measurements will include total runtime, corpus data size on disk, disk writes for constructing train and development sets at each stage, and disk writes for model checkpoints and predictions. These are measured using Python's in-built 'psutil'[14] tool. Lastly, maximum CPU usage, maximum memory consumption, and GPU utility and memory usage will be monitored throughout the experiments using the 'nvidia-ml-py3' library[15], which is a Python wrapper around the NVML library[16]. Important for retrieval and inference operations is to mimic a realistic setting, where each search operation should act as if it would be operating in real-time practical applications. To ensure this realism in performance estimation, although not fully simulated, the sequential retrieval latency on the dev dataset will be measured (essentially batch size of 1). Since the training dataset is not used in the inference of the various stages and also is much larger, it will undergo batched measurement with a consistent batch size of 128 to process it in a reasonable time. For the inference latency comparisons, the time to perform the average inference on a sample in milliseconds will not only encompass the inference time, but also the preparation time of the data for each stage. It should be noted that the time to load the model for each inference step will be excluded from the latency comparisons.

[14] https://psutil.readthedocs.io/en/latest/

[15] https://github.com/nicolargo/nvidia-ml-py3

[16] https://developer.nvidia.com/nvidia-management-library-nvml

# Chapter 5

# Results and Discussion

In this chapter, we present the results of our experiments. The main goal here is the analysis of how indexing supporting facts can enhance the efficiency of a fact-checking pipeline. Our analysis is organized around the research questions posed in Chapter 3, providing a structured approach for deriving meaningful insights. It should be noted that only the key observations will be analysed in this chapter. So only a subset of the results will be discussed here, namely, the performance measured in accuracy (percentage) and efficiency measured in latency (milliseconds). For the full tables containing all the measurements, refer to Appendix A for the difference experiment settings on HoVer data and similarly Appendix B on WiCE data. Finally, we also present an alternative perspective of the tabular results data through graph visualizations in Appendix C, aiming to enhance comprehension of the impact of our findings.

## 5.1 RQ 1: How does indexing supporting facts improve information retrieval efficiency?

In this section, we investigate the impact of indexing supporting facts on information retrieval efficiency by comparing the disk space utilization and retrieval latency across different experiment settings. Here we aim to discern the benefits of storing only supporting facts in the index as opposed to the entire corpus.

### 5.1.1 Corpus Size

To get an idea of how storing just the supporting facts data in the index improves efficiency compared to storing the entire corpus, a comparison can be made on how much these different settings occupy disk space. As mentioned in Section 4.4, to get an accurate estimate, only the dictionaries containing the article's title and document text are saved to raw JSON files. Across all experiment settings as seen in Table 5.1, a notable reduction in disk space usage is observed compared to the original Wikipedia document corpus. This reduction ranges from approximately 45% (claim detection) to 55% depending on the setting for the HoVer corpus data. Likewise, for the WiCE corpus data, we can observe approximately 44% to 57% reduction. Moreover, in correlation with the reduced disk size, it is evident that the number of sentences stored in the index also decreases across each experiment setting compared to the original corpus data. For HoVer this ranges from 52% (claim detection) to 61% (citation extraction) and WiCE ranges

| Experiment setting | Disk Size | Size reduction | #Sentences | Sentences reduction |
|---|---|---|---|---|
| *HoVer* | | | | |
| Original | 11.28 GiB | - | 94,914,378 | - |
| Claim detection | 6.19 GiB | 45% | 45,894,704 | 52% |
| Citation Extraction | 5.07 GiB | 55% | 36,886,889 | 61% |
| Fusion | 5.45 GiB | 52% | 39,842,574 | 58 % |
| *WiCE* | | | | |
| Original | 15.28 GiB | - | 126,533,841 | - |
| Claim detection | 8.56 GiB | 44% | 61,040,380 | 52% |
| Citation Extraction | 6.56 GiB | 57% | 51,735,961 | 59% |
| Fusion | 6.85 GiB | 55% | 54,070,295 | 57% |

Table 5.1: Comparison sizes for the corpora per experiment setting, consisting of English Wikipedia articles 2017 (HoVer) and 2024 (WiCE). Reduction is measured with respect to the original data setting.

from 52% to 59%. This indicates that at least half of the sentences are considered as not claim-worthy across the different re-ranking methods.

### 5.1.2 Retrieval Latency

**Sparse retrieval** Following the reduction in disk size, a notable enhancement in retrieval latency is evident, as demonstrated in both the Term-based and Neural-based document retrieval columns of Table 5.2. To avoid any ambiguity, it's crucial to clarify that the speedup listed in the table pertains to the total latency, which is relevant for addressing RQ2, rather than solely focusing on document retrieval. Regarding document retrieval latency (which encompasses both column values), there's an observed speedup ranging from approximately 1.5x (334 ms) to 1.6x (316 ms) compared to the original experimental setting for HoVer (495 ms). Similarly, in WiCE experiments, we witness a comparable speedup rate ranging from 1.4x (446 ms) to 1.6x (399 ms) compared to the original experimental setting (636 ms). This observation suggests that while the reduced text size contributes to expedited retrieval, the enhancement is only somewhat proportional.

**CPU-based Dense Retrieval** One might typically anticipate a more pronounced disparity between the original data and the reranked data in the document retrieval phase. However when transitioning from the Sparse retrieval setup to the Dense retrieval setup, as depicted in the first column of Table 5.3, only negligible differences between the different settings are observed. This is attributed to FAISS utilizing vectors instead of computing the relevance ranking of documents to the query, as is the case with BM25. Despite variations in the length of each article across settings, the number of text embeddings (with fixed dimensionality size) created remains constant, corresponding to the number of encoded text spans, which is consistent across settings. Thus minimizing the impact of extracting supporting facts on document retrieval latency when using Dense Retrieval. Comparing the Dense document retrieval (CPU) column in Table 5.3 to the baselines listed in Table 5.2, it is observed to be of a similar latency or even slightly slower. For HoVer, we can observe a 0.9x (523 ms) to 1.0x (479 ms) compared to the baseline (495 ms). Likewise, for WiCE we can observe a similar latency speedup of 0.9x (685 ms) to 1.0x (610 ms)

| Sparse Retrieval with Re-ranking setup | Term-based document retrieval | | Neural-based document retrieval | Sentence Retrieval | Claim Verification | Total Latency | Speedup |
|---|---|---|---|---|---|---|---|
| | CPU | GPU | | | | | |
| *HoVer* | | | | | | | |
| **Original (baseline)** | **426 ms** | - | **69 ms** | **157 ms** | **7 ms** | **659 ms** | - |
| Claim detection | 257 ms | - | 71 ms | - | 10 ms | 338 ms | 1.9x |
| Citation Extraction | 246 ms | - | 70 ms | - | 11 ms | 327 ms | 2.0x |
| Fusion | 265 ms | - | 69 ms | - | 11 ms | 345 ms | 1.9x |
| *WiCE* | | | | | | | |
| **Original (baseline)** | **559 ms** | - | **77 ms** | **186 ms** | **9 ms** | **831 ms** | - |
| Claim detection | 372 ms | - | 74 ms | - | 22 ms | 468 ms | 1.8x |
| Citation Extraction | 330 ms | - | 69 ms | - | 20 ms | 419 ms | 2.0x |
| Fusion | 347 ms | - | 69 ms | - | 20 ms | 436 ms | 1.9x |

Table 5.2: Retrieval and inference latency for Sparse retrieval with Re-ranking setup on data settings. Speedup is compared with respect to the total latency of the original data setting (bold font).

speedup compared to its baseline (636 ms). This suggests that the indexing of supporting facts would not significantly impact information retrieval efficiency in such scenarios.

| Dense Retrieval setup | Term-based document retrieval | | Sentence Retrieval | Claim Verification | Total Latency | | Speedup | |
|---|---|---|---|---|---|---|---|---|
| | CPU | GPU | | | CPU | GPU | CPU | GPU |
| *HoVer* | | | | | | | | |
| **Original (baseline)** | **491 ms** | | **157 ms** | **7 ms** | **659 ms** | | - | - |
| Original (+ Sent. Select.) | 515 ms | 31 ms | 153 ms | 8 ms | 676 ms | 192 ms | 1.0x | 3.4x |
| Original | 523 ms | 30 ms | - | 9 ms | 532 ms | 39 ms | 1.2x | 16.9x |
| Claim detection | 513 ms | 23 ms | - | 8 ms | 521 ms | 31 ms | 1.3x | 21.3x |
| Citation Extraction | 479 ms | 23 ms | - | 9 ms | 488 ms | 32 ms | 1.4x | 20.6x |
| Fusion | 500 ms | 23 ms | - | 9 ms | 509 ms | 32 ms | 1.3x | 20.6x |
| *WiCE* | | | | | | | | |
| **Original (baseline)** | **636 ms** | | **186 ms** | **9 ms** | **831 ms** | | - | - |
| Original (+ Sent. Select.) | 685 ms | 34 ms | 184 ms | 9 ms | 878 ms | 227 ms | 1.0x | 3.7x |
| Original | 610 ms | 34 ms | - | 9 ms | 619 ms | 43 ms | 1.3x | 19.3x |
| Claim detection | 622 ms | 31 ms | - | 9 ms | 631 ms | 40 ms | 1.3x | 20.8x |
| Citation Extraction | 610 ms | 31 ms | - | 9 ms | 619 ms | 40 ms | 1.3x | 20.8x |
| Fusion | 619 ms | 31 ms | - | 9 ms | 628 ms | 40 ms | 1.3x | 20.8x |

Table 5.3: Retrieval and inference latency for Dense retrieval setup on data settings. Speedup is compared with respect to the total latency of the Sparse Retrieval setup with original data setting (bold font).

**GPU-based Dense Retrieval** However, it is worth noting that Dense retrieval can still be faster, particularly with dense retrieval libraries such as FAISS offering GPU support, which can yield substantial speedups compared to both CPU retrieval of BM25 and FAISS. This advantage is evident in the data, showcasing notable speedups ranging from 16.6x to 22.3x speedup for HoVer GPU retrieval over CPU retrieval, and 17.9x to 20.2x speedup for WiCE. Furthermore, when comparing FAISS GPU retrieval to the BM25 retrieval, we can see an approximate 16.0x (31 ms) to 21.5x (23 ms) speedup for HoVer and 18.7x (34 ms) to 20.5x (31 ms) speedup for WiCE. Therefore the GPU-based approach makes Dense Retrieval a viable option, unlike the CPU-based variant.

### 5.1.3 Key Takeaways

Extracting supporting facts from the data corpus can lead to only requiring to store at least half of the data. Although this has a positive effect on the latency for Sparse retrieval, with Dense document retrieval this is not the case due to how the vector embeddings are constructed (being per article rather than per sentence). Furthermore, while CPU-based Dense retrieval may not necessarily outperform Sparse retrieval methods in terms of latency, thereby presenting less immediate appeal, the incorporation of GPU support leads to significant speed enhancements. Thus, although extracting supporting facts does not help much in Dense document retrieval unlike Sparse retrieval in terms of retrieval latency, the incorporation of the GPU-based Dense retrieval renders it a much more compelling option for achieving efficiency.

## 5.2 RQ 2: How does indexing supporting facts affect overall pipeline efficiency and downstream fact-checking performance?

In continuation of the previous research inquiry concerning retrieval latency and disk size, this section delves into an analysis of the overall inference time across the entire pipeline. Additionally, recognizing that faster processing times do not necessarily equate to better performance a further analysis will be done on the performance metrics.

### 5.2.1 Inference Latency

**Sparse Retrieval Setup:** The enhancement in retrieval latency, as evidenced in Table 5.2, mirrors a noticeable improvement in the overall inference latency across the pipeline. This improvement spans approximately 1.9x to 2.0x for the HoVer experiments and 1.8x to 2.0x for WiCE experiments. However, the reduction in total latency cannot be solely ascribed to faster retrieval times. It also arises from the elimination of the Sentence Retrieval stage, which previously imposed significant latency overhead. Upon closer inspection of Table 5.2, it becomes apparent that the absence of the Sentence Retrieval stage impacts the Claim Verification stage. Notably, experiments conducted on the original corpus data exhibit much lower inference latency compared to supporting facts data. Nevertheless, the variance between these experiment settings is minimal, and the impact on total latency results is insignificant. This overall trend indicates that indexing supporting facts for the BM25 retrieval setup predominantly benefits inference times for the Rule-based document retrieval and Sentence Retrieval stages. Furthermore, it reveals that the Claim Verification stage is slightly, yet negligibly, affected when considering the entire pipeline inference.

**Dense Retrieval Setup:** In a similar vein as the document retrieval comparisons of RQ1 (see Section 5.1.2), the total inference of the Dense retrieval setup presents notable differences in results between CPU- and GPU-based Dense retrieval compared to Sparse retrieval. This divergence is evident in Table 5.3, where for HoVer experiments, the CPU-based approach exhibits a 1.2x to 1.4x speedup, while the GPU-based approach demonstrates a 16.9x to 21.3x speedup compared to the baseline. Similarly, WiCE experiments show approximately a 1.3x speedup for the CPU-based approach and 19.3x to 20.8x speedup for the GPU-based approach. The key distinction lies in the influence of omitting the Sentence Retrieval stage for the original corpus data. Its omission introduces significant overhead to the total latency. For the CPU-based approach, this translates to a 1.3x speedup for HoVer (676 ms vs. 532 ms) and a 1.4x speedup for WiCE (878 ms vs. 619 ms). Conversely, the GPU-based approach experiences a 4.9x speedup for HoVer (192 ms vs. 39 ms) and a 5.3x speedup for WiCE (227 ms vs. 43 ms). Overall, this underscores that including Sentence Retrieval adds substantial overhead, especially for GPU-based approaches operating with lower latency magnitudes. Therefore, the supporting facts data for Dense Retrieval, while not significantly impacting document retrieval, offers significant speedup for total inference latency, allowing for the effective omission of the Sentence Retrieval stage and its associated latency overhead.

### 5.2.2 Performance Metrics Evaluation

| Experiment setting | Accuracy | F1 | | Precision | | Recall | |
|---|---|---|---|---|---|---|---|
| | | Weighted | Macro | Weighted | Macro | Weighted | Macro |
| *Sparse retrieval with Re-ranking* | | | | | | | |
| Original | **67.79** | **67.59** | **67.63** | **68.45** | **68.39** | **67.79** | **67.93** |
| Claim detection | 62.33 | 62.02 | 62.08 | 62.98 | 62.92 | 62.33 | 62.50 |
| Citation Extraction | 60.91 | 60.61 | 60.66 | 61.47 | 61.42 | 60.91 | 61.07 |
| Fusion | 62.28 | 62.15 | 62.18 | 62.60 | 62.56 | 62.28 | 62.39 |
| *Dense Retrieval* | | | | | | | |
| Original (with Sent. Select.) | 64.60 | 64.45 | 64.45 | 64.86 | 64.86 | 64.60 | 64.60 |
| Original | 62.90 | 62.72 | 62.76 | 63.33 | 63.28 | 62.90 | 63.02 |
| Claim detection | 61.50 | 60.94 | 60.94 | 62.20 | 62.20 | 61.50 | 61.50 |
| Citation Extraction | 59.67 | 59.40 | 59.46 | 60.13 | 60.09 | 59.67 | 59.82 |
| Fusion | 59.51 | 59.32 | 59.37 | 59.85 | 59.81 | 59.51 | 59.64 |
| *Index Compression* | | | | | | | |
| Original (with Sent. Select.) | 63.30 | 62.54 | 62.54 | 64.48 | 64.48 | 63.30 | 63.30 |
| Original | 63.02 | 62.08 | 62.08 | 64.46 | 64.46 | 63.02 | 63.02 |
| Claim detection | 61.92 | 61.71 | 61.71 | 62.19 | 62.19 | 61.92 | 61.93 |
| Citation Extraction | 59.98 | 59.12 | 59.12 | 60.89 | 60.89 | 59.98 | 59.98 |
| Fusion | 61.58 | 61.43 | 61.43 | 61.75 | 61.75 | 61.58 | 61.58 |

Table 5.4: Performance experiments on HoVer data and adjustments using full document text of English Wikipedia. The bold-styled values represent the baseline while the underlined-styled values represent the highest scores of the re-ranked data within a retrieval setup category.

**Sparse Retrieval performance** Utilising the metrics laid out in Section 4.4, the pipeline results have been evaluated for the different settings and laid out in Table 5.4 for the HoVer experiments and Table 5.5 for WiCE experiments. When comparing the different HoVer experiment settings within the Sparse Retrieval setup, Claim detection comes the closest to the baseline with

| Experiment setting | Accuracy | F1 | | Precision | | Recall | |
|---|---|---|---|---|---|---|---|
| | | Weighted | Macro | Weighted | Macro | Weighted | Macro |
| *Sparse retrieval with Re-ranking* | | | | | | | |
| Original | **63.69** | **61.84** | **55.24** | **61.12** | **56.54** | **63.69** | **55.32** |
| Claim detection | 61.90 | 60.12 | 53.33 | 59.27 | 54.26 | 61.90 | 53.53 |
| Citation Extraction | 61.01 | 59.56 | 52.96 | 58.75 | 53.59 | 61.01 | 53.09 |
| Fusion | 63.39 | 60.21 | 52.48 | 59.46 | 54.69 | 63.39 | 53.27 |
| *Dense Retrieval* | | | | | | | |
| Original (with Sent. Select.) | 61.61 | 60.95 | 55.21 | 60.47 | 55.49 | 61.61 | 55.13 |
| Original | 60.42 | 58.80 | 51.96 | 57.90 | 52.58 | 60.42 | 52.19 |
| Claim detection | 61.01 | 58.94 | 51.78 | 57.96 | 52.70 | 61.01 | 52.17 |
| Citation Extraction | 58.63 | 58.48 | 52.92 | 58.35 | 52.95 | 58.63 | 52.91 |
| Fusion | 61.31 | 59.34 | 52.30 | 58.40 | 53.23 | 61.31 | 52.62 |
| *Index Compression* | | | | | | | |
| Original (with Sent. Select.) | 62.46 | 61.38 | 55.27 | 60.74 | 55.84 | 62.46 | 55.20 |
| Original | 60.46 | 60.63 | 55.64 | 60.81 | 55.60 | 60.46 | 55.70 |
| Claim detection | 59.31 | 59.02 | 53.32 | 58.77 | 53.39 | 59.31 | 53.30 |
| Citation Extraction | 60.74 | 59.21 | 52.42 | 58.34 | 53.04 | 60.74 | 52.60 |
| Fusion | 63.04 | 59.79 | 51.89 | 58.94 | 53.97 | 63.04 | 52.76 |

Table 5.5: Performance experiments on WiCE data and adjustments using full document text of English Wikipedia. The bold-styled values represent the baseline while the underlined-styled values represent the highest scores of the re-ranked data within a retrieval setup category.

close to 5.5 points difference across the metrics for the HoVer experiments. Important to note is that Fusion follows close with less than a point difference. For the WiCE Sparse retrieval setup, the opposite occurred with the Fusion data being the closest with a marginal 0.3 point difference followed by Claim detection with a 1.5 points difference. In both datasets, the Citation extraction takes the biggest loss in accuracy that being 6.9 points for HoVer and 2.7 points difference for WiCE. We can reason the fact that citation extraction takes the biggest performance degradation to the fact that not all claim-worthy sentences contain citations, therefore missing out on crucial evidence sentences. Unlike the other settings which consider the complete text instead of only the cited sentences and determine claim-worthiness on what the claim-detection model selects. Overall, relating to the inference time, we can see that for HoVer with a speedup of at least 1.5x to 1.6x, we only lose 6.9 to 5.5 points in performance across various metrics for the best re-ranking setup. Likewise, for WiCE, with a speedup of 1.4x to 1.6x we only lose 2.7 to 0.3 points. This positively demonstrates that indexing just the supporting facts does show meaningful results in terms of overall pipeline efficiency, while maintaining roughly the same performance. Additionally, this also indicates we can achieve good results by using a combination of citation extraction together with another supporting facts extraction method such as Claim detection.

**Dense Retrieval performance** When examining the performance results of Dense retrieval compared to Sparse Retrieval, it becomes evident that there is a slight decline across all experiments. For HoVer, this decline ranges from a modest 0.8 point difference in Claim detection to a more substantial 2.9 points in Fusion data. Similarly, WiCE experiences a loss ranging from a 0.9 difference in accuracy between Claim detection settings to approximately 2.4 points in Citation extraction. Crucially, it is to assess how these performances compare against the baselines. In HoVer, the accuracy loss ranges from 8.3 points for Fusion data to 6.3 points for Claim detection. WiCE experiences a loss ranging from 5.1 points in Citation extraction to 2.4 points in Fusion. These findings suggest that while transitioning from Sparse retrieval to just a Dense retrieval component incurs some loss, it's not substantial across various experiments involving supporting facts data. Moreover, the performance is notably strong in claim detection, while citation extraction lags behind by only a few points. Interestingly, while Fusion performs as well as Citation extraction in HoVer experiments, Fusion data outperforms Claim detection in WiCE. This highlights the significance of combining citation extraction with another supporting facts extraction method to achieve optimal results, similar to the Sparse retrieval setup.

**Sentence Retrieval stage ablation** Comparing experiments on the original data between the two retrieval methods reveals a more significant decline for HoVer, with a loss of 3.2 points with Sentence Selection and 4.9 points without it. For WiCE, the difference is 2.1 points with Sentence Selection and 3.3 points without it. When assessing these losses against the baselines, it becomes evident that both methods generally outperform the supporting facts data experiments by a few points. This suggests that the contribution of the Sentence Retrieval stage in the pipeline to performance improvement is marginal. With the supporting facts extraction thus becomes quite effective in achieving nearly the same performance. Consequently, to enhance efficiency, eliminating this Sentence Retrieval stage would result in only a loss of less than a few points.

### 5.2.3 Key Takeaways

Incorporating supporting facts into both Sparse and Dense retrieval setups yields noteworthy enhancements in overall pipeline efficiency. Sparse retrieval setups demonstrate speedups ranging from up to around 1.5x, while Dense retrieval setups exhibit even more substantial improvements, achieving up to approximately 20.0x with GPU-based approaches. These notable speedups are primarily attributed to the removal of the Sentence Retrieval stage, which incurs considerable latency overhead. Further evaluation indicates a minor decline in performance when transitioning from Sparse to Dense retrieval, though the loss is not substantial. Specifically, claim detection remains robust, while citation extraction may lag behind by a few points. However, Fusion data yields promising results, often comparable to or outperforming other extraction methods, emphasizing the significance of amalgamating various extraction techniques for supporting facts. Moreover, ablation experiments on the Sentence Retrieval stage reveal its marginal contribution to performance improvement. Comparisons between original data and supporting facts data show only a slight decline in performance, showcasing that utilising only the supporting facts only incurs a modest loss in performance (around 6 points for HoVer and 3 points for WiCE). This suggests that although supporting facts do not affect document retrieval latency in the Dense Retrieval setup, it does help with overall pipeline latency due to avoiding the latency overhead of Sentence Selection. In conclusion, these results underscore the meaningful impact of indexing supporting facts on the overall pipeline efficiency, with only minimal losses in downstream fact-checking performance.

## 5.3 RQ 3: In what ways does index compression enhance the efficiency of dense retrieval and fact-checking systems?

In this final research inquiry concerning the addition of index compression, this section explores how index compression improves upon Dense Retrieval in not only the constructed index size, but also document retrieval and total inference latency. Additionally, a final comparison will be made on the overall performance against Sparse retrieval and standard Dense Retrieval.

### 5.3.1 Compressed Index Size

In our FAISS experiments, we consistently observe an index size of approximately 7.51 GiB across all HoVer settings and 9.70 GiB across all WiCE settings. While one might anticipate that re-ranking would influence the amount of text utilized for generating vector embeddings, it's crucial to note that the index size remains unchanged. This is due to the fact that we generate vector embeddings on a per-article basis with only the text itself being altered. To address this issue, we employed JPQ, an index compression model. Despite using a relatively high number of subvectors for the JPQ model (M=96), we observed a significant reduction in the total index size. Specifically, the individual vector embeddings now occupy only 104.12 B in storage space, down from 1.5 KiB previously. This reduction is remarkable. For the HoVer experiments, the index size decreased from 7.51 GiB to 544.89 MiB, and for the WiCE experiments, we observed a decrease from 9.70 GiB to 672.95 MiB. Overall, this constitutes an impressive reduction of nearly 93% or a compression ratio of 14.4:1 in index size for both experiment setups. It's worth noting that employing fewer sub-vectors could potentially lead to an even more substantial reduction in index size; however, this would come at the cost of decreased performance.

| Index Compression setup | Term-based document retrieval | | Sentence Retrieval | Claim Verification | Total Latency | | Speedup | |
|---|---|---|---|---|---|---|---|---|
| | CPU | GPU | | | CPU | GPU | CPU | GPU |
| *HoVer* | | | | | | | | |
| **Original (baseline)** | **491 ms** | | **157 ms** | **7 ms** | **659 ms** | | - | - |
| Original (+ Sent. Select.) | 53 ms | 13 ms | 153 ms | 8 ms | 214 ms | 174 ms | 3.1x | 3.8x |
| Original | 55 ms | 13 ms | - | 12 ms | 67 ms | 25 ms | 9.8x | 26.4x |
| Claim detection | 51 ms | 12 ms | - | 9 ms | 60 ms | 21 ms | 11.0x | 31.4x |
| Citation Extraction | 46 ms | 11 ms | - | 9 ms | 51 ms | 20 ms | 12.9x | 33.0x |
| Fusion | 51 ms | 12 ms | - | 12 ms | 63 ms | 24 ms | 10.5x | 27.5x |
| *WiCE* | | | | | | | | |
| **Original (baseline)** | **636 ms** | | **186 ms** | **9 ms** | **831 ms** | | - | - |
| Original (+ Sent. Select.) | 97 ms | 43 ms | 186 ms | 9 ms | 292 ms | 238 ms | 2.8x | 3.5x |
| Original | 95 ms | 43 ms | - | 11 ms | 106 ms | 54 ms | 7.8x | 15.4x |
| Claim detection | 92 ms | 37 ms | - | 11 ms | 103 ms | 48 ms | 8.1x | 17.3x |
| Citation Extraction | 89 ms | 37 ms | - | 9 ms | 98 ms | 46 ms | 8.5x | 18.1x |
| Fusion | 89 ms | 37 ms | - | 9 ms | 98 ms | 46 ms | 8.5x | 18.1x |

Table 5.6: Retrieval and inference latency for Index compression setup. Speedup is compared with respect to the total latency of the Sparse Retrieval setup with original data setting (bold font).

### 5.3.2 Pipeline Efficiency

**Document Retrieval Latency** When examining the retrieval latency outlined in Table 5.6, a notable observation can be made towards the Dense document retrieval compared to the Dense Retrieval results outlined in Table 5.3. This significant enhancement can be primarily attributed to the utilization of the index compression model, which effectively reduces the index size. As a result, retrieval latency experiences a considerable improvement due to the smaller vector embeddings, facilitating faster similarity computation. Here one can observe a substantial speedup achieved in CPU retrieval of approximately 10.0x across the HoVer experiment settings and 7.0x for WiCE experiments. Similarly, GPU retrieval exhibits a speedup of approximately 2.0x for HoVer experiments and 0.8x for WiCE experiments. This is generally in line with the reported results in the original JPQ paper [45]. Although the measurements for HoVer fall in line with these reported results, one may notice that the WiCE retrieval speedup is lower than that of HoVer. This is even worse for the GPU-based retrieval latency instead of being better than the standard GPU-based Dense retrieval. We reason this to the fact that the WiCE claim dataset is a lot more complex. In the original WiCE paper, the results that were reported already indicate a not so particularly high performance being achieved. This coupled with the use of a different model for creating the embeddings results in marginally worse performance instead of a speedup such as the case with HoVer.

**Pipeline Inference Latency** In examining the total inference latency, as further detailed in Table 5.6, the utilization of compressed indexing and the ensuing document retrieval speed enhancements result in a notable boost across the board. The advancements brought about by JPQ, which further build upon the foundations of Dense Retrieval, are particularly significant. Notably, CPU latency has seen a substantial improvement compared to previous benchmarks on the supporting fact data, exhibiting a noteworthy speedup ranging from 10.5x to 12.9x for HoVer experiments, and 8.1x to 8.5x for WiCE experiments relative to their respective baselines. Meanwhile, the GPU-based approach, especially in the case of HoVer experiments, has yielded even more impressive results, achieving speedups ranging from 27.5x to 33.0x. While WiCE experiments on the GPU may not experience such dramatic speedups, they still showcase marked enhancements over their original baselines that range from 17.3x to 18.1x speedups. When assessing the impact of the Sentence Selection stage on the original corpus data settings, the findings reinforce the observations made in the standard Dense Retrieval setup. Furthermore, the disparity in the reported speedups between the tables underscores the significance of incorporating index compression.

### 5.3.3 Performance Metrics Evaluation

When comparing the performance of JPQ in the HoVer experiments (as shown in Table 5.4) as well as the performance of the WiCE experiments (presented in Table 5.5), a notable trend emerges. The index compression brought by JPQ generally yields higher scores compared to the standard Dense retrieval experiments. This improvement is particularly striking as the gap between the JPQ experiments and the baseline performances is further narrowed. In the HoVer experiments, this enhancement ranges from marginal increases of less than a point in Claim detection and Citation extraction to a significant 2-point boost in the Fusion data. Conversely, in the WiCE experiments, while Claim detection experiences a slight decline of almost 2 points, Citation extraction and Fusion demonstrate the opposite trend. Typically, one might expect index compression techniques to yield inferior results compared to the standard Dense retrieval setup due to the lossy nature of compressing embeddings. However, a straightforward explanation for this unexpected improvement lies in the utilization of different pre-trained models for generating the embeddings. In the standard Dense retrieval, we rely on the all-MiniLM-L6-v2 model, which maps sentences and paragraphs to a 384-dimensional dense vector space. In contrast, the JPQ model employed for index compression initially generates embeddings of size 768 and subsequently reduces the embedding size using PQ centroids to achieve smaller vector sizes. Furthermore, it's worth noting that JPQ learns the index for the query vectors, unlike the approach in standard Dense retrieval where the index is kept separate. The latter essentially operates in a zero-shot inference manner, as we do not fine-tune the encoders on specific datasets but instead store and retrieve the created embeddings directly in our FAISS setup.

### 5.3.4 Key Takeaways

Enhancing Dense retrieval through the use of index compression via the JPQ model has remarkably reduced the index size for Dense retrieval by a substantial 93%. Further analysis indicates significant speedups of up to 10.0x for the CPU-based approach, while the GPU-based approach achieves a modest speedup of up to 2.0x in the HoVer experiments. However, it experiences a slight slowdown in the WiCE experiments. A huge emphasis on achieving efficiency is particularly pertinent in the context of CPU-based Dense Retrieval with index compression. Here the latency times of the CPU-based approach come in close to the GPU-based approach. These findings not only signify efficiency gains concerning resource utilization for index storage, but also pave the way for experiments on lower-end machines especially ones lacking GPU capabilities. Thereby maximizing the benefits of CPU-based methodologies. Regarding performance, experiments involving index compression generally outperform standard Dense retrieval. This superiority can be attributed to the utilization of different pre-trained models and learned index techniques, resulting in slightly enhanced outcomes.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

This research study aimed to explore the impact of indexing supporting facts from large data collection as to improve fact-checking pipelines. Existing work on the fact-checking process focuses primarily on the fact-verification part, but not on the evidence retrieval part. Here we explored three distinct fact-extraction methods to attain supporting facts, namely, the use of a claim detection model, citation extraction and a fusion of the two. Aside from methods for identifying the supporting facts, we also explored various retrieval methods for indexing these. To guide us in this research, we aimed to answer the following three research questions:

- RQ 1: How does indexing supporting facts improve information retrieval efficiency?
- RQ 2: How does indexing supporting facts affect overall pipeline efficiency and downstream fact-checking performance?
- RQ 3: In what ways does index compression enhance the efficiency of dense retrieval and fact-checking systems?

From our in-depth analysis, we made several observations. Incorporating supporting facts into both Sparse and Dense retrieval setups leads to significant enhancements in overall pipeline efficiency. Sparse retrieval setups experience speedups of up to around 1.5x, while Dense retrieval setups show even more substantial improvements, reaching up to approximately 20.0x with GPU-based approaches. These speedups are primarily due to the removal of the Sentence Retrieval stage, which incurs considerable latency overhead. While there may be a minor decline in performance when transitioning from Sparse to Dense retrieval, particularly in citation extraction, Fusion data often yields promising results, emphasizing the importance of amalgamating various extraction techniques. Ablation experiments on the Sentence Retrieval stage reveal its marginal contribution to performance improvement. Comparisons between original data and supporting facts data show only a slight decline in performance, showcasing that utilizing only supporting facts incurs a modest loss. Further experiments incorporating index compression substantially reduce the index size for Dense retrieval by 93%, resulting in significant speedups, particularly for CPU-based approaches. Here a key observation to be made is that CPU-based approaches, which experience up to 10.0x the speedup, come in close to the latency times of GPU-based approaches. Thereby effectively paving the way for fact-checking pipelines to be run on lower-end machines, that do not benefit from more GPU-based methodologies but do for

CPU-based ones.

All these findings underscore the meaningful impact of indexing just the supporting facts on retrieval efficiency and consequently the overall pipeline efficiency, with only minimal losses in downstream fact-checking performance. With optimizing efficiency comes several implications for which real-world applications stand to benefit significantly. Facilitating faster fact-checking processes enables various organizations to verify information in real-time where timely verification is paramount. For example in journalism where news organizations need to verify the authenticity of new information before being able to publish any reports. These advancements also contribute to scalability and resource efficiency, enabling fact-checking solutions to be deployed on a wider range of devices, including those with limited computational resources. This accessibility expands the reach of fact-checking technology, empowering users to make informed decisions and navigate online content more effectively. Moreover, the conservation of resources, such as reduced energy consumption, adds an eco-friendly dimension, rendering these solutions more appealing to companies and platforms. In essence, streamlining the general fact-checking process promises to yield significant benefits, ultimately aiding in the fight against misinformation and disinformation prevalent in the online sphere.

## 6.2 Future Work

To foster continued research related to the topic of our research, we identified several limitations in our work. Additionally, we have pinpointed several areas for improvement aimed at enhancing the efficiency and performance of the fact-checking process, while also facilitating a more comprehensive analysis.

### 6.2.1 Limitations

**Indexing granularity** In our re-ranking of the corpus data, we processed the article text at a sentence level. However, for use in the fact-checking pipeline, we concatenated sentences into single texts per article. This does come with the risk of possible topic drift due to sentences, that may be claim-worthy on their own, but not pertaining to the claim also being used as part of the retrieved relevant documents. To alleviate this, exploring alternative granularities like storing sentence-level text instead of on a per-document article basis could be beneficial. However, this may sacrifice crucial contextual information that is part of the evidence sentences. Additionally, it will increase computational complexity in creating the retrieval index. It should be noted that for dense retrieval where generating text embeddings becomes more intensive, techniques like index compression would play a more crucial role in alleviating that problem. Therefore, thorough experimentation and analysis with regard to the granularity level would make for another important step towards efficiency.

**Retrieval methods** In our study, we focused solely on a select few retrieval methods, thereby limiting the scope of our findings to these specific techniques. Consequently, we advocate for enhanced generalizability by delving into a broader range of Sparse and Dense retrieval methods, alongside investigating varied index compression methodologies. Moreover, we encourage the exploration of alternative retrieval approaches not covered in our analysis, which could potentially yield greater efficiency gains or mitigate performance degradation. Furthermore, regarding index compression, our experimentation was confined to the utilization of 96 subvectors. While our initial findings demonstrated considerable promise in terms of efficiency, we suggest a more comprehensive examination involving varying quantities of subvectors. This is to ascertain a deeper understanding of their impact in not only the efficiency aspect but also the performance. Lastly, we also advocate for a more in-depth evaluation of the retrieval performance as for our work we only looked at the efficiency in terms of latency.

**Implementation efficiency** Although the code we used for running the experiments sufficed for the research we performed. We do acknowledge that certain parts of the code could have been better optimised. Notably, the fact-checking pipeline, derived from the original HoVer work, relies on outdated dependencies. For our work, we did not update these dependencies as that would require changing their code base or even re-implementing some parts of the code which fell outside the scope of our work. Additionally, comes the fact that HoVer's fact-checking pipeline is limited in its capabilities. For example, the Claim verification part can only evaluate to binary labels and not multi-labels. In a more practical setting, a more granular level of labels can be more helpful in the case of denoting when no information can be found that supports or contradicts a statement. Lastly, the citation extraction implementation, as detailed in Section 4.3, encountered runtime issues that could have been addressed with better optimization strategies about the model usage. Going forward, refining these aspects of the codebase could contribute to more robust and efficient research outcomes.

### 6.2.2 Extensions

**Supporting facts extraction** To enhance efficiency while also optimizing performance in our re-ranking setup, several key improvements can be made. Firstly, upgrading the claim-detection model from an off-the-shelf solution to a custom-trained model would enable better contextual understanding within text spans, allowing for the detection of single or multiple claims within a coherent context. This improvement necessitates a robust dataset for training, presenting an opportunity for further refinement. Secondly, enhancing citation extraction beyond solely selecting sentences containing the citation symbols is crucial. This as surrounding sentences may also be part of the citation and thus contribute to enriching the extracted information. Lastly, exploring alternative fusion methods that integrate both claim detection and citation extraction, along with potentially novel approaches, can lead to more effective re-ranking strategies. By combining these enhancements, we can achieve a more refined and comprehensive system that balances efficiency with performance in our re-ranking workflow.

**More comprehensive analysis** To further enhance our work, it is essential to diversify the data used in experimentation by incorporating a broader range of complex claim datasets, as well as exploring corpus data beyond Wikipedia. It's worth noting that utilizing larger corpus data sizes can yield more realistic results, as practical applications often involve datasets ranging in the hundreds of millions or even larger. This expansion will broaden the scope of our analysis and ensure the robustness and adaptability of our solutions across various contexts. Moreover, analyzing the fact-checking pipeline within a scalable real-world application would be pivotal. Our experiments, conducted on a single machine, have limitations in replicating real-world complexities. By scaling up, we can assess performance under heavier workloads, optimize settings for efficiency, and evaluate robustness to failures. This approach will provide more actionable insights for the practical deployment of fact-checking solutions.

# Bibliography

[1] Amin Ahmad, Noah Constant, Yinfei Yang, and Daniel Cer. Reqa: An evaluation for end-to-end answer retrieval models. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 137–146, Hong Kong, China, nov 2019. Association for Computational Linguistics.

[2] Badr AlKhamissi, Millicent Li, Asli Celikyilmaz, Mona Diab, and Marjan Ghazvininejad. A review on language models as knowledge bases, 2022.

[3] Uri Alon, Frank F. Xu, Junxian He, Sudipta Sengupta, Dan Roth, and Graham Neubig. Neuro-symbolic language modeling with automaton-augmented retrieval, 2022.

[4] Amos Azaria and Tom Mitchell. The internal state of an llm knows when its lying, 2023.

[5] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors, 2018.

[6] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization, 2021.

[7] Christopher Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11, 01 2010.

[8] Anthony Chen, Panupong Pasupat, Sameer Singh, Hongrae Lee, and Kelvin Guu. Purr: Efficiently editing language model hallucinations by denoising language model corruptions, 2023.

[9] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions, 2017.

[10] Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. Rarr: Researching and revising what language models say, using language models, 2023.

[11] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014.

[12] Zhijiang Guo, Michael Schlichtkrull, and Andreas Vlachos. A survey on automated fact-checking. *Transactions of the Association for Computational Linguistics*, 10:178–206, 2022.

[13] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020.

[14] Yikun Han, Chunjiang Liu, and Pengfei Wang. A comprehensive survey on vector database: Storage and retrieval technique, challenge, 2023.

[15] Benjamin Heinzerling and Kentaro Inui. Language models as knowledge bases: On entity representations, storage capacity, and paraphrased queries. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1772–1791, Online, April 2021. Association for Computational Linguistics.

[16] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), mar 2023.

[17] Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. Hover: A dataset for many-hop fact extraction and claim verification, 2020.

[18] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[19] Ryo Kamoi, Tanya Goyal, Juan Rodriguez, and Greg Durrett. WiCE: Real-world entailment for claims in Wikipedia. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7561–7583, Singapore, December 2023. Association for Computational Linguistics.

[20] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020.

[21] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models, 2020.

[22] Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. Copy is all you need. In *The Eleventh International Conference on Learning Representations*, 2023.

[23] Eric Lazarski, Mahmood Al-Khassaweneh, and Cynthia Howard. Using nlp for fact checking: A survey. *Designs*, 5(3), 2021.

[24] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[25] Junyi Li, Xiaoxue Cheng, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. HaluEval: A Large-Scale Hallucination Evaluation Benchmark for Large Language Models. *arXiv e-prints*, page arXiv:2305.11747, May 2023.

[26] Shaobo Li, Xiaoguang Li, Lifeng Shang, Zhenhua Dong, Chengjie Sun, Bingquan Liu, Zhenzhou Ji, Xin Jiang, and Qun Liu. How pre-trained language models capture factual knowledge? A causal-inspired analysis. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1720–1732. Association for Computational Linguistics, 2022.

[27] Xingxuan Li, Liying Cheng, Qingyu Tan, Hwee Tou Ng, Shafiq Joty, and Lidong Bing. Unlocking temporal question answering for large language models using code execution, 2023.

[28] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation, 2023.

[29] Stanislav Morozov and Artem Babenko. Unsupervised neural quantization for compressed-domain similarity search, 2019.

[30] Yixin Nie, Haonan Chen, and Mohit Bansal. Combining fact extraction and verification with neural semantic matching networks, 2018.

[31] OpenAI. Gpt-4 technical report, 2023.

[32] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[33] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. Check your facts and try again: Improving large language models with external knowledge and automated feedback.

[34] Fabio Petroni, Samuel Broscheit, Aleksandra Piktus, Patrick Lewis, Gautier Izacard, Lucas Hosseini, Jane Dwivedi-Yu, Maria Lomeli, Timo Schick, Pierre-Emmanuel Mazaré, Armand Joulin, Edouard Grave, and Sebastian Riedel. Improving wikipedia verifiability with ai, 2022.

[35] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. 2023.

[36] Paul Roit, Johan Ferret, Lior Shani, Roee Aharoni, Geoffrey Cideron, Robert Dadashi, Matthieu Geist, Sertan Girgin, Léonard Hussenot, Orgad Keller, Nikola Momchev, Sabela Ramos, Piotr Stanczyk, Nino Vieillard, Olivier Bachem, Gal Elidan, Avinatan Hassidim, Olivier Pietquin, and Idan Szpektor. Factually consistent summarization via reinforcement learning with textual entailment feedback, 2023.

[37] Aalok Sathe, Salar Ather, Tuan Manh Le, Nathan Perry, and Joonsuk Park. Automated fact-checking of claims from Wikipedia. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara,

Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 6874–6882, Marseille, France, May 2020. European Language Resources Association.

[38] Qingyu Tan, Hwee Tou Ng, and Lidong Bing. Towards benchmarking and improving the temporal reasoning capability of large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14820–14835, Toronto, Canada, July 2023. Association for Computational Linguistics.

[39] James Thorne and Andreas Vlachos. Automated fact checking: Task formulations, methods and future directions, 2018.

[40] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification, 2018.

[41] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[42] Logesh Kumar Umapathi, Ankit Pal, and Malaikannan Sankarasubbu. Med-halt: Medical domain hallucination test for large language models, 2023.

[43] Zhimin Wei, Xiaowei Xu, Chenglin Wang, Zhenyu Liu, Peng Xin, and Wei Zhang. An index construction and similarity retrieval method based on sentence-bert. In *2022 7th International Conference on Image, Vision and Computing (ICIVC)*, pages 934–938, 2022.

[44] Xiang Yue, Boshi Wang, Kai Zhang, Ziru Chen, Yu Su, and Huan Sun. Automatic evaluation of attribution by large language models, 2023.

[45] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Jointly optimizing query encoder and product quantization to improve retrieval performance, 2021.

[46] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing dense retrieval model training with hard negatives, 2021.

[47] Yizhe Zhang, Siqi Sun, Xiang Gao, Yuwei Fang, Chris Brockett, Michel Galley, Jianfeng Gao, and Bill Dolan. Retgen: A joint framework for retrieval and grounded text generation modeling. 2022.

[48] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. Siren's song in the ai ocean: A survey on hallucination in large language models, 2023.

[49] Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. Dense text retrieval based on pretrained language models: A survey, 2022.

[50] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proc. VLDB Endow.*, 16(8):1979–1991, jun 2023.

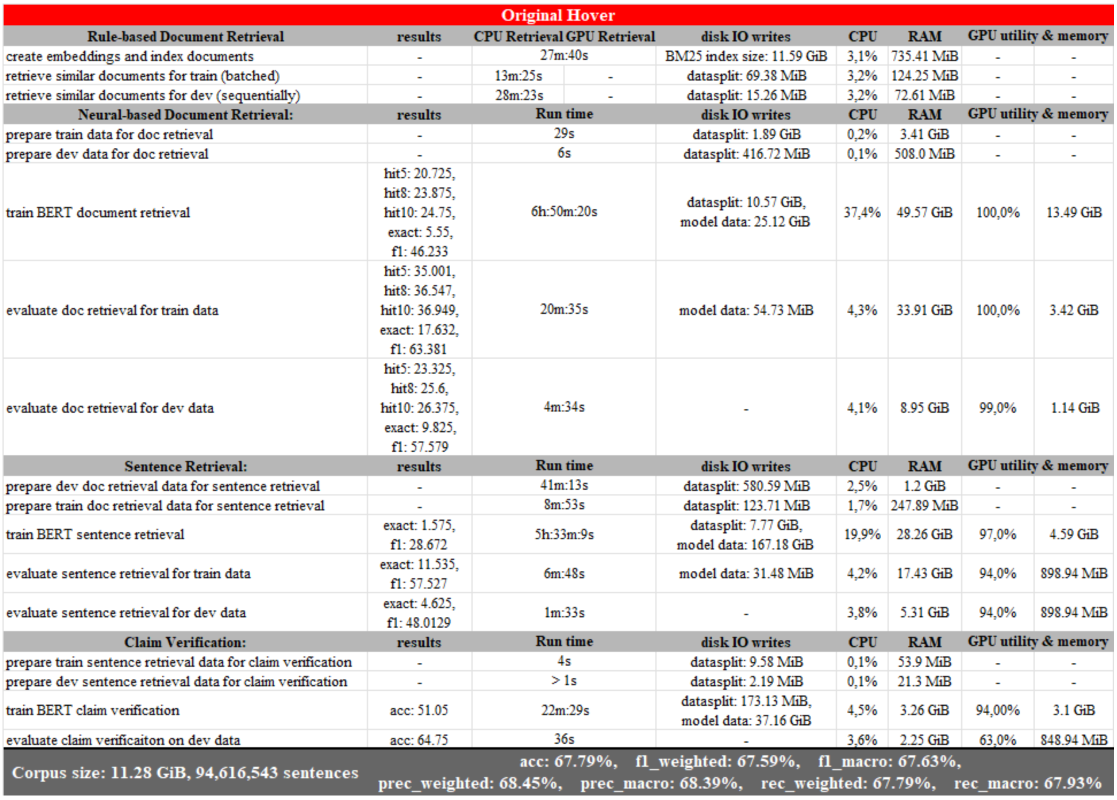
[51] Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. A survey on model compression for large language models, 2023.

# Appendix A

# Detailed HoVer Experiments Measurements

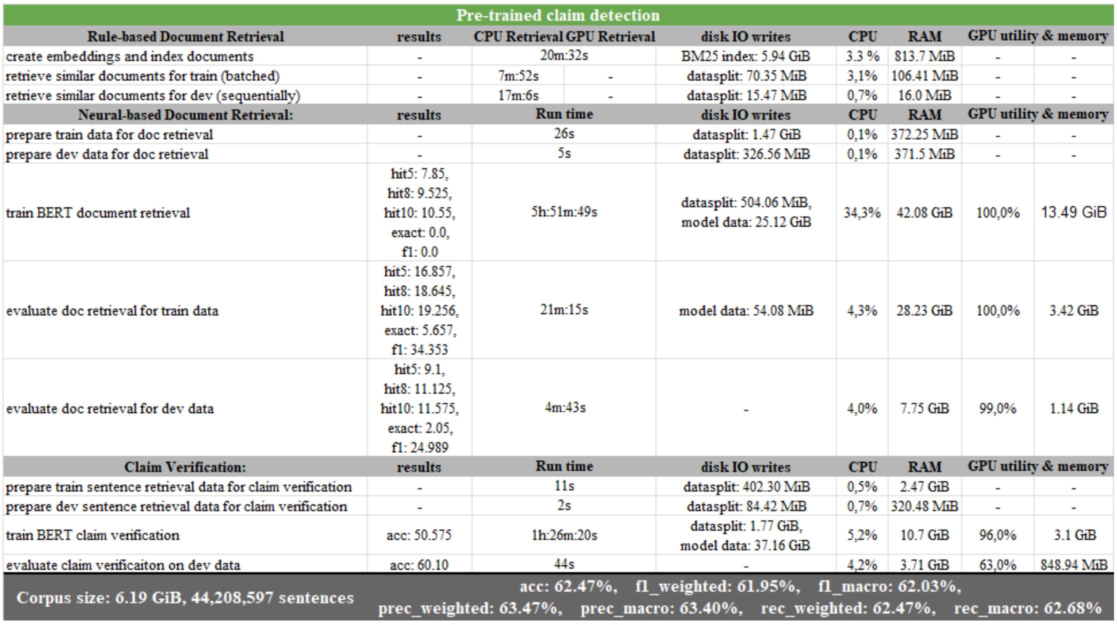
This chapter provides step-by-step measurement tables (refer to Section 4.4) for each step of the pipeline on the HoVer claim dataset. It's organized into three sections based on index and retrieval settings. Within each section, tables depict the baseline, reranking, and ablation setups for data corpus indexing (see Chapter 3).

## A.1 Sparse retrieval setup with Reranking



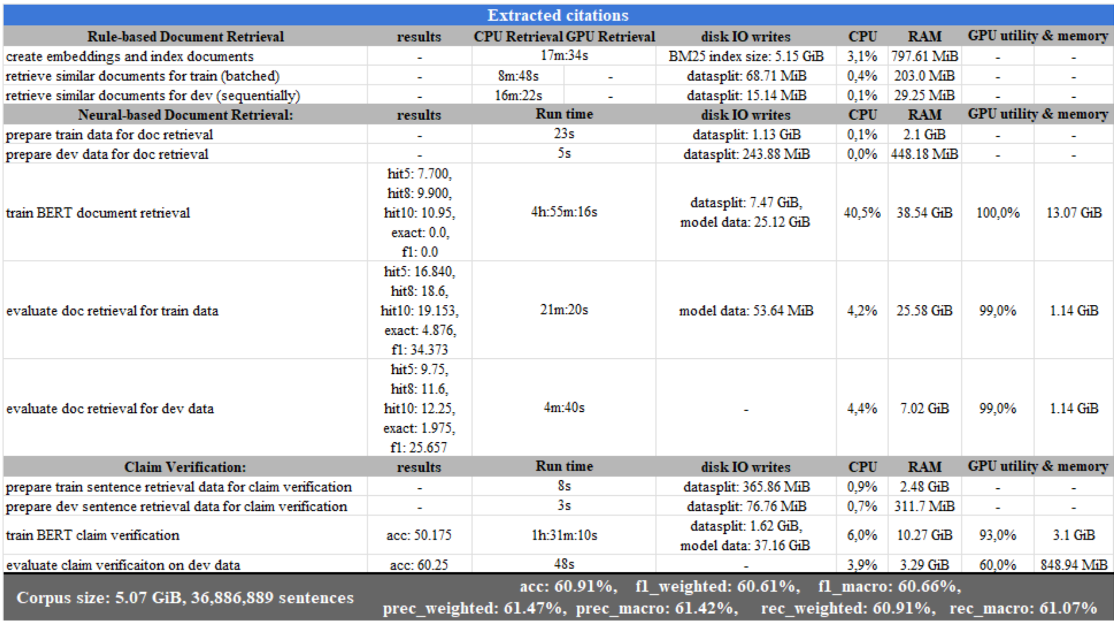
| Original Hover | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 27m:40s | | BM25 index size: 11.59 GiB | 3,1% | 735.41 MiB | - | - |
| retrieve similar documents for train (batched) | - | 13m:25s | - | datasplit: 69.38 MiB | 3,2% | 124.25 MiB | - | - |
| retrieve similar documents for dev (sequentially) | - | 28m:23s | - | datasplit: 15.26 MiB | 3,2% | 72.61 MiB | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 29s | | datasplit: 1.89 GiB | 0,2% | 3.41 GiB | - | - |
| prepare dev data for doc retrieval | - | 6s | | datasplit: 416.72 MiB | 0,1% | 508.0 MiB | - | - |
| train BERT document retrieval | hit5: 20.725,<br>hit8: 23.875,<br>hit10: 24.75,<br>exact: 5.55,<br>f1: 46.233 | 6h:50m:20s | | datasplit: 10.57 GiB,<br>model data: 25.12 GiB | 37,4% | 49.57 GiB | 100,0% | 13.49 GiB |
| evaluate doc retrieval for train data | hit5: 35.001,<br>hit8: 36.547,<br>hit10: 36.949,<br>exact: 17.632,<br>f1: 63.381 | 20m:35s | | model data: 54.73 MiB | 4,3% | 33.91 GiB | 100,0% | 3.42 GiB |
| evaluate doc retrieval for dev data | hit5: 23.325,<br>hit8: 25.6,<br>hit10: 26.375,<br>exact: 9.825,<br>f1: 57.579 | 4m:34s | | - | 4,1% | 8.95 GiB | 99,0% | 1.14 GiB |
| **Sentence Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare dev doc retrieval data for sentence retrieval | - | 41m:13s | | datasplit: 580.59 MiB | 2,5% | 1.2 GiB | - | - |
| prepare train doc retrieval data for sentence retrieval | - | 8m:53s | | datasplit: 123.71 MiB | 1,7% | 247.89 MiB | - | - |
| train BERT sentence retrieval | exact: 1.575,<br>f1: 28.672 | 5h:33m:9s | | datasplit: 7.77 GiB,<br>model data: 167.18 GiB | 19,9% | 28.26 GiB | 97,0% | 4.59 GiB |
| evaluate sentence retrieval for train data | exact: 11.535,<br>f1: 57.527 | 6m:48s | | model data: 31.48 MiB | 4,2% | 17.43 GiB | 94,0% | 898.94 MiB |
| evaluate sentence retrieval for dev data | exact: 4.625,<br>f1: 48.0129 | 1m:33s | | - | 3,8% | 5.31 GiB | 94,0% | 898.94 MiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 4s | | datasplit: 9.58 MiB | 0,1% | 53.9 MiB | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 2.19 MiB | 0,1% | 21.3 MiB | - | - |
| train BERT claim verification | acc: 51.05 | 22m:29s | | datasplit: 173.13 MiB,<br>model data: 37.16 GiB | 4,5% | 3.26 GiB | 94,00% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 64.75 | 36s | | - | 3,6% | 2.25 GiB | 63,0% | 848.94 MiB |
| **Corpus size: 11.28 GiB, 94,616,543 sentences** | **acc: 67.79%, f1_weighted: 67.59%, f1_macro: 67.63%,<br>prec_weighted: 68.45%, prec_macro: 68.39%, rec_weighted: 67.79%, rec_macro: 67.93%** | | | | | | | |

Figure A.1: BM25 retrieval on the original HoVer data corpus.



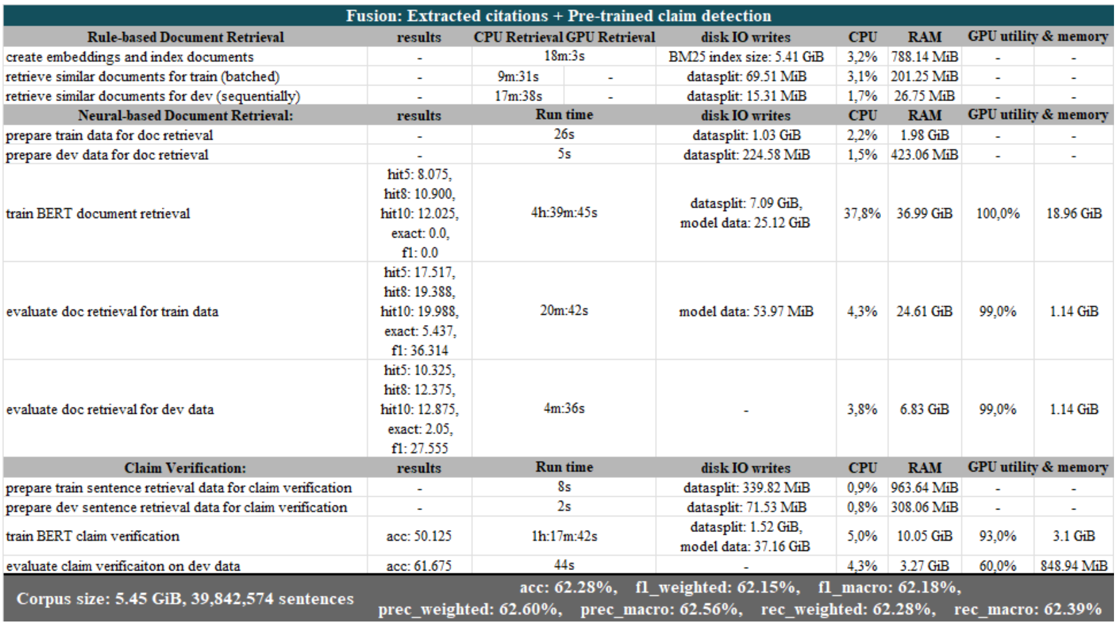
| Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 20m:32s | | BM25 index: 5.94 GiB | 3.3 % | 813.7 MiB | - | - |
| retrieve similar documents for train (batched) | - | 7m:52s | - | datasplit: 70.35 MiB | 3,1% | 106.41 MiB | - | - |
| retrieve similar documents for dev (sequentially) | - | 17m:6s | - | datasplit: 15.47 MiB | 0,7% | 16.0 MiB | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 26s | | datasplit: 1.47 GiB | 0,1% | 372.25 MiB | - | - |
| prepare dev data for doc retrieval | - | 5s | | datasplit: 326.56 MiB | 0,1% | 371.5 MiB | - | - |
| train BERT document retrieval | hit5: 7.85,<br>hit8: 9.525,<br>hit10: 10.55,<br>exact: 0.0,<br>f1: 0.0 | 5h:51m:49s | | datasplit: 504.06 MiB,<br>model data: 25.12 GiB | 34,3% | 42.08 GiB | 100,0% | 13.49 GiB |
| evaluate doc retrieval for train data | hit5: 16.857,<br>hit8: 18.645,<br>hit10: 19.256,<br>exact: 5.657,<br>f1: 34.353 | 21m:15s | | model data: 54.08 MiB | 4,3% | 28.23 GiB | 100,0% | 3.42 GiB |
| evaluate doc retrieval for dev data | hit5: 9.1,<br>hit8: 11.125,<br>hit10: 11.575,<br>exact: 2.05,<br>f1: 24.989 | 4m:43s | | - | 4,0% | 7.75 GiB | 99,0% | 1.14 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 11s | | datasplit: 402.30 MiB | 0,5% | 2.47 GiB | - | - |
| prepare dev sentence retrieval data for claim verification | - | 2s | | datasplit: 84.42 MiB | 0,7% | 320.48 MiB | - | - |
| train BERT claim verification | acc: 50.575 | 1h:26m:20s | | datasplit: 1.77 GiB,<br>model data: 37.16 GiB | 5,2% | 10.7 GiB | 96,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.10 | 44s | | - | 4,2% | 3.71 GiB | 63,0% | 848.94 MiB |
| **Corpus size: 6.19 GiB, 44,208,597 sentences** | **acc: 62.47%, f1_weighted: 61.95%, f1_macro: 62.03%,<br>prec_weighted: 63.47%, prec_macro: 63.40%, rec_weighted: 62.47%, rec_macro: 62.68%** | | | | | | | |

Figure A.2: BM25 retrieval on the claim-detected HoVer data corpus.



| Extracted citations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 17m:34s | | BM25 index size: 5.15 GiB | 3,1% | 797.61 MiB | - | - |
| retrieve similar documents for train (batched) | - | 8m:48s | - | datasplit: 68.71 MiB | 0,4% | 203.0 MiB | - | - |
| retrieve similar documents for dev (sequentially) | - | 16m:22s | - | datasplit: 15.14 MiB | 0,1% | 29.25 MiB | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 23s | | datasplit: 1.13 GiB | 0,1% | 2.1 GiB | - | - |
| prepare dev data for doc retrieval | - | 5s | | datasplit: 243.88 MiB | 0,0% | 448.18 MiB | - | - |
| train BERT document retrieval | hit5: 7.700,<br>hit8: 9.900,<br>hit10: 10.95,<br>exact: 0.0,<br>f1: 0.0 | 4h:55m:16s | | datasplit: 7.47 GiB,<br>model data: 25.12 GiB | 40,5% | 38.54 GiB | 100,0% | 13.07 GiB |
| evaluate doc retrieval for train data | hit5: 16.840,<br>hit8: 18.6,<br>hit10: 19.153,<br>exact: 4.876,<br>f1: 34.373 | 21m:20s | | model data: 53.64 MiB | 4,2% | 25.58 GiB | 99,0% | 1.14 GiB |
| evaluate doc retrieval for dev data | hit5: 9.75,<br>hit8: 11.6,<br>hit10: 12.25,<br>exact: 1.975,<br>f1: 25.657 | 4m:40s | | - | 4,4% | 7.02 GiB | 99,0% | 1.14 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 8s | | datasplit: 365.86 MiB | 0,9% | 2.48 GiB | - | - |
| prepare dev sentence retrieval data for claim verification | - | 3s | | datasplit: 76.76 MiB | 0,7% | 311.7 MiB | - | - |
| train BERT claim verification | acc: 50.175 | 1h:31m:10s | | datasplit: 1.62 GiB,<br>model data: 37.16 GiB | 6,0% | 10.27 GiB | 93,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.25 | 48s | | - | 3,9% | 3.29 GiB | 60,0% | 848.94 MiB |
| **Corpus size: 5.07 GiB, 36,886,889 sentences** | **acc: 60.91%, f1_weighted: 60.61%, f1_macro: 60.66%,<br>prec_weighted: 61.47%, prec_macro: 61.42%, rec_weighted: 60.91%, rec_macro: 61.07%** | | | | | | | |

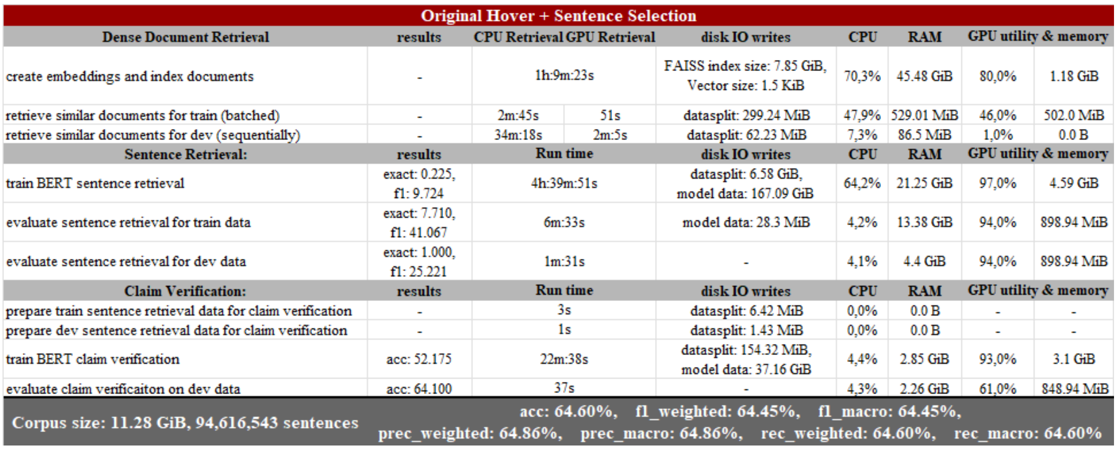
Figure A.3: BM25 retrieval on citation extracted HoVer data corpus.



| Fusion: Extracted citations + Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 18m:3s | | BM25 index size: 5.41 GiB | 3,2% | 788.14 MiB | - | - |
| retrieve similar documents for train (batched) | - | 9m:31s | - | datasplit: 69.51 MiB | 3,1% | 201.25 MiB | - | - |
| retrieve similar documents for dev (sequentially) | - | 17m:38s | - | datasplit: 15.31 MiB | 1,7% | 26.75 MiB | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 26s | | datasplit: 1.03 GiB | 2,2% | 1.98 GiB | - | - |
| prepare dev data for doc retrieval | - | 5s | | datasplit: 224.58 MiB | 1,5% | 423.06 MiB | - | - |
| train BERT document retrieval | hit5: 8.075,<br>hit8: 10.900,<br>hit10: 12.025,<br>exact: 0.0,<br>f1: 0.0 | 4h:39m:45s | | datasplit: 7.09 GiB,<br>model data: 25.12 GiB | 37,8% | 36.99 GiB | 100,0% | 18.96 GiB |
| evaluate doc retrieval for train data | hit5: 17.517,<br>hit8: 19.388,<br>hit10: 19.988,<br>exact: 5.437,<br>f1: 36.314 | 20m:42s | | model data: 53.97 MiB | 4,3% | 24.61 GiB | 99,0% | 1.14 GiB |
| evaluate doc retrieval for dev data | hit5: 10.325,<br>hit8: 12.375,<br>hit10: 12.875,<br>exact: 2.05,<br>f1: 27.555 | 4m:36s | | - | 3,8% | 6.83 GiB | 99,0% | 1.14 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 8s | | datasplit: 339.82 MiB | 0,9% | 963.64 MiB | - | - |
| prepare dev sentence retrieval data for claim verification | - | 2s | | datasplit: 71.53 MiB | 0,8% | 308.06 MiB | - | - |
| train BERT claim verification | acc: 50.125 | 1h:17m:42s | | datasplit: 1.52 GiB,<br>model data: 37.16 GiB | 5,0% | 10.05 GiB | 93,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 61.675 | 44s | | - | 4,3% | 3.27 GiB | 60,0% | 848.94 MiB |
| **Corpus size: 5.45 GiB, 39,842,574 sentences** | **acc: 62.28%, f1_weighted: 62.15%, f1_macro: 62.18%,<br>prec_weighted: 62.60%, prec_macro: 62.56%, rec_weighted: 62.28%, rec_macro: 62.39%** | | | | | | | |

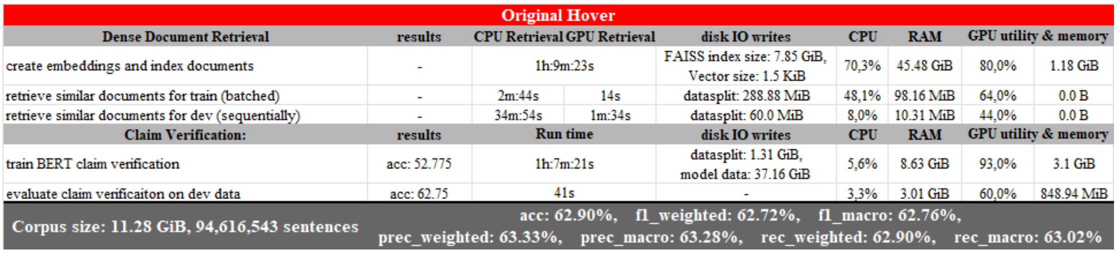
Figure A.4: BM25 retrieval on the fusion (citations + claim-detected) HoVer data corpus.

## A.2 Dense retrieval setup



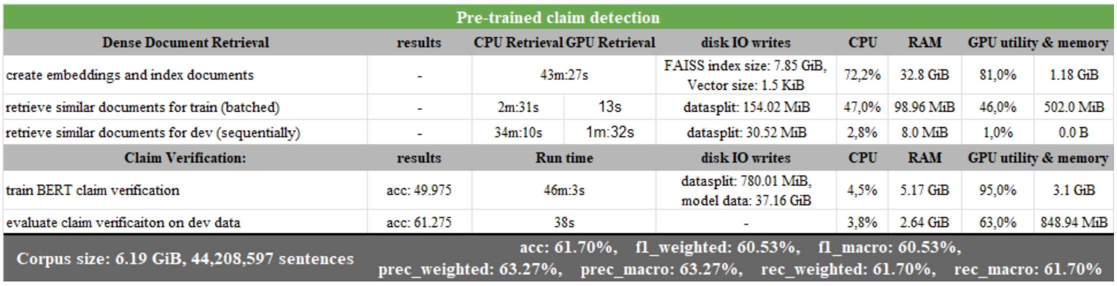
| Original Hover + Sentence Selection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 1h:9m:23s | | FAISS index size: 7.85 GiB,<br>Vector size: 1.5 KiB | 70,3% | 45.48 GiB | 80,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 2m:45s | 51s | datasplit: 299.24 MiB | 47,9% | 529.01 MiB | 46,0% | 502.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 34m:18s | 2m:5s | datasplit: 62.23 MiB | 7,3% | 86.5 MiB | 1,0% | 0.0 B |
| **Sentence Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT sentence retrieval | exact: 0.225,<br>f1: 9.724 | 4h:39m:51s | | datasplit: 6.58 GiB,<br>model data: 167.09 GiB | 64,2% | 21.25 GiB | 97,0% | 4.59 GiB |
| evaluate sentence retrieval for train data | exact: 7.710,<br>f1: 41.067 | 6m:33s | | model data: 28.3 MiB | 4,2% | 13.38 GiB | 94,0% | 898.94 MiB |
| evaluate sentence retrieval for dev data | exact: 1.000,<br>f1: 25.221 | 1m:31s | | - | 4,1% | 4.4 GiB | 94,0% | 898.94 MiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 3s | | datasplit: 6.42 MiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | 1s | | datasplit: 1.43 MiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 52.175 | 22m:38s | | datasplit: 154.32 MiB,<br>model data: 37.16 GiB | 4,4% | 2.85 GiB | 93,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 64.100 | 37s | | - | 4,3% | 2.26 GiB | 61,0% | 848.94 MiB |
| **Corpus size: 11.28 GiB, 94,616,543 sentences** | **acc: 64.60%, f1_weighted: 64.45%, f1_macro: 64.45%,<br>prec_weighted: 64.86%, prec_macro: 64.86%, rec_weighted: 64.60%, rec_macro: 64.60%** | | | | | | | |

Figure A.5: FAISS retrieval with Sentence-Selection stage on the original HoVer data corpus.



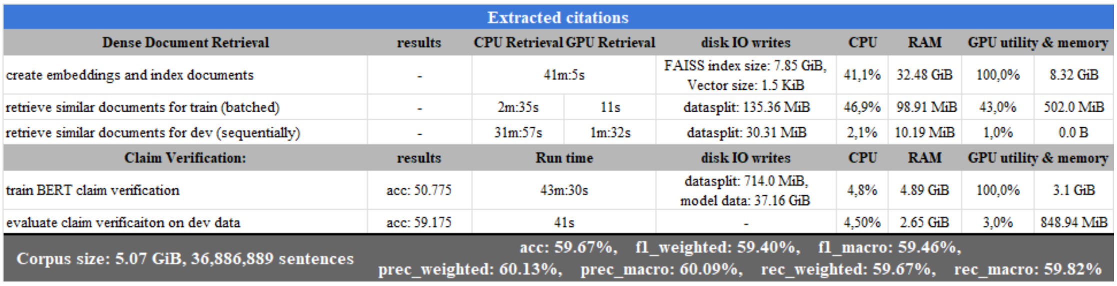
| Original Hover | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 1h:9m:23s | | FAISS index size: 7.85 GiB,<br>Vector size: 1.5 KiB | 70,3% | 45.48 GiB | 80,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 2m:44s | 14s | datasplit: 288.88 MiB | 48,1% | 98.16 MiB | 64,0% | 0.0 B |
| retrieve similar documents for dev (sequentially) | - | 34m:54s | 1m:34s | datasplit: 60.0 MiB | 8,0% | 10.31 MiB | 44,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 52.775 | 1h:7m:21s | | datasplit: 1.31 GiB,<br>model data: 37.16 GiB | 5,6% | 8.63 GiB | 93,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 62.75 | 41s | | - | 3,3% | 3.01 GiB | 60,0% | 848.94 MiB |
| **Corpus size: 11.28 GiB, 94,616,543 sentences** | **acc: 62.90%, f1_weighted: 62.72%, f1_macro: 62.76%,<br>prec_weighted: 63.33%, prec_macro: 63.28%, rec_weighted: 62.90%, rec_macro: 63.02%** | | | | | | | |

Figure A.6: FAISS retrieval on the original HoVer data corpus.



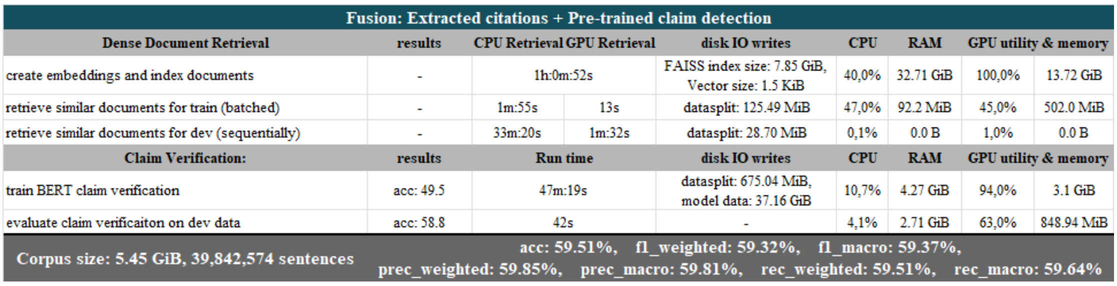
| Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 43m:27s | | FAISS index size: 7.85 GiB,<br>Vector size: 1.5 KiB | 72,2% | 32.8 GiB | 81,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 2m:31s | 13s | datasplit: 154.02 MiB | 47,0% | 98.96 MiB | 46,0% | 502.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 34m:10s | 1m:32s | datasplit: 30.52 MiB | 2,8% | 8.0 MiB | 1,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 49.975 | 46m:3s | | datasplit: 780.01 MiB,<br>model data: 37.16 GiB | 4,5% | 5.17 GiB | 95,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 61.275 | 38s | | - | 3,8% | 2.64 GiB | 63,0% | 848.94 MiB |
| **Corpus size: 6.19 GiB, 44,208,597 sentences** | **acc: 61.70%, f1_weighted: 60.53%, f1_macro: 60.53%, prec_weighted: 63.27%, prec_macro: 63.27%, rec_weighted: 61.70%, rec_macro: 61.70%** | | | | | | | |

Figure A.7: FAISS retrieval on the claim-detected HoVer data corpus.



| Extracted citations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 41m:5s | | FAISS index size: 7.85 GiB,<br>Vector size: 1.5 KiB | 41,1% | 32.48 GiB | 100,0% | 8.32 GiB |
| retrieve similar documents for train (batched) | - | 2m:35s | 11s | datasplit: 135.36 MiB | 46,9% | 98.91 MiB | 43,0% | 502.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 31m:57s | 1m:32s | datasplit: 30.31 MiB | 2,1% | 10.19 MiB | 1,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 50.775 | 43m:30s | | datasplit: 714.0 MiB,<br>model data: 37.16 GiB | 4,8% | 4.89 GiB | 100,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 59.175 | 41s | | - | 4,50% | 2.65 GiB | 3,0% | 848.94 MiB |
| **Corpus size: 5.07 GiB, 36,886,889 sentences** | **acc: 59.67%, f1_weighted: 59.40%, f1_macro: 59.46%, prec_weighted: 60.13%, prec_macro: 60.09%, rec_weighted: 59.67%, rec_macro: 59.82%** | | | | | | | |

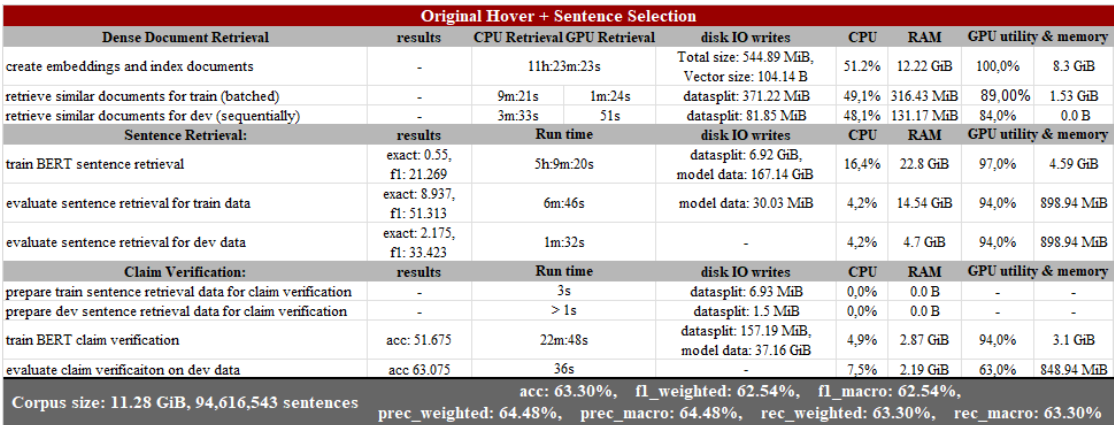
Figure A.8: FAISS retrieval on citation extracted HoVer data corpus.



| Fusion: Extracted citations + Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 1h:0m:52s | | FAISS index size: 7.85 GiB,<br>Vector size: 1.5 KiB | 40,0% | 32.71 GiB | 100,0% | 13.72 GiB |
| retrieve similar documents for train (batched) | - | 1m:55s | 13s | datasplit: 125.49 MiB | 47,0% | 92.2 MiB | 45,0% | 502.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 33m:20s | 1m:32s | datasplit: 28.70 MiB | 0,1% | 0.0 B | 1,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 49.5 | 47m:19s | | datasplit: 675.04 MiB,<br>model data: 37.16 GiB | 10,7% | 4.27 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 58.8 | 42s | | - | 4,1% | 2.71 GiB | 63,0% | 848.94 MiB |
| **Corpus size: 5.45 GiB, 39,842,574 sentences** | **acc: 59.51%, f1_weighted: 59.32%, f1_macro: 59.37%, prec_weighted: 59.85%, prec_macro: 59.81%, rec_weighted: 59.51%, rec_macro: 59.64%** | | | | | | | |

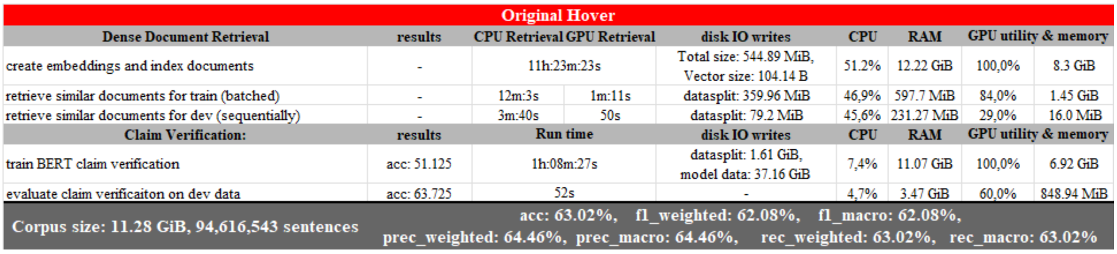
Figure A.9: FAISS retrieval on the fusion (citations + claim-detected) HoVer data corpus.

## A.3 Index compression setup



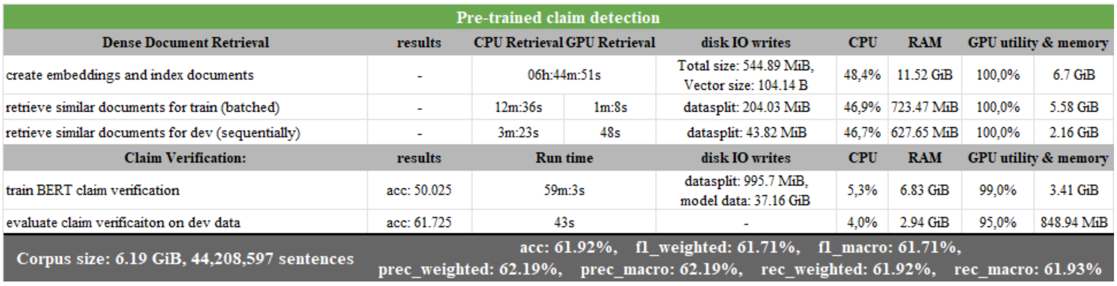
| Original Hover + Sentence Selection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 11h:23m:23s | | Total size: 544.89 MiB,<br>Vector size: 104.14 B | 51.2% | 12.22 GiB | 100,0% | 8.3 GiB |
| retrieve similar documents for train (batched) | - | 9m:21s | 1m:24s | datasplit: 371.22 MiB | 49,1% | 316.43 MiB | 89,00% | 1.53 GiB |
| retrieve similar documents for dev (sequentially) | - | 3m:33s | 51s | datasplit: 81.85 MiB | 48,1% | 131.17 MiB | 84,0% | 0.0 B |
| **Sentence Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT sentence retrieval | exact: 0.55,<br>f1: 21.269 | 5h:9m:20s | | datasplit: 6.92 GiB,<br>model data: 167.14 GiB | 16,4% | 22.8 GiB | 97,0% | 4.59 GiB |
| evaluate sentence retrieval for train data | exact: 8.937,<br>f1: 51.313 | 6m:46s | | model data: 30.03 MiB | 4,2% | 14.54 GiB | 94,0% | 898.94 MiB |
| evaluate sentence retrieval for dev data | exact: 2.175,<br>f1: 33.423 | 1m:32s | | - | 4,2% | 4.7 GiB | 94,0% | 898.94 MiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 3s | | datasplit: 6.93 MiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 1.5 MiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 51.675 | 22m:48s | | datasplit: 157.19 MiB,<br>model data: 37.16 GiB | 4,9% | 2.87 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc 63.075 | 36s | | - | 7,5% | 2.19 GiB | 63,0% | 848.94 MiB |
| **Corpus size: 11.28 GiB, 94,616,543 sentences** | **acc: 63.30%, f1_weighted: 62.54%, f1_macro: 62.54%,<br>prec_weighted: 64.48%, prec_macro: 64.48%, rec_weighted: 63.30%, rec_macro: 63.30%** | | | | | | | |

Figure A.10: JPQ retrieval with Sentence-Selection stage on the original HoVer data corpus.



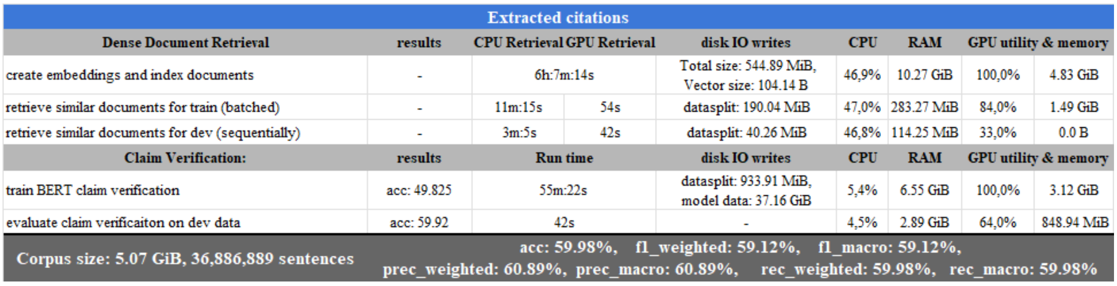
| Original Hover | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 11h:23m:23s | | Total size: 544.89 MiB,<br>Vector size: 104.14 B | 51.2% | 12.22 GiB | 100,0% | 8.3 GiB |
| retrieve similar documents for train (batched) | - | 12m:3s | 1m:11s | datasplit: 359.96 MiB | 46,9% | 597.7 MiB | 84,0% | 1.45 GiB |
| retrieve similar documents for dev (sequentially) | - | 3m:40s | 50s | datasplit: 79.2 MiB | 45,6% | 231.27 MiB | 29,0% | 16.0 MiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 51.125 | 1h:08m:27s | | datasplit: 1.61 GiB,<br>model data: 37.16 GiB | 7,4% | 11.07 GiB | 100,0% | 6.92 GiB |
| evaluate claim verificaiton on dev data | acc: 63.725 | 52s | | - | 4,7% | 3.47 GiB | 60,0% | 848.94 MiB |
| **Corpus size: 11.28 GiB, 94,616,543 sentences** | **acc: 63.02%, f1_weighted: 62.08%, f1_macro: 62.08%,<br>prec_weighted: 64.46%, prec_macro: 64.46%, rec_weighted: 63.02%, rec_macro: 63.02%** | | | | | | | |

Figure A.11: JPQ retrieval on the original HoVer data corpus.



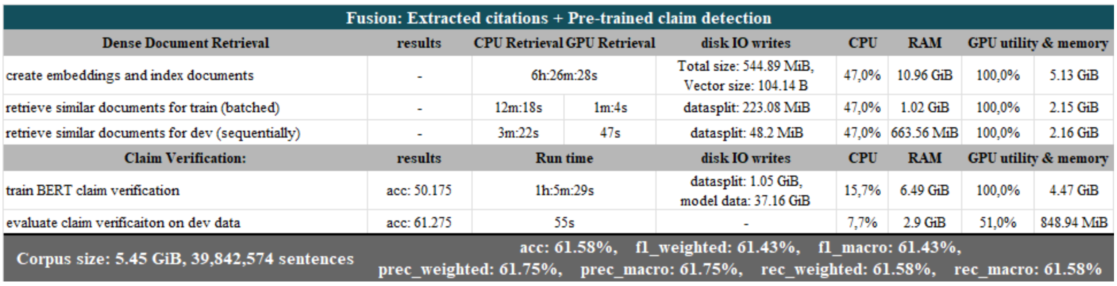
| Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 06h:44m:51s | | Total size: 544.89 MiB,<br>Vector size: 104.14 B | 48,4% | 11.52 GiB | 100,0% | 6.7 GiB |
| retrieve similar documents for train (batched) | - | 12m:36s | 1m:8s | datasplit: 204.03 MiB | 46,9% | 723.47 MiB | 100,0% | 5.58 GiB |
| retrieve similar documents for dev (sequentially) | - | 3m:23s | 48s | datasplit: 43.82 MiB | 46,7% | 627.65 MiB | 100,0% | 2.16 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 50.025 | 59m:3s | | datasplit: 995.7 MiB,<br>model data: 37.16 GiB | 5,3% | 6.83 GiB | 99,0% | 3.41 GiB |
| evaluate claim verificaiton on dev data | acc: 61.725 | 43s | | - | 4,0% | 2.94 GiB | 95,0% | 848.94 MiB |
| **Corpus size: 6.19 GiB, 44,208,597 sentences** | **acc: 61.92%, f1_weighted: 61.71%, f1_macro: 61.71%,<br>prec_weighted: 62.19%, prec_macro: 62.19%, rec_weighted: 61.92%, rec_macro: 61.93%** | | | | | | | |

Figure A.12: JPQ retrieval on the claim-detected HoVer data corpus.



| Extracted citations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 6h:7m:14s | | Total size: 544.89 MiB,<br>Vector size: 104.14 B | 46,9% | 10.27 GiB | 100,0% | 4.83 GiB |
| retrieve similar documents for train (batched) | - | 11m:15s | 54s | datasplit: 190.04 MiB | 47,0% | 283.27 MiB | 84,0% | 1.49 GiB |
| retrieve similar documents for dev (sequentially) | - | 3m:5s | 42s | datasplit: 40.26 MiB | 46,8% | 114.25 MiB | 33,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 49.825 | 55m:22s | | datasplit: 933.91 MiB,<br>model data: 37.16 GiB | 5,4% | 6.55 GiB | 100,0% | 3.12 GiB |
| evaluate claim verificaiton on dev data | acc: 59.92 | 42s | | - | 4,5% | 2.89 GiB | 64,0% | 848.94 MiB |
| **Corpus size: 5.07 GiB, 36,886,889 sentences** | **acc: 59.98%, f1_weighted: 59.12%, f1_macro: 59.12%,<br>prec_weighted: 60.89%, prec_macro: 60.89%, rec_weighted: 59.98%, rec_macro: 59.98%** | | | | | | | |

Figure A.13: JPQ retrieval on citation extracted HoVer data corpus.



| Fusion: Extracted citations + Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 6h:26m:28s | | Total size: 544.89 MiB,<br>Vector size: 104.14 B | 47,0% | 10.96 GiB | 100,0% | 5.13 GiB |
| retrieve similar documents for train (batched) | - | 12m:18s | 1m:4s | datasplit: 223.08 MiB | 47,0% | 1.02 GiB | 100,0% | 2.15 GiB |
| retrieve similar documents for dev (sequentially) | - | 3m:22s | 47s | datasplit: 48.2 MiB | 47,0% | 663.56 MiB | 100,0% | 2.16 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 50.175 | 1h:5m:29s | | datasplit: 1.05 GiB,<br>model data: 37.16 GiB | 15,7% | 6.49 GiB | 100,0% | 4.47 GiB |
| evaluate claim verificaiton on dev data | acc: 61.275 | 55s | | - | 7,7% | 2.9 GiB | 51,0% | 848.94 MiB |
| **Corpus size: 5.45 GiB, 39,842,574 sentences** | **acc: 61.58%, f1_weighted: 61.43%, f1_macro: 61.43%,<br>prec_weighted: 61.75%, prec_macro: 61.75%, rec_weighted: 61.58%, rec_macro: 61.58%** | | | | | | | |

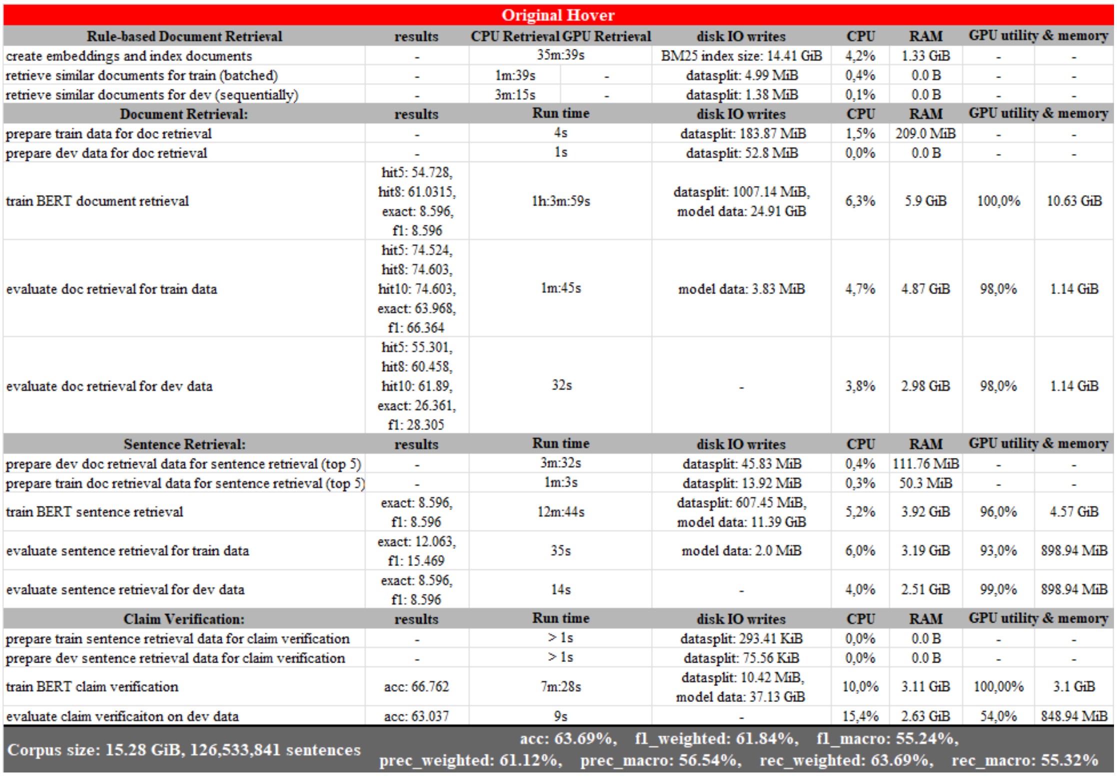
Figure A.14: JPQ retrieval on the fusion (citations + claim-detected) HoVer data corpus.

# Appendix B

# Detailed WiCE Experiments Measurements

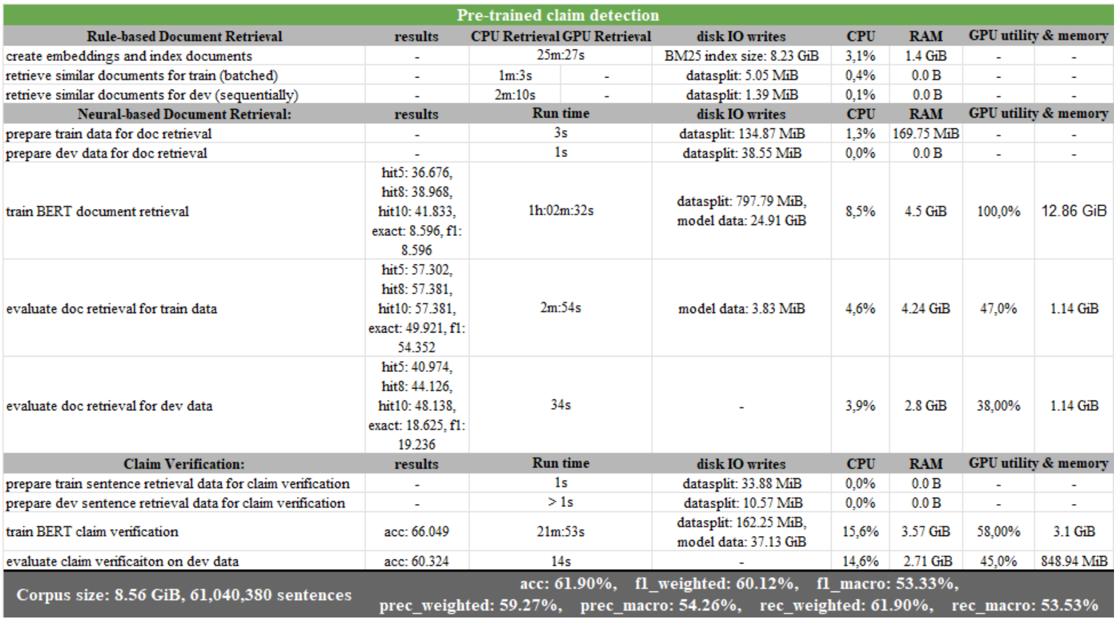
Similar section structure as Appendix A, however the experiments here are conducted on the WiCE dataset instead of the HoVer dataset.

## B.1 Sparse retrieval setup with Reranking



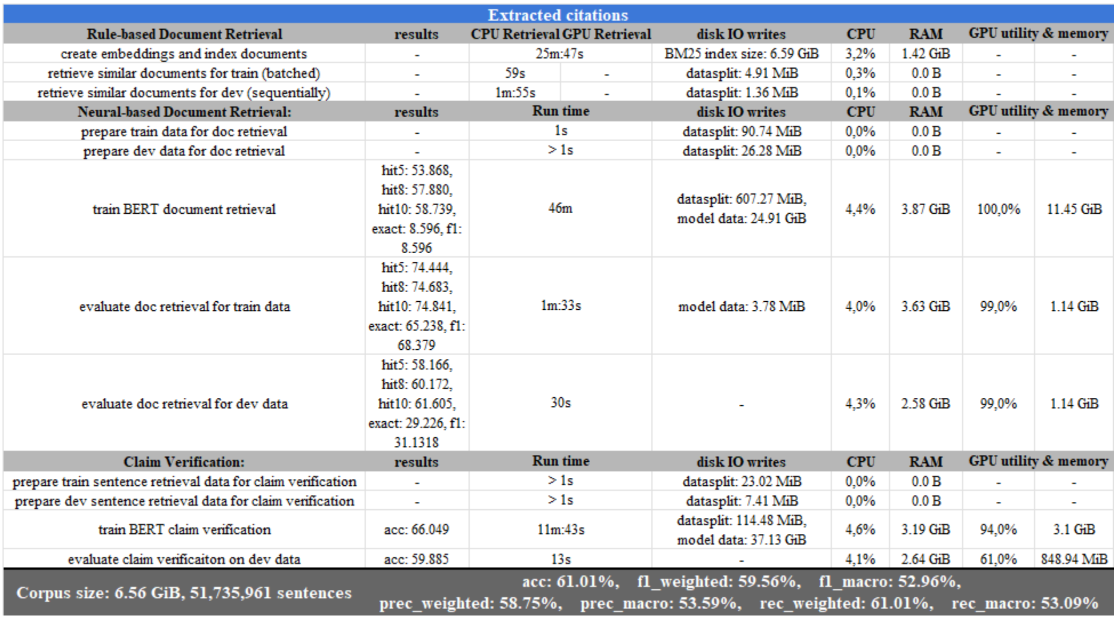
| Original Hover | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 35m:39s | | BM25 index size: 14.41 GiB | 4,2% | 1.33 GiB | - | - |
| retrieve similar documents for train (batched) | - | 1m:39s | - | datasplit: 4.99 MiB | 0,4% | 0.0 B | - | - |
| retrieve similar documents for dev (sequentially) | - | 3m:15s | - | datasplit: 1.38 MiB | 0,1% | 0.0 B | - | - |
| **Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 4s | | datasplit: 183.87 MiB | 1,5% | 209.0 MiB | - | - |
| prepare dev data for doc retrieval | - | 1s | | datasplit: 52.8 MiB | 0,0% | 0.0 B | - | - |
| train BERT document retrieval | hit5: 54.728,<br>hit8: 61.0315,<br>exact: 8.596,<br>f1: 8.596 | 1h:3m:59s | | datasplit: 1007.14 MiB,<br>model data: 24.91 GiB | 6,3% | 5.9 GiB | 100,0% | 10.63 GiB |
| evaluate doc retrieval for train data | hit5: 74.524,<br>hit8: 74.603,<br>hit10: 74.603,<br>exact: 63.968,<br>f1: 66.364 | 1m:45s | | model data: 3.83 MiB | 4,7% | 4.87 GiB | 98,0% | 1.14 GiB |
| evaluate doc retrieval for dev data | hit5: 55.301,<br>hit8: 60.458,<br>hit10: 61.89,<br>exact: 26.361,<br>f1: 28.305 | 32s | | - | 3,8% | 2.98 GiB | 98,0% | 1.14 GiB |
| **Sentence Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare dev doc retrieval data for sentence retrieval (top 5) | - | 3m:32s | | datasplit: 45.83 MiB | 0,4% | 111.76 MiB | - | - |
| prepare train doc retrieval data for sentence retrieval (top 5) | - | 1m:3s | | datasplit: 13.92 MiB | 0,3% | 50.3 MiB | - | - |
| train BERT sentence retrieval | exact: 8.596,<br>f1: 8.596 | 12m:44s | | datasplit: 607.45 MiB,<br>model data: 11.39 GiB | 5,2% | 3.92 GiB | 96,0% | 4.57 GiB |
| evaluate sentence retrieval for train data | exact: 12.063,<br>f1: 15.469 | 35s | | model data: 2.0 MiB | 6,0% | 3.19 GiB | 93,0% | 898.94 MiB |
| evaluate sentence retrieval for dev data | exact: 8.596,<br>f1: 8.596 | 14s | | - | 4,0% | 2.51 GiB | 99,0% | 898.94 MiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | > 1s | | datasplit: 293.41 KiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 75.56 KiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 66.762 | 7m:28s | | datasplit: 10.42 MiB,<br>model data: 37.13 GiB | 10,0% | 3.11 GiB | 100,00% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 63.037 | 9s | | - | 15,4% | 2.63 GiB | 54,0% | 848.94 MiB |
| **Corpus size: 15.28 GiB, 126,533,841 sentences** | **acc: 63.69%, f1_weighted: 61.84%, f1_macro: 55.24%,<br>prec_weighted: 61.12%, prec_macro: 56.54%, rec_weighted: 63.69%, rec_macro: 55.32%** | | | | | | | |

Figure B.1: BM25 retrieval on the original WiCE data corpus.



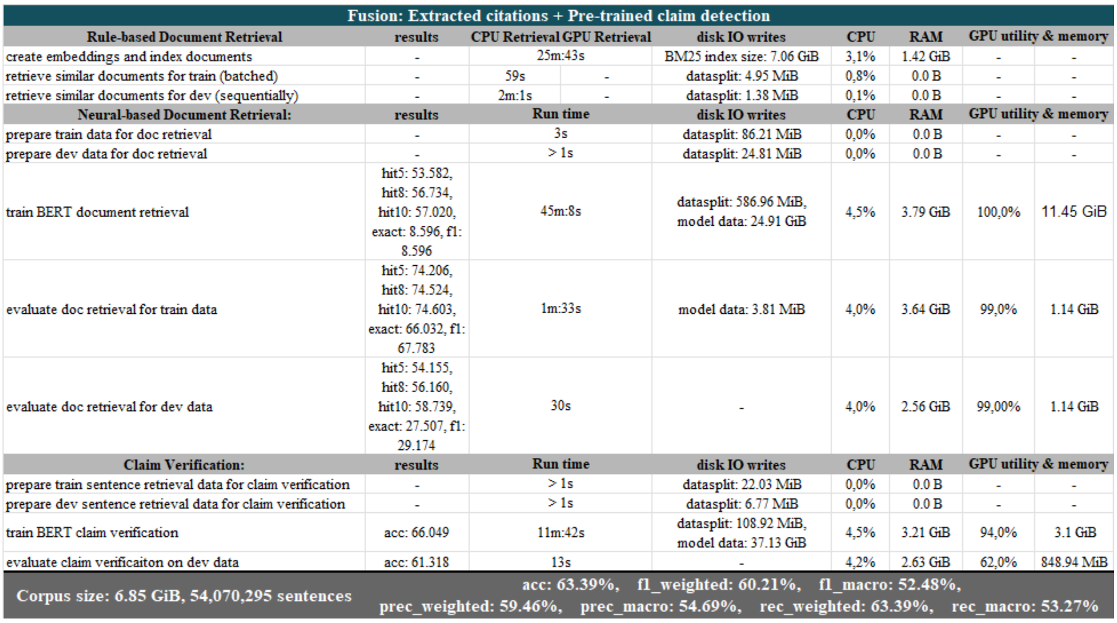
| Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 25m:27s | | BM25 index size: 8.23 GiB | 3,1% | 1.4 GiB | - | - |
| retrieve similar documents for train (batched) | - | 1m:3s | - | datasplit: 5.05 MiB | 0,4% | 0.0 B | - | - |
| retrieve similar documents for dev (sequentially) | - | 2m:10s | - | datasplit: 1.39 MiB | 0,1% | 0.0 B | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 3s | | datasplit: 134.87 MiB | 1,3% | 169.75 MiB | - | - |
| prepare dev data for doc retrieval | - | 1s | | datasplit: 38.55 MiB | 0,0% | 0.0 B | - | - |
| train BERT document retrieval | hit5: 36.676, hit8: 38.968, hit10: 41.833, exact: 8.596, f1: 8.596 | 1h:02m:32s | | datasplit: 797.79 MiB, model data: 24.91 GiB | 8,5% | 4.5 GiB | 100,0% | 12.86 GiB |
| evaluate doc retrieval for train data | hit5: 57.302, hit8: 57.381, hit10: 57.381, exact: 49.921, f1: 54.352 | 2m:54s | | model data: 3.83 MiB | 4,6% | 4.24 GiB | 47,0% | 1.14 GiB |
| evaluate doc retrieval for dev data | hit5: 40.974, hit8: 44.126, hit10: 48.138, exact: 18.625, f1: 19.236 | 34s | | - | 3,9% | 2.8 GiB | 38,00% | 1.14 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | 1s | | datasplit: 33.88 MiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 10.57 MiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 66.049 | 21m:53s | | datasplit: 162.25 MiB, model data: 37.13 GiB | 15,6% | 3.57 GiB | 58,00% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.324 | 14s | | - | 14,6% | 2.71 GiB | 45,0% | 848.94 MiB |
| **Corpus size: 8.56 GiB, 61,040,380 sentences** | **acc: 61.90%, f1_weighted: 60.12%, f1_macro: 53.33%, prec_weighted: 59.27%, prec_macro: 54.26%, rec_weighted: 61.90%, rec_macro: 53.53%** | | | | | | | |

Figure B.2: BM25 retrieval on the claim-detected WiCE data corpus.



| Extracted citations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 25m:47s | | BM25 index size: 6.59 GiB | 3,2% | 1.42 GiB | - | - |
| retrieve similar documents for train (batched) | - | 59s | - | datasplit: 4.91 MiB | 0,3% | 0.0 B | - | - |
| retrieve similar documents for dev (sequentially) | - | 1m:55s | - | datasplit: 1.36 MiB | 0,1% | 0.0 B | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 1s | | datasplit: 90.74 MiB | 0,0% | 0.0 B | - | - |
| prepare dev data for doc retrieval | - | > 1s | | datasplit: 26.28 MiB | 0,0% | 0.0 B | - | - |
| train BERT document retrieval | hit5: 53.868, hit8: 57.880, hit10: 58.739, exact: 8.596, f1: 8.596 | 46m | | datasplit: 607.27 MiB, model data: 24.91 GiB | 4,4% | 3.87 GiB | 100,0% | 11.45 GiB |
| evaluate doc retrieval for train data | hit5: 74.444, hit8: 74.683, hit10: 74.841, exact: 65.238, f1: 68.379 | 1m:33s | | model data: 3.78 MiB | 4,0% | 3.63 GiB | 99,0% | 1.14 GiB |
| evaluate doc retrieval for dev data | hit5: 58.166, hit8: 60.172, hit10: 61.605, exact: 29.226, f1: 31.1318 | 30s | | - | 4,3% | 2.58 GiB | 99,0% | 1.14 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | > 1s | | datasplit: 23.02 MiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 7.41 MiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 66.049 | 11m:43s | | datasplit: 114.48 MiB, model data: 37.13 GiB | 4,6% | 3.19 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 59.885 | 13s | | - | 4,1% | 2.64 GiB | 61,0% | 848.94 MiB |
| **Corpus size: 6.56 GiB, 51,735,961 sentences** | **acc: 61.01%, f1_weighted: 59.56%, f1_macro: 52.96%, prec_weighted: 58.75%, prec_macro: 53.59%, rec_weighted: 61.01%, rec_macro: 53.09%** | | | | | | | |

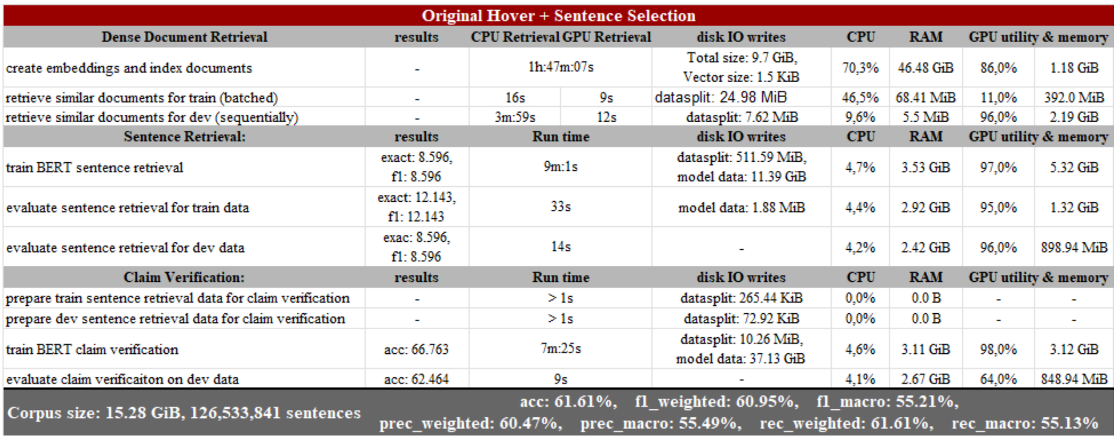
Figure B.3: BM25 retrieval on citation extracted WiCE data corpus.



| Fusion: Extracted citations + Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Rule-based Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 25m:43s | | BM25 index size: 7.06 GiB | 3,1% | 1.42 GiB | - | - |
| retrieve similar documents for train (batched) | - | 59s | - | datasplit: 4.95 MiB | 0,8% | 0.0 B | - | - |
| retrieve similar documents for dev (sequentially) | - | 2m:1s | - | datasplit: 1.38 MiB | 0,1% | 0.0 B | - | - |
| **Neural-based Document Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train data for doc retrieval | - | 3s | | datasplit: 86.21 MiB | 0,0% | 0.0 B | - | - |
| prepare dev data for doc retrieval | - | > 1s | | datasplit: 24.81 MiB | 0,0% | 0.0 B | - | - |
| train BERT document retrieval | hit5: 53.582, hit8: 56.734, hit10: 57.020, exact: 8.596, f1: 8.596 | 45m:8s | | datasplit: 586.96 MiB, model data: 24.91 GiB | 4,5% | 3.79 GiB | 100,0% | 11.45 GiB |
| evaluate doc retrieval for train data | hit5: 74.206, hit8: 74.524, hit10: 74.603, exact: 66.032, f1: 67.783 | 1m:33s | | model data: 3.81 MiB | 4,0% | 3.64 GiB | 99,0% | 1.14 GiB |
| evaluate doc retrieval for dev data | hit5: 54.155, hit8: 56.160, hit10: 58.739, exact: 27.507, f1: 29.174 | 30s | | - | 4,0% | 2.56 GiB | 99,00% | 1.14 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | > 1s | | datasplit: 22.03 MiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 6.77 MiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 66.049 | 11m:42s | | datasplit: 108.92 MiB, model data: 37.13 GiB | 4,5% | 3.21 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 61.318 | 13s | | - | 4,2% | 2.63 GiB | 62,0% | 848.94 MiB |
| **Corpus size: 6.85 GiB, 54,070,295 sentences** | **acc: 63.39%, f1_weighted: 60.21%, f1_macro: 52.48%, prec_weighted: 59.46%, prec_macro: 54.69%, rec_weighted: 63.39%, rec_macro: 53.27%** | | | | | | | |

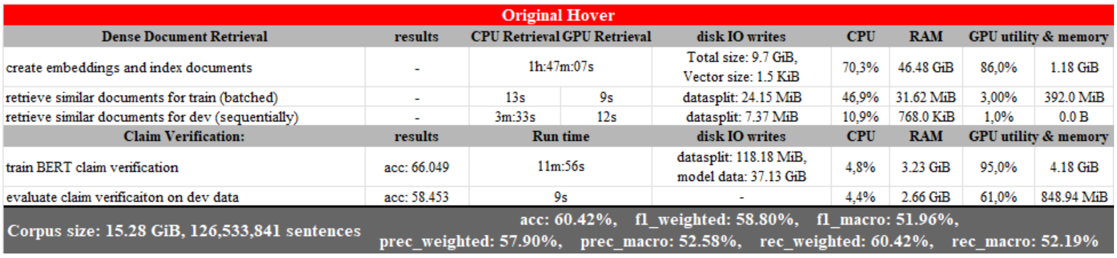
Figure B.4: BM25 retrieval on the fusion (citations + claim-detected) WiCE data corpus.

## B.2 Dense retrieval setup



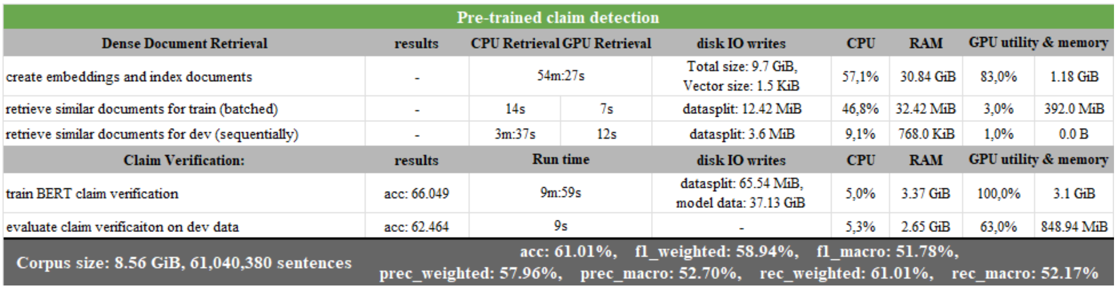
| Original Hover + Sentence Selection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Dense Document Retrieval | results | CPU Retrieval | GPU Retrieval | disk IO writes | CPU | RAM | GPU utility & memory | |
| create embeddings and index documents | - | 1h:47m:07s | | Total size: 9.7 GiB,<br>Vector size: 1.5 KiB | 70,3% | 46.48 GiB | 86,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 16s | 9s | datasplit: 24.98 MiB | 46,5% | 68.41 MiB | 11,0% | 392.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 3m:59s | 12s | datasplit: 7.62 MiB | 9,6% | 5.5 MiB | 96,0% | 2.19 GiB |
| Sentence Retrieval: | results | Run time | | disk IO writes | CPU | RAM | GPU utility & memory | |
| train BERT sentence retrieval | exact: 8.596,<br>f1: 8.596 | 9m:1s | | datasplit: 511.59 MiB,<br>model data: 11.39 GiB | 4,7% | 3.53 GiB | 97,0% | 5.32 GiB |
| evaluate sentence retrieval for train data | exact: 12.143,<br>f1: 12.143 | 33s | | model data: 1.88 MiB | 4,4% | 2.92 GiB | 95,0% | 1.32 GiB |
| evaluate sentence retrieval for dev data | exac: 8.596,<br>f1: 8.596 | 14s | | - | 4,2% | 2.42 GiB | 96,0% | 898.94 MiB |
| Claim Verification: | results | Run time | | disk IO writes | CPU | RAM | GPU utility & memory | |
| prepare train sentence retrieval data for claim verification | - | > 1s | | datasplit: 265.44 KiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 72.92 KiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 66.763 | 7m:25s | | datasplit: 10.26 MiB,<br>model data: 37.13 GiB | 4,6% | 3.11 GiB | 98,0% | 3.12 GiB |
| evaluate claim verificaiton on dev data | acc: 62.464 | 9s | | - | 4,1% | 2.67 GiB | 64,0% | 848.94 MiB |
| Corpus size: 15.28 GiB, 126,533,841 sentences | acc: 61.61%, f1_weighted: 60.95%, f1_macro: 55.21%,<br>prec_weighted: 60.47%, prec_macro: 55.49%, rec_weighted: 61.61%, rec_macro: 55.13% | | | | | | | |

Figure B.5: FAISS retrieval without Sentence-Selection stage on the original WiCE data corpus.



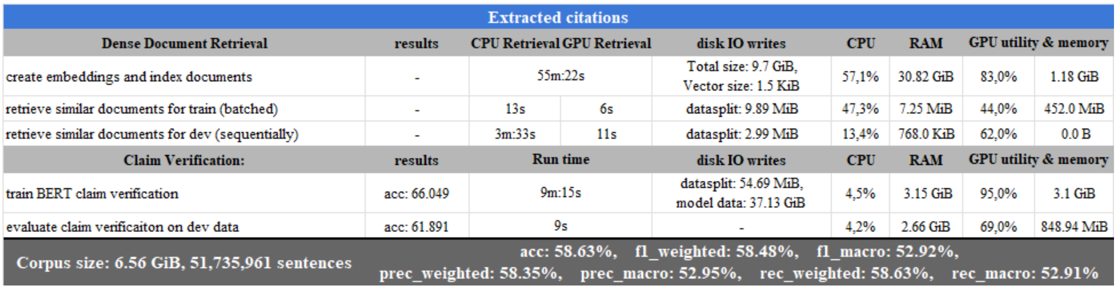
| Original Hover | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Dense Document Retrieval | results | CPU Retrieval | GPU Retrieval | disk IO writes | CPU | RAM | GPU utility & memory | |
| create embeddings and index documents | - | 1h:47m:07s | | Total size: 9.7 GiB,<br>Vector size: 1.5 KiB | 70,3% | 46.48 GiB | 86,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 13s | 9s | datasplit: 24.15 MiB | 46,9% | 31.62 MiB | 3,00% | 392.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 3m:33s | 12s | datasplit: 7.37 MiB | 10,9% | 768.0 KiB | 1,0% | 0.0 B |
| Claim Verification: | results | Run time | | disk IO writes | CPU | RAM | GPU utility & memory | |
| train BERT claim verification | acc: 66.049 | 11m:56s | | datasplit: 118.18 MiB,<br>model data: 37.13 GiB | 4,8% | 3.23 GiB | 95,0% | 4.18 GiB |
| evaluate claim verificaiton on dev data | acc: 58.453 | 9s | | - | 4,4% | 2.66 GiB | 61,0% | 848.94 MiB |
| Corpus size: 15.28 GiB, 126,533,841 sentences | acc: 60.42%, f1_weighted: 58.80%, f1_macro: 51.96%,<br>prec_weighted: 57.90%, prec_macro: 52.58%, rec_weighted: 60.42%, rec_macro: 52.19% | | | | | | | |

Figure B.6: FAISS retrievalon the original WiCE data corpus.



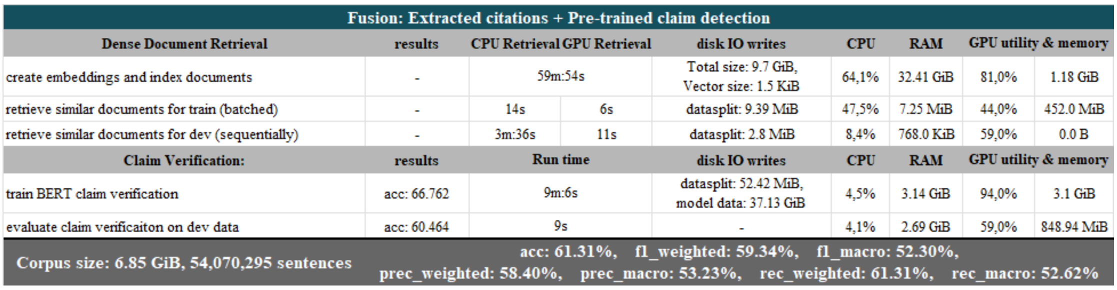
| Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Dense Document Retrieval | results | CPU Retrieval | GPU Retrieval | disk IO writes | CPU | RAM | GPU utility & memory | |
| create embeddings and index documents | - | 54m:27s | | Total size: 9.7 GiB,<br>Vector size: 1.5 KiB | 57,1% | 30.84 GiB | 83,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 14s | 7s | datasplit: 12.42 MiB | 46,8% | 32.42 MiB | 3,0% | 392.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 3m:37s | 12s | datasplit: 3.6 MiB | 9,1% | 768.0 KiB | 1,0% | 0.0 B |
| Claim Verification: | results | Run time | | disk IO writes | CPU | RAM | GPU utility & memory | |
| train BERT claim verification | acc: 66.049 | 9m:59s | | datasplit: 65.54 MiB,<br>model data: 37.13 GiB | 5,0% | 3.37 GiB | 100,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 62.464 | 9s | | - | 5,3% | 2.65 GiB | 63,0% | 848.94 MiB |
| Corpus size: 8.56 GiB, 61,040,380 sentences | acc: 61.01%, f1_weighted: 58.94%, f1_macro: 51.78%,<br>prec_weighted: 57.96%, prec_macro: 52.70%, rec_weighted: 61.01%, rec_macro: 52.17% | | | | | | | |

Figure B.7: FAISS retrieval on the claim-detected WiCE data corpus.



| Extracted citations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 55m:22s | | Total size: 9.7 GiB,<br>Vector size: 1.5 KiB | 57,1% | 30.82 GiB | 83,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 13s | 6s | datasplit: 9.89 MiB | 47,3% | 7.25 MiB | 44,0% | 452.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 3m:33s | 11s | datasplit: 2.99 MiB | 13,4% | 768.0 KiB | 62,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 66.049 | 9m:15s | | datasplit: 54.69 MiB,<br>model data: 37.13 GiB | 4,5% | 3.15 GiB | 95,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 61.891 | 9s | | - | 4,2% | 2.66 GiB | 69,0% | 848.94 MiB |
| **Corpus size: 6.56 GiB, 51,735,961 sentences** | **acc: 58.63%, f1_weighted: 58.48%, f1_macro: 52.92%,<br>prec_weighted: 58.35%, prec_macro: 52.95%, rec_weighted: 58.63%, rec_macro: 52.91%** | | | | | | | |

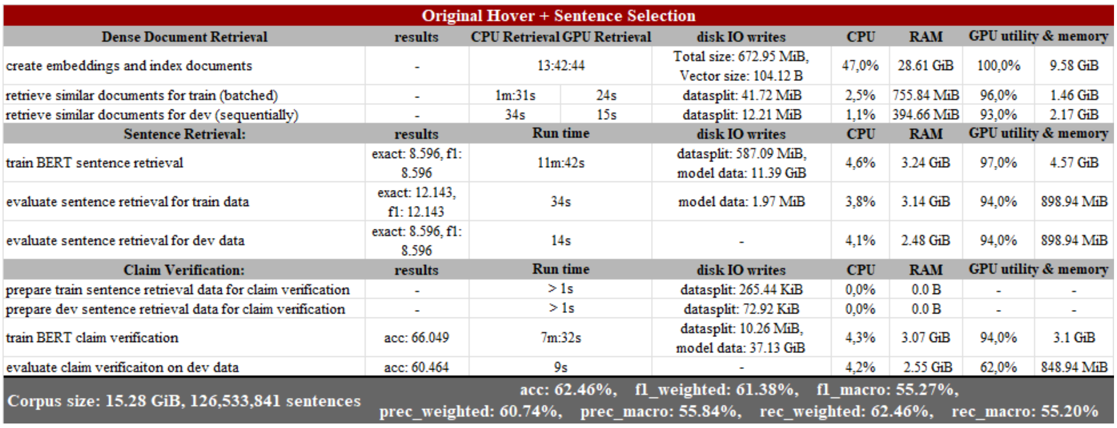
Figure B.8: FAISS retrieval on citation extracted WiCE data corpus.



| Fusion: Extracted citations + Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 59m:54s | | Total size: 9.7 GiB,<br>Vector size: 1.5 KiB | 64,1% | 32.41 GiB | 81,0% | 1.18 GiB |
| retrieve similar documents for train (batched) | - | 14s | 6s | datasplit: 9.39 MiB | 47,5% | 7.25 MiB | 44,0% | 452.0 MiB |
| retrieve similar documents for dev (sequentially) | - | 3m:36s | 11s | datasplit: 2.8 MiB | 8,4% | 768.0 KiB | 59,0% | 0.0 B |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 66.762 | 9m:6s | | datasplit: 52.42 MiB,<br>model data: 37.13 GiB | 4,5% | 3.14 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.464 | 9s | | - | 4,1% | 2.69 GiB | 59,0% | 848.94 MiB |
| **Corpus size: 6.85 GiB, 54,070,295 sentences** | **acc: 61.31%, f1_weighted: 59.34%, f1_macro: 52.30%,<br>prec_weighted: 58.40%, prec_macro: 53.23%, rec_weighted: 61.31%, rec_macro: 52.62%** | | | | | | | |

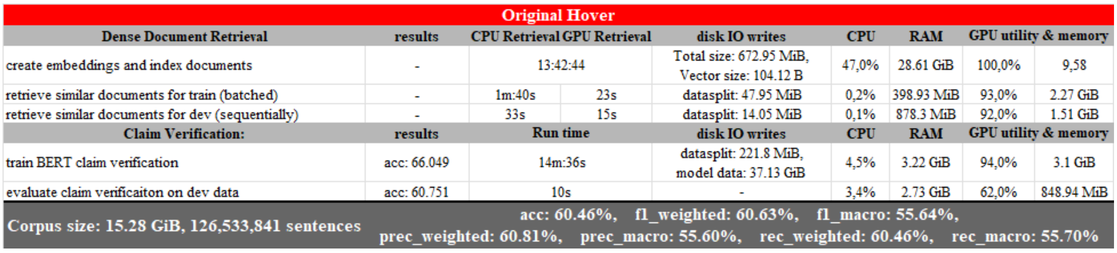
Figure B.9: FAISS retrieval on the fusion (citations + claim-detected) WiCE data corpus.

## B.3 Index compression setup



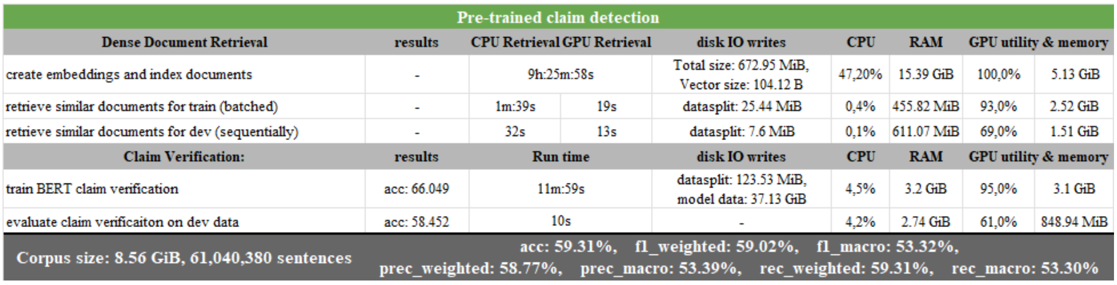
| Original Hover + Sentence Selection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 13:42:44 | | Total size: 672.95 MiB,<br>Vector size: 104.12 B | 47,0% | 28.61 GiB | 100,0% | 9.58 GiB |
| retrieve similar documents for train (batched) | - | 1m:31s | 24s | datasplit: 41.72 MiB | 2,5% | 755.84 MiB | 96,0% | 1.46 GiB |
| retrieve similar documents for dev (sequentially) | - | 34s | 15s | datasplit: 12.21 MiB | 1,1% | 394.66 MiB | 93,0% | 2.17 GiB |
| **Sentence Retrieval:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT sentence retrieval | exact: 8.596, f1: 8.596 | 11m:42s | | datasplit: 587.09 MiB,<br>model data: 11.39 GiB | 4,6% | 3.24 GiB | 97,0% | 4.57 GiB |
| evaluate sentence retrieval for train data | exact: 12.143, f1: 12.143 | 34s | | model data: 1.97 MiB | 3,8% | 3.14 GiB | 94,0% | 898.94 MiB |
| evaluate sentence retrieval for dev data | exact: 8.596, f1: 8.596 | 14s | | - | 4,1% | 2.48 GiB | 94,0% | 898.94 MiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| prepare train sentence retrieval data for claim verification | - | > 1s | | datasplit: 265.44 KiB | 0,0% | 0.0 B | - | - |
| prepare dev sentence retrieval data for claim verification | - | > 1s | | datasplit: 72.92 KiB | 0,0% | 0.0 B | - | - |
| train BERT claim verification | acc: 66.049 | 7m:32s | | datasplit: 10.26 MiB,<br>model data: 37.13 GiB | 4,3% | 3.07 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.464 | 9s | | - | 4,2% | 2.55 GiB | 62,0% | 848.94 MiB |
| **Corpus size: 15.28 GiB, 126,533,841 sentences** | **acc: 62.46%, f1_weighted: 61.38%, f1_macro: 55.27%,<br>prec_weighted: 60.74%, prec_macro: 55.84%, rec_weighted: 62.46%, rec_macro: 55.20%** | | | | | | | |

Figure B.10: JPQ retrieval without Sentence-Selection stage on the original WiCE data corpus.



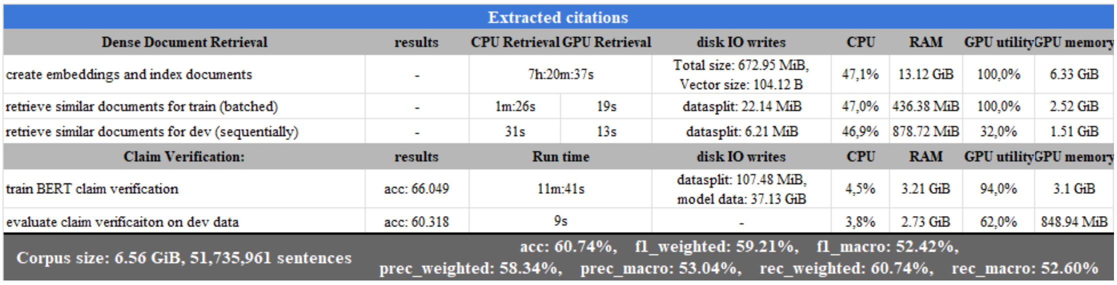
| Original Hover | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 13:42:44 | | Total size: 672.95 MiB,<br>Vector size: 104.12 B | 47,0% | 28.61 GiB | 100,0% | 9,58 |
| retrieve similar documents for train (batched) | - | 1m:40s | 23s | datasplit: 47.95 MiB | 0,2% | 398.93 MiB | 93,0% | 2.27 GiB |
| retrieve similar documents for dev (sequentially) | - | 33s | 15s | datasplit: 14.05 MiB | 0,1% | 878.3 MiB | 92,0% | 1.51 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 66.049 | 14m:36s | | datasplit: 221.8 MiB,<br>model data: 37.13 GiB | 4,5% | 3.22 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.751 | 10s | | - | 3,4% | 2.73 GiB | 62,0% | 848.94 MiB |
| **Corpus size: 15.28 GiB, 126,533,841 sentences** | **acc: 60.46%, f1_weighted: 60.63%, f1_macro: 55.64%,<br>prec_weighted: 60.81%, prec_macro: 55.60%, rec_weighted: 60.46%, rec_macro: 55.70%** | | | | | | | |

Figure B.11: JPQ retrieval on the original WiCE data corpus.



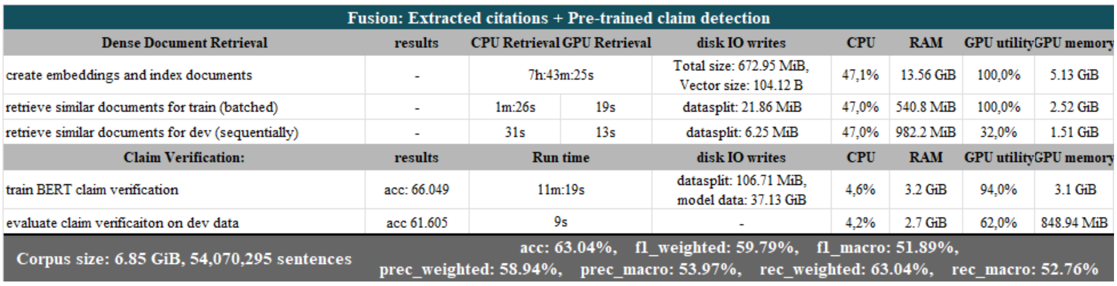
| Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| create embeddings and index documents | - | 9h:25m:58s | | Total size: 672.95 MiB,<br>Vector size: 104.12 B | 47,20% | 15.39 GiB | 100,0% | 5.13 GiB |
| retrieve similar documents for train (batched) | - | 1m:39s | 19s | datasplit: 25.44 MiB | 0,4% | 455.82 MiB | 93,0% | 2.52 GiB |
| retrieve similar documents for dev (sequentially) | - | 32s | 13s | datasplit: 7.6 MiB | 0,1% | 611.07 MiB | 69,0% | 1.51 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility & memory** | |
| train BERT claim verification | acc: 66.049 | 11m:59s | | datasplit: 123.53 MiB,<br>model data: 37.13 GiB | 4,5% | 3.2 GiB | 95,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 58.452 | 10s | | - | 4,2% | 2.74 GiB | 61,0% | 848.94 MiB |
| **Corpus size: 8.56 GiB, 61,040,380 sentences** | **acc: 59.31%, f1_weighted: 59.02%, f1_macro: 53.32%,<br>prec_weighted: 58.77%, prec_macro: 53.39%, rec_weighted: 59.31%, rec_macro: 53.30%** | | | | | | | |

Figure B.12: JPQ retrieval on the claim-detected WiCE data corpus.



| Extracted citations | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility** | **GPU memory** |
| create embeddings and index documents | - | 7h:20m:37s | | Total size: 672.95 MiB,<br>Vector size: 104.12 B | 47,1% | 13.12 GiB | 100,0% | 6.33 GiB |
| retrieve similar documents for train (batched) | - | 1m:26s | 19s | datasplit: 22.14 MiB | 47,0% | 436.38 MiB | 100,0% | 2.52 GiB |
| retrieve similar documents for dev (sequentially) | - | 31s | 13s | datasplit: 6.21 MiB | 46,9% | 878.72 MiB | 32,0% | 1.51 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility** | **GPU memory** |
| train BERT claim verification | acc: 66.049 | 11m:41s | | datasplit: 107.48 MiB,<br>model data: 37.13 GiB | 4,5% | 3.21 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc: 60.318 | 9s | | - | 3,8% | 2.73 GiB | 62,0% | 848.94 MiB |
| **Corpus size: 6.56 GiB, 51,735,961 sentences** | **acc: 60.74%, f1_weighted: 59.21%, f1_macro: 52.42%,<br>prec_weighted: 58.34%, prec_macro: 53.04%, rec_weighted: 60.74%, rec_macro: 52.60%** | | | | | | | |

Figure B.13: JPQ retrieval on citation extracted WiCE data corpus.



| Fusion: Extracted citations + Pre-trained claim detection | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Dense Document Retrieval** | **results** | **CPU Retrieval** | **GPU Retrieval** | **disk IO writes** | **CPU** | **RAM** | **GPU utility** | **GPU memory** |
| create embeddings and index documents | - | 7h:43m:25s | | Total size: 672.95 MiB,<br>Vector size: 104.12 B | 47,1% | 13.56 GiB | 100,0% | 5.13 GiB |
| retrieve similar documents for train (batched) | - | 1m:26s | 19s | datasplit: 21.86 MiB | 47,0% | 540.8 MiB | 100,0% | 2.52 GiB |
| retrieve similar documents for dev (sequentially) | - | 31s | 13s | datasplit: 6.25 MiB | 47,0% | 982.2 MiB | 32,0% | 1.51 GiB |
| **Claim Verification:** | **results** | **Run time** | | **disk IO writes** | **CPU** | **RAM** | **GPU utility** | **GPU memory** |
| train BERT claim verification | acc: 66.049 | 11m:19s | | datasplit: 106.71 MiB,<br>model data: 37.13 GiB | 4,6% | 3.2 GiB | 94,0% | 3.1 GiB |
| evaluate claim verificaiton on dev data | acc 61.605 | 9s | | - | 4,2% | 2.7 GiB | 62,0% | 848.94 MiB |
| **Corpus size: 6.85 GiB, 54,070,295 sentences** | **acc: 63.04%, f1_weighted: 59.79%, f1_macro: 51.89%,<br>prec_weighted: 58.94%, prec_macro: 53.97%, rec_weighted: 63.04%, rec_macro: 52.76%** | | | | | | | |

Figure B.14: JPQ retrieval on the fusion (citations + claim-detected) WiCE data corpus.

# Appendix C

# Experiment Results Graphs

This chapter illustrates part of the tabular data results detailed in Chapter 5 into a simplified perspective by employing graph visualization. This with the aim to offer a clearer understanding of the comparisons between various settings.

## C.1 Document Retrieval Latency

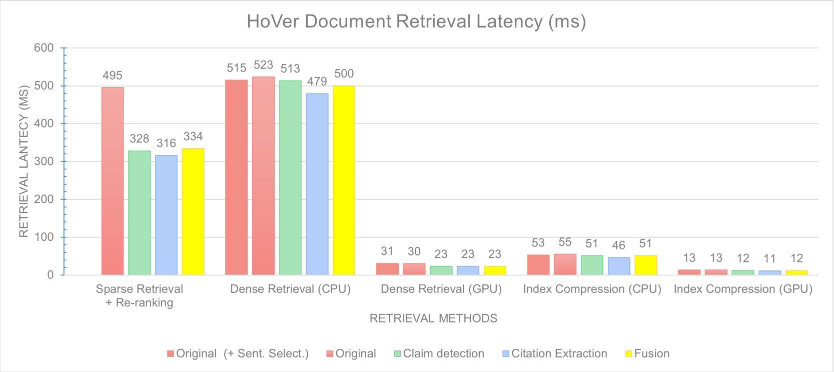


Figure C.1: Document retrieval latency comparison between all the HoVer experiments.

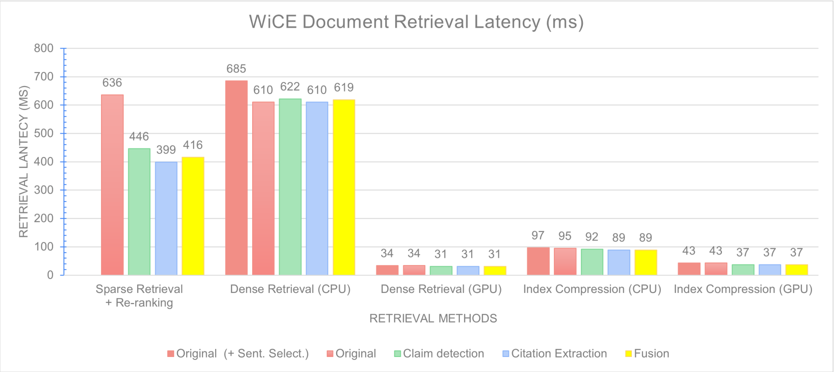


Figure C.2: Document retrieval latency comparison between all the WiCE experiments.

## C.2 Pipeline Inference Latency

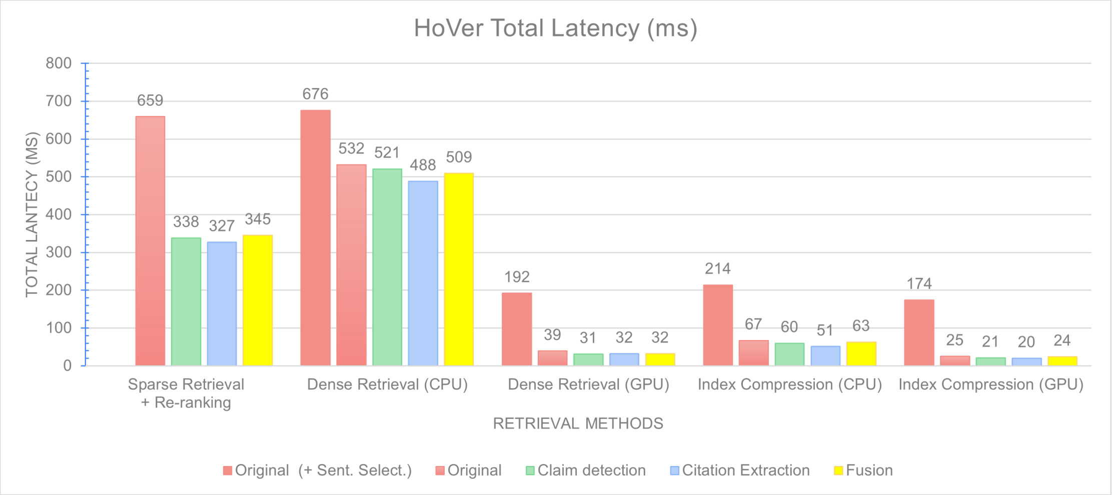


Figure C.3: Overall pipeline latency comparison between all the HoVer experiments.

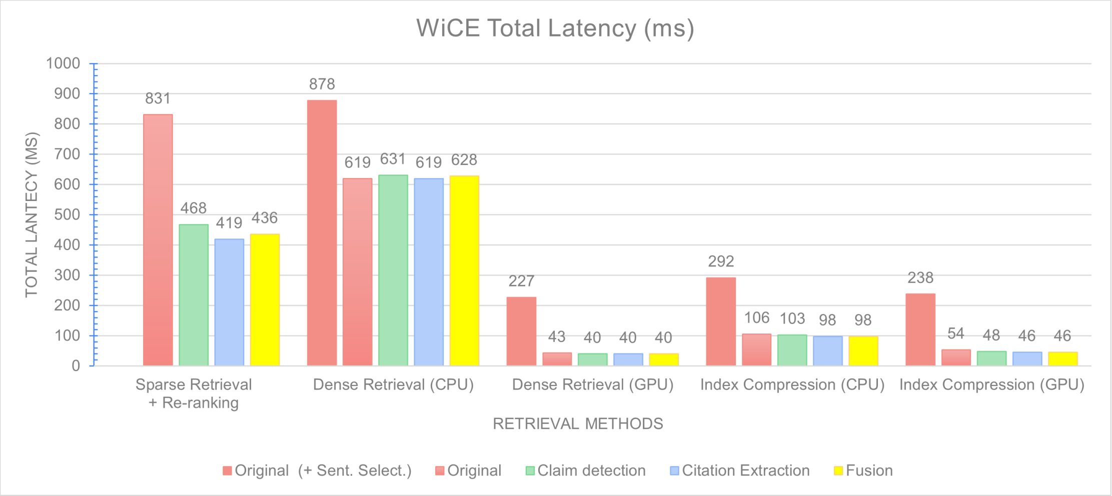


Figure C.4: Overall pipeline latency comparison between all the WiCE experiments.

## C.3 Performance Accuracy Evaluation

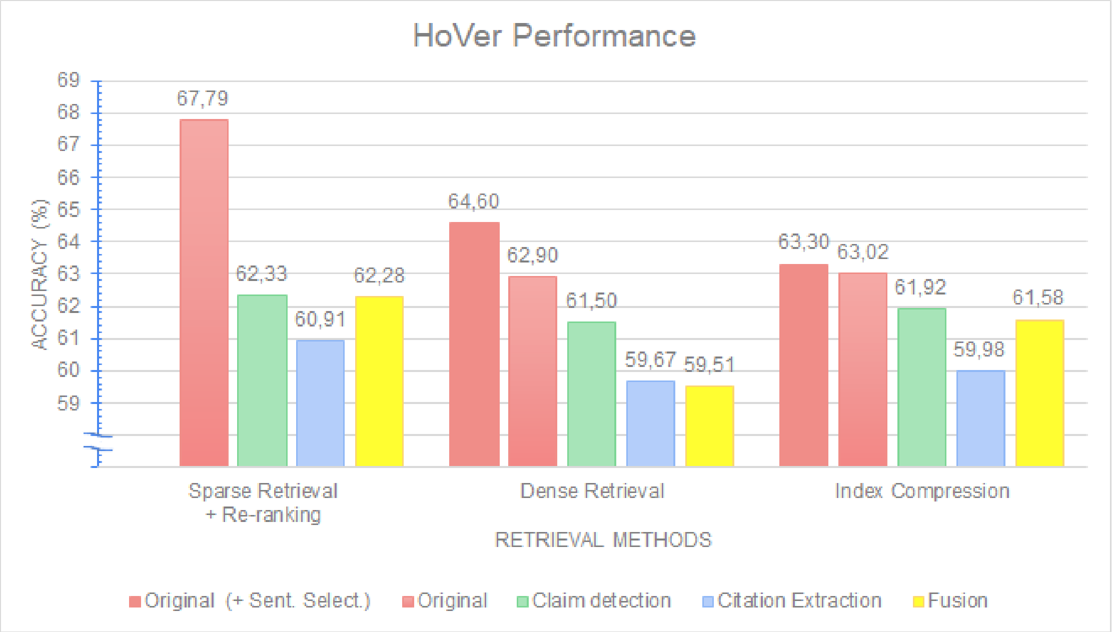


Figure C.5: Accuracy comparison between all the HoVer experiments.

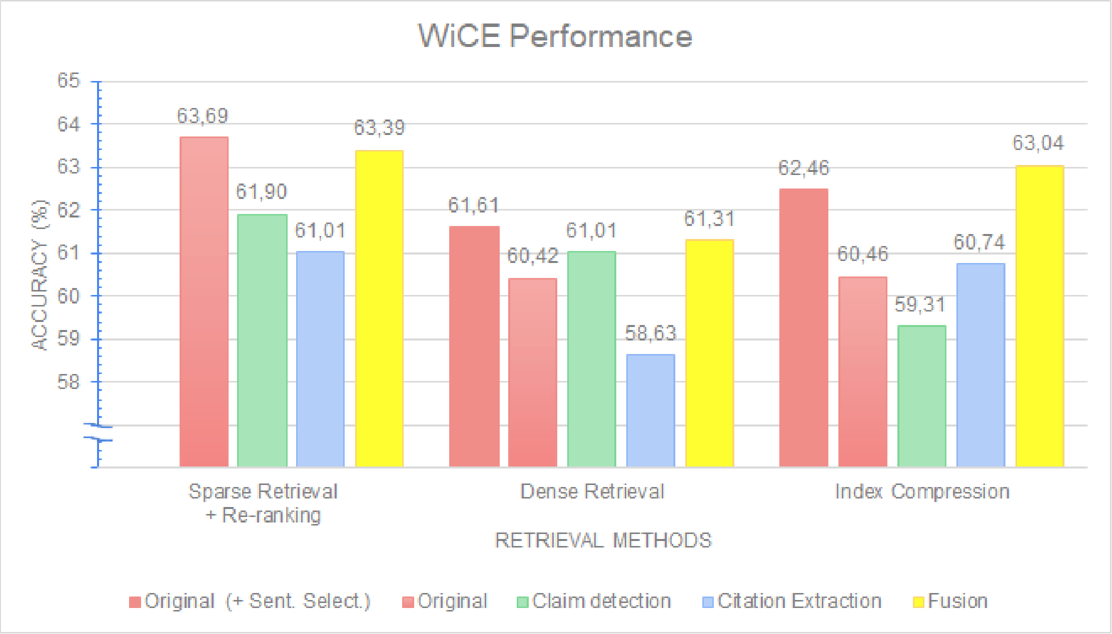


Figure C.6: Accuracy comparison between all the WiCE experiments.