CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Benchmarking Framework - Evaluating Corpora Search Systems Using a Spectrum of Complex Search Queries

Master's thesis in Computer science and engineering

ADELL TATROUS

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Benchmarking Framework - Evaluating Corpora Search Systems Using a Spectrum of Complex Search Queries

ADELL TATROUS

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Benchmarking Framework - Evaluating Corpora Search Systems Using a Spectrum of Complex Search Queries
ADELL TATROUS

Supervisor: Peter Ljunglöf, Department of Computer Science and Engineering & Department of Swedish, Multilingualism, and Language Technology
Examiner: Alejandro Russo, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2025

Benchmarking Framework - Evaluating Corpora Search Systems Using a Spectrum of Complex Search Queries
ADELL TATROUS
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis presents a benchmark designed to evaluate corpora search systems in digital humanities, focusing on handling complex search queries. By analyzing user-generated queries from Korp server logs, the research examines various query structures and computational challenges. The study develops a two-level abstraction process to simplify query patterns, enabling clustering and feature extraction techniques. The Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm is used to segment queries into clusters representing different user behaviors. The performance of these clusters is evaluated by executing queries on selected corpora using Corpus Workbench (CWB), with execution times as the main metric. The results indicate variability in query execution efficiency, influenced by corpus size and query complexity. To improve benchmark clarity, clusters with similar structures were grouped to create a more focused evaluation framework. The findings show how query complexity and data volume affect system performance, highlighting the importance of scalability in corpora search systems. This benchmark provides a framework for evaluating the performance of corpora search systems in digital humanities, contributing to the development of digital research tools in this field.

Keywords: Digital Humanities, Corpora Search Systems, Benchmark, Korp, Corpus Workbench, HDBSCAN Clustering.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Peter Ljunglöf, whose expertise, guidance, and unwavering support were pivotal throughout the entirety of this project. Your insightful feedback significantly contributed to the successful completion of this work.

I would also like to extend my heartfelt thanks to everyone else who supported me along this journey. This includes my teachers, colleagues, friends, and family, whose advice, assistance, and encouragement have been invaluable. Your collective support, whether through practical help or moral encouragement, has made a significant difference, and for that, I am truly grateful.

Thank you all for your invaluable contributions.

Adell Tatrous, Gothenburg, 2025-06-11

Contents

List of Figures

List of Tables

List of Acronyms

CPU	Central Processing Unit.
CQP	Corpus Query Processor.
CV	Coefficient of Variation.
CWB	Corpus Workbench.
DBCV	Density-Based Clustering Validation.
DBSCAN	Density-Based Spatial Clustering of Applications with Noise.
GLOSH	Global-Local Outlier Scores from Hierarchies.
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise.
IQR	Interquartile Range.
MAD	Median Absolute Deviation.
ML	Machine Learning.
NLP	Natural Language Processing.
regex	Regular Expression.
SD	Standard Deviation.
UD	Universal Dependencies.
VRT	VeRticalized Text.
WSL	Windows Subsystem for Linux.
XML	Extensible Markup Language.

1

Introduction

The field of digital humanities demonstrates the impact of digital technology on traditional humanities disciplines [1]. Central to this field is the use of annotated corpora—searchable collections of digital texts that have become important tools for researchers. These corpora allow for the exploration of linguistic patterns, historical narratives, and cultural insights.

The variety of corpora search systems, each with its own implementation, highlights the need for a benchmark to evaluate these systems' performance in handling complex search queries. This benchmark should focus on the nuances of query formulation and their computational implications, covering a range of query complexities to assess how well these systems manage linguistic and computational challenges.

1.1 Related Work

The development of a benchmark for evaluating the performance of corpora search systems is a key issue in digital humanities and computational linguistics. As researchers increasingly rely on digital resources, the need for advanced search methodologies becomes more evident [2]. Despite numerous studies on search methodologies and corpus analysis, a gap remains in establishing a comprehensive benchmark that evaluates corpora search systems across diverse implementations, particularly in their ability to handle the linguistic and computational demands inherent in humanities research.

Research has shown the influence of query formulation on the performance of search systems [3]. This finding underscores the need for benchmarks that can assess how effectively different systems manage various complexities in query formulation and execution speed. However, the application of these insights specifically to annotated corpora within digital humanities research is limited, revealing a gap in current evaluation practices.

Interdisciplinary challenges in digital humanities projects indicate that while computational tools are increasingly integrated into humanities research, there remains a lack of standardized evaluation metrics that address the specific needs of humanities scholars [4]. This report emphasizes the importance of computational tools that support the interdisciplinary nature of digital humanities. It discusses the role of advanced search systems in processing large data sets, opening new research pos-

sibilities, and fostering collaboration between computer scientists and humanities scholars to bridge methodological gaps. This is important for achieving accurate and meaningful search results in digital humanities contexts.

In the realm of big data and web search engines, significant progress has been made in developing benchmarks for scalable data processing [5]. These benchmarks focus on generating large volumes of data while preserving semantics and locality. However, they do not address the specific challenges faced by digital humanities researchers, such as the need for detailed linguistic annotations and the efficient handling of diverse and complex corpora data sets.

There is a strong case for more tailored evaluation approaches to bridge the gap between Natural Language Processing (NLP) and digital humanities [6]. The lack of communication and diverging goals between the NLP and digital humanities communities hinders the development of effective benchmarks that meet the specific needs of humanities scholars. This gap further underscores the need for a benchmark specifically designed for corpora search systems in digital humanities.

While there have been advances in benchmarking for various digital and computational fields, there is a clear need for a benchmark tailored to the unique requirements of corpora search systems in digital humanities. Such a benchmark should facilitate comparisons across different systems by addressing the complexities of query formulation, execution speed, and performance evaluation.

1.2 Aim

This thesis aims to develop a benchmark tailored for evaluating corpora search systems in digital humanities. The benchmark is intended to facilitate comparisons among different corpora search systems, including implementations of the same system, by encompassing a broad range of complex search queries. This contribution is expected to improve the understanding of the efficacy of search technology within the field.

1.3 Objectives

The objectives of this thesis are as follows:

1. **Processing and Analyzing Korp Server Logs:** Obtain, parse, clean, and analyze the server logs from Korp, Språkbanken (The Swedish Language Bank)s concordance search tool at the University of Gothenburg [7], to extract query events; their accessibility and volume ensure a comprehensive dataset.
2. **Query Categorization:** This involves categorizing queries by their structural complexity to establish a consistent framework for benchmarking across different search contexts.
3. **Execution of Categorized Queries on Selected Corpora:** This step includes testing the categorized queries on selected corpora from Språkbanken

using Corpus Workbench (CWB) [8], which is the backbone of Korp, evaluating query execution times, and documenting performance across different corpora.

4. **Benchmark Development:** Establish a benchmark that uses query execution times as a metric for evaluating the performance of corpora search systems. This process involves selecting representative queries from each category based on selected evaluation criteria, such as execution success rates and variability in execution times, to ensure that the benchmark includes queries that provide a consistent basis for comparing system performance.

1.4 Delimitations

This thesis acknowledges several limitations that may impact its broader applicability and generalizability. Firstly, the reliance on Korp server logs for query analysis may limit the scope of findings to systems that use Corpus Query Processor (CQP) as a query language [8]. This specialization might affect the benchmark's effectiveness in broader contexts, particularly in digital humanities research involving different languages or textual corpora. Additionally, the diversity of digital humanities queries and corpora search systems presents challenges in standardizing performance metrics. Despite efforts to ensure that these metrics are equitable, the variability in system architectures could lead to discrepancies in evaluations. Lastly, the focus of this thesis on linguistic and computational criteria excludes accuracy and user experience considerations, which are important for the practical adoption of corpora search systems in humanities research.

As the field evolves, the benchmark will require updates to remain relevant and useful in evaluating the performance of corpora search systems.

2

Theory

Natural Language Processing (NLP) techniques, rooted in both linguistics and computer science, are used for automating the understanding and manipulation of human language. NLP involves a set of algorithms that translate text into a form that computers can understand and process. As a component of modern computational linguistics toolsets, NLP is important for the development of technologies that use language data. The effectiveness of NLP depends on the quality and characteristics of the data it processes, often derived from annotated corpora. These corpora serve as training and validation sets that support the development of models capable of performing language understanding tasks, contributing to advancements in the field [9].

Building on this foundation, the field of computational linguistics has changed with the integration of analytical tools and data management platforms such as CWB and its core component CQP [8], as well as the corpus infrastructure developed by Språkbanken, known as Korp [7]. These innovations support the examination and exploration of linguistic data, improving the management and querying of large-scale annotated corpora and influencing approaches to linguistic research.

In parallel with these technological developments, Machine Learning (ML) has become increasingly important, improving our ability to manage and interpret large volumes of data. The use of ML algorithms allows researchers to utilize structured data, often referred to as ML features, for data categorization and model training. These models help identify patterns and make predictions [10]. The Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm is particularly suited to analyze large unsorted datasets. Its clustering method identifies structures and connections within data, offering a useful approach for exploring patterns that may inform various types of research, including applications in computational linguistics [11].

Understanding variability within and between datasets is important in statistical analysis, with measures such as the Standard Deviation (SD) and the Coefficient of Variation (CV) playing a significant role. SD measures the spread of values within a single dataset, while CV provides a standardized way to assess relative variability, which is especially useful for comparing data across different scales. These measures are used to compare group means and make reliable statistical conclusions [12], [13].

This chapter explores these tools and methods, focusing on their impact on digi-

tal humanities and computational linguistics. The following sections will discuss the use of annotated corpora for NLP applications, the role of ML in managing and interpreting complex datasets, and the importance of SD and CV in statistical analysis.

2.1 Annotated Corpora and Corpus Management

This section covers annotated corpora and corpus management tools, which are used in linguistic analysis and NLP applications.

2.1.1 Annotated Corpora

An annotated corpus is a text dataset that includes additional linguistic information, such as part-of-speech tags, syntactic structures, and semantic roles. These annotations can be added either through human effort or automated processes. Annotated corpora are essential for applications in computational linguistics and digital humanities, enabling detailed linguistic analysis and supporting the development and evaluation of NLP models [14].

There are several types of annotated corpora, each designed to capture different linguistic features [14]. The primary types of annotations for written text corpora include:

- **Lemmatization:** Identifies and marks each word with its base (dictionary) form to standardize different word forms.
- **Part-of-Speech Tagging:** Labels each word with its part of speech (e.g., noun, verb, adjective), along with syntactic categories and morphological features.
- **Syntactic Parse Trees:** Represent the hierarchical relationships between words and phrases, illustrating the syntactic structure of a sentence.
- **Semantic Annotation:** Involves identifying and labeling the meanings of words and phrases in context, including word sense disambiguation and thematic roles.

In addition, there are specialized annotations for spoken and multimodal corpora, such as phonetic and phonological annotations, prosodic annotations, sign language and gesture annotations, and interactional annotations [14].

Annotations can be stored in various formats, including inline or embedded annotations, where the annotations are contained within the same file as the primary data; multi-tiered or interlinear annotations, where the data and annotations appear on separate lines within the same file; and standoff or standalone annotations, where the annotations are stored in separate files and linked to the primary data via pointers [14].

For example, consider the following annotation for the sentence “The children sang songs and wrote letters to their grandparents”:

Word	The	children	sang	songs	and	wrote	letters	to	their	grandparents
POS	DT	NNS	VBD	NNS	CC	VBD	NNS	IN	PRP$	NNS
Lemma	the	child	sing	song	and	write	letter	to	their	grandparent

In this example:

- **Word:** Displays the original sentence.
- **POS:** Indicates the part-of-speech tags for each word (DT = Determiner, NNS = Noun, plural, VBD = Verb, past tense, CC = Coordinating Conjunction, IN = Preposition, PRP$ = Possessive Pronoun).
- **Lemma:** Represents the base forms of the words.

Annotated corpora in the digital humanities support linguistic analysis, text mining, and distant reading, enabling scholars to examine linguistic features and trends across large text sets. These tools assist in understanding cultural shifts, historical language use, and thematic developments in literature, enhancing both qualitative and quantitative research in the humanities [15].

2.1.2 Corpus Management Tools and Frameworks

This section discusses Corpus Workbench (CWB), Corpus Query Processor (CQP), and the Korp system developed by Språkbanken. These tools are used for managing and querying large-scale linguistic corpora, representing developments in digital humanities and computational linguistics.

2.1.2.1 Corpus Workbench and Corpus Query Processor

CWB, developed at the University of Stuttgart, along with its core component CQP, is a widely used tool in computational linguistics. CWB provides a framework for analyzing, managing, and querying annotated corpora. This technology allows linguists to perform complex searches and analyses of linguistic data [8].

CWB is recognized for its capacity to manage and query large corpora, efficiently accommodating multiple layers of word-level annotation. Its indexing tools and integration with CQP enable researchers to conduct detailed linguistic inquiries, making it useful for exploring linguistic patterns and phenomena. The architecture's scalability and efficiency have made CWB a commonly used framework in computational linguistics [8].

Central to CWB is CQP, which enhances the corpus analysis framework with a concordance system and a flexible search language. This feature allows for complex query patterns at both the word or annotation level and across sequences of tokens, supporting detailed linguistic analysis. CQP's search language enables researchers to engage with linguistic data in various ways, expanding the scope of computational linguistics research [8].

Using the annotated corpus example from Section 2.1.1, consider the following CQP query:

```
[word = "children"] [pos = "VBD"]
```

This query searches for instances where the word "children" is immediately followed by a verb in the past tense (e.g., "sang"), resulting in phrases like "children sang". The result would be any occurrence of "children" followed by a verb tagged as "VBD" in the annotated corpus, such as in the sentence "The children sang songs and wrote letters to their grandparents".

CQP queries are formulated using a structured notation that allows researchers to specify detailed characteristics of tokens through various attributes and logical operators [16]. Here are the key components of a CQP query:

- **Tokens and Attributes:** Tokens, typically words or punctuation marks, are represented within square brackets `'[ ]'`. For example, the following query defines a token where the word exactly matches "hi":

  ```
  [word = "hi"]
  ```

  This token comprises:

  - *Attribute parameter:* `word` — Specifies the attribute to be queried.
  - *Attribute operator:* `=` — Indicates the exact match condition between the attribute and its value.
  - *Attribute value:* `"hi"` — The specific value that the attribute must match.

- **Annotation Search:** Users can search based on detailed annotations, such as lemma, part of speech, and semantic features, across multiple levels of language structure. For example, the following query specifies a search for the word "run" where the part of speech (POS) tag is "VB" (base form verb) in a corpus query:

  ```
  [word = "run" & pos = "VB"]
  ```

- **Attributes and Operators:** Operators such as `'&'` allow for combining multiple attributes within tokens and enabling complex queries. For example, the following query seeks tokens where the word is "do", its part of speech is a verb, and its morphosyntactic descriptor indicates it is in the infinitive active form:

  ```
  [word = "do" & pos = "VB" & msd = "VB.INF.AKT"]
  ```

- **Positional Attributes:** Queries can also target specific positional attributes to enhance search specificity, such as querying for a specific lemma or part of speech within a specified range of tokens. For example, the following query searches for a verb that comes after one to three consecutive nouns:

  ```
  [pos = "NN"]{1,3} [pos = "VB"]
  ```

- **Boolean Logic in Token Grouping:** In CQP, researchers can use parentheses `'( )'` for token grouping alongside Boolean operators (`'&'`, `'|'`, `'!'` representing AND, OR, NOT) to craft sophisticated linguistic queries. For

example, the following query searches for either the noun “apple” followed by a verb or the noun “banana” followed by an adjective:

```
([word = "apple" & pos = "NN"] [pos = "VB"]) |
 ([word = "banana" & pos = "NN"] [pos = "JJ"])
```

- **Regular Expression (regex):** CQP incorporates regexes for powerful pattern matching within tokens. For example, the following query can match any word beginning with “un”, like “unusual” or “understand”:

```
[word = "un.*"]
```

This structured approach to query formulation supports the precise targeting of linguistic phenomena, making CQP a useful tool in computational linguistics.

2.1.2.2 Språkbanken and Korp

Språkbanken, based at the University of Gothenburg, and its corpus infrastructure Korp, integrate technological tools with linguistic research for the Swedish language. Språkbanken is a national resource that supports language technology research by providing access to a collection of Swedish text corpora and linguistic tools. Korp, which utilizes CWB, is designed to improve the accessibility, analysis, and curation of Swedish linguistic data [7].

Korp uses CWB to support corpus analysis, management, and interaction. This integration allows Korp to offer functions ranging from the importation and annotation of corpora to providing a web interface for engaging with linguistic data. By incorporating CWB, Korp facilitates access to linguistic resources and supports the exploration of Swedish corpora, contributing to linguistic studies in Sweden [7].

Korp was selected for our study based on its stable infrastructure, ease of use, and the availability of extensive logs due to our affiliation with the University of Gothenburg. The size of these logs, covering 73 corpora with a total of 910 million tokens and 57 million sentences, allows for detailed and large-scale linguistic analysis. This enables a thorough examination of language patterns and usage, making Korp a suitable resource for our research. Additionally, Korp’s open-source nature makes it well-suited for this thesis, providing flexibility and potential for collaboration [7].

2.1.2.3 Alternative Frameworks

In addition to CWB and CQP, several other frameworks and tools are commonly used in corpus management and linguistic analysis. These frameworks provide different functionalities that cater to the varied needs of researchers and educators working with linguistic data.

2.1.2.3.1 Universal Dependencies One example is Universal Dependencies (UD), a framework aimed at establishing a uniform standard for annotating grammatical features across languages. UD supports a range of linguistic studies and applications, particularly in NLP and computational linguistics. It provides guidelines for morphosyntactic annotation, including detailed parts of speech, morphological

features, and syntactic dependencies. This framework is maintained by a collaborative community and has developed treebanks for over 100 languages, making it a valuable resource for multilingual parser development and cross-lingual learning [17].

2.1.2.3.2 Sketch Engine Another significant tool is Sketch Engine, which offers features for text corpus creation, management, and analysis. Sketch Engine supports multiple languages and provides tools for generating word sketches—automatic, corpus-based summaries of a word's grammatical and collocational behavior. This makes Sketch Engine particularly useful for lexicography and broader language studies [18].

2.1.2.3.3 ANNIS3 Similarly, ANNIS3 is another architecture for corpus query and visualization, designed to handle richly annotated and heterogeneous linguistic corpora. This framework uses a graph-based representation that supports multiple and potentially conflicting segmentation layers, allowing for the flexible integration of diverse sources of corpus data. ANNIS3's web-based interface offers customizable visualizations and specialized views of primary data, making it possible to conduct detailed linguistic and digital humanities research. ANNIS3 has been used in various case studies, including richly annotated newspaper treebanks, multilingual manuscript materials, and multimodal spoken language recordings [19].

2.1.2.3.4 AntConc Another useful tool is AntConc, a freeware corpus analysis toolkit developed by Laurence Anthony, particularly designed for educational settings such as technical writing classrooms. AntConc includes tools like a concordancer, word and keyword frequency generators, and tools for cluster and lexical bundle analysis. Its user-friendly interface supports both simple wildcard and complex regular expression searches. AntConc is accessible on multiple platforms and does not require installation, making it convenient for educational environments and individual learners [20].

2.2 Clustering Techniques

Clustering is a fundamental unsupervised ML technique used across various fields, including computational linguistics and digital humanities, to group datasets based on similarities.

This section introduces the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) algorithm, a method that adapts to varying data densities and integrates hierarchical clustering with outlier detection methods, such as Global-Local Outlier Scores from Hierarchies (GLOSH) within the HDBSCAN* framework. These techniques are useful for forming clusters and identifying anomalies that could influence analysis outcomes. Additionally, this section discusses the validation of density-based clustering algorithms, such as Density-Based Clustering Validation (DBCV), which is important for ensuring the reliability of clustering results in complex data scenarios.

2.2.1 Hierarchical Density-Based Spatial Clustering of Applications with Noise

HDBSCAN is a clustering algorithm that extends Density-Based Spatial Clustering of Applications with Noise (DBSCAN) by incorporating a hierarchical clustering approach. Unlike DBSCAN, which uses a single density threshold, HDBSCAN constructs a hierarchy of clusters over varying density levels, allowing for the extraction of flat clusters based on their stability across different scales [11].

The algorithm begins by transforming the dataset into a tree of core distances, based on the density connectivity of points. Clusters emerge at different levels of this tree by adjusting the core distance parameter, enabling HDBSCAN to identify clusters with varying densities. This hierarchical approach helps improve cluster identification and facilitates the detection of outliers and noise in the data [11].

Clusters are extracted from the tree using a measure of cluster stability that evaluates the persistence of clusters throughout the hierarchy. This method helps ensure that the resulting clusters accurately reflect the underlying structure of the data [11].

2.2.2 Integrated Outlier Detection in HDBSCAN*

The HDBSCAN* framework enhances traditional density-based clustering by integrating outlier detection capabilities. The Global-Local Outlier Scores from Hierarchies (GLOSH) method, developed as part of the hierarchical density estimation approach, provides a mechanism for identifying outliers within the data [21].

Outliers can affect clustering results, potentially leading to skewed interpretations or misleading cluster configurations. GLOSH addresses this by evaluating both global and local outlier factors based on the hierarchical structure of the data's density distribution. This method leverages the density-based nature of HDBSCAN* to calculate outlier scores that reflect deviations on both global and local scales [21].

This integrated approach allows for a more detailed assessment of data points that do not conform to typical cluster patterns, aiding decision-making during data preprocessing and clustering. For example, distinguishing between noise and anomalies can help in tuning the clustering process by adjusting parameters such as minimum cluster size [21].

Incorporating GLOSH into the clustering analysis workflow improves the interpretability of clustering outcomes, supporting more effective visualization and analysis of data structures [21].

2.2.3 Density-Based Clustering Validation

Validating the results of density-based clustering algorithms like HDBSCAN is necessary to ensure the reliability of clustering outcomes. Traditional validation indices often do not perform well when dealing with arbitrarily shaped clusters or data with varying densities. To address this, a new relative validation index specifically designed for density-based clusters has been developed [22].

This validation index uses a kernel density estimate to compute the density of each object in the dataset and assesses the connectivity within and between clusters. The quality of clustering is measured by the density connectedness of points within the cluster compared to those outside it. This approach is effective for data that does not conform to globular cluster shapes, as it reflects the properties of density-based clustering algorithms [22].

These advancements in clustering and validation methods contribute to the robustness and applicability of density-based clustering in real-world scenarios, particularly in fields where data can be irregular and noisy [22].

2.2.4 Alternative Clustering Techniques

While HDBSCAN and its associated outlier detection methods are useful for certain clustering tasks, other techniques offer different advantages depending on the data and the application. By exploring alternative clustering methods, it is possible to address specific data characteristics and requirements more effectively. Each technique has its strengths and limitations, allowing practitioners to choose the most suitable approach for their particular use case.

2.2.4.1 K-Means Clustering

K-Means is a commonly used clustering algorithm that partitions a dataset into a predefined number of clusters, where each data point belongs to the cluster with the nearest mean. The algorithm iteratively assigns data points to clusters and updates cluster centroids to minimize variance within each cluster. However, K-Means assumes spherical cluster shapes and is sensitive to the initial placement of centroids, which can lead to suboptimal clustering outcomes [23].

2.2.4.2 Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering builds a hierarchy of clusters through a bottom-up approach. Starting with individual data points as single clusters, the algorithm iteratively merges the closest pairs of clusters until a single cluster or a predefined number of clusters is achieved. This method produces a dendrogram, which helps visualize the merging process and determine the optimal number of clusters. Agglomerative clustering is flexible in terms of cluster shapes and does not require a prior specification of the number of clusters, though it can be computationally intensive for large datasets [24].

2.2.4.3 Spectral Clustering

Spectral clustering uses the eigenvalues of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. This method is effective for identifying clusters with complex structures and can handle non-globular cluster shapes. Spectral clustering involves constructing a similarity graph of the data points and using graph partitioning techniques to find clusters. Despite

its effectiveness, it requires careful construction of the similarity matrix and selection of parameters such as the number of clusters and the scaling of the similarity measure [25].

2.3 Standard Deviation and Coefficient of Variation

Understanding variability within and between datasets is essential in statistical analysis. The Standard Deviation (SD) measures the spread of values within a single dataset, while the Coefficient of Variation (CV) provides a standardized way to measure the relative variability in relation to the mean of the data. These measures are important when comparing datasets of different scales and ensuring reliable statistical conclusions [12], [13].

2.3.1 Standard Deviation

SD is a measure of the dispersion or spread of a set of values. It quantifies the amount of variation in a dataset. The formula for the SD of a sample is given by:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where:

- s is the sample SD,
- n is the sample size,
- x_i are the individual sample values,
- $\bar{x}$ is the sample mean.

SD provides insight into the average distance of each data point from the mean, aiding in understanding the variability within a dataset. A higher SD indicates greater variability, while a lower SD suggests that data points are closer to the mean, implying more consistent data. In this context, a lower SD indicates less variability, which is often preferred when seeking reliable data [12].

2.3.2 Coefficient of Variation

CV is a standardized measure of dispersion, defined as the ratio of SD to the mean. It is used to compare the degree of variation between datasets with different units or means. The formula for CV is:

$$CV = \frac{s}{\bar{x}}$$

where:

- s is the sample SD,
- $\bar{x}$ is the sample mean.

Sometimes, CV is multiplied by 100 to express the result as a percentage, making it easier to interpret in certain contexts. CV is particularly useful when comparing variability between datasets with different units or when the mean may varies across groups. A lower CV indicates less relative variability, while a higher CV suggests greater dispersion in relation to the mean. CV is widely used in fields where the relative variability of data is of interest [13].

2.3.3 Example Calculation

Consider two datasets representing the test scores of students from two different classes:

- **Class A:** Mean score $\bar{x}_A = 75$, Standard Deviation $s_A = 10$, Number of students $n_A = 20$
- **Class B:** Mean score $\bar{x}_B = 85$, Standard Deviation $s_B = 12$, Number of students $n_B = 25$

For each class, SD provides a measure of how spread out the students' scores are around the mean.

  - **Class A:** The SD of 10 indicates that, on average, the scores deviate by 10 points from the mean score of 75.
  - **Class B:** The SD of 12 indicates a slightly larger spread, with an average deviation of 12 points from the mean score of 85.

In this case, Class B has a slightly higher variation in scores compared to Class A, as indicated by their respective SD values.

While SD gives us information about the absolute variability, CV helps us understand the relative variability by taking into account the mean score of each class. CV is calculated as:

$$CV = \frac{s}{\bar{x}}$$

For Class A:

$$CV_A = \frac{10}{75} = 0.133 \text{ (or } 13.3\%)$$

For Class B:

$$CV_B = \frac{12}{85} = 0.141 \text{ (or } 14.1\%)$$

Although Class B has a higher absolute SD (12 compared to 10), CV shows that the relative variability of Class B (14.1%) is also slightly higher than that of Class A (13.3%). This means that, in proportion to their respective mean scores, Class B's test results are more spread out than Class A's.

2.3.4 Alternative Statistical Measures

While SD and CV are commonly used to measure variability, other statistical measures can offer additional insights into the structure of data. These alternatives may be more suitable in situations where SD and CV are less effective, providing a clearer understanding of specific aspects of data variability and distribution.

2.3.4.1 Mean Absolute Deviation

The Median Absolute Deviation (MAD) is a robust measure of statistical dispersion, particularly useful for datasets with outliers or non-normal distributions. It is defined as:

$$MAD = \text{median}(|x_i - \text{median}(x)|)$$

where:

- x_i are the individual sample values.

MAD is less sensitive to extreme values compared to SD, making it advantageous for datasets with significant outliers. It provides insight into the variability of a dataset by focusing on the median of the absolute deviations from the median of the dataset [26].

2.3.4.2 Interquartile Range

The Interquartile Range (IQR) measures the spread of the middle 50% of data points, calculated as the difference between the third quartile ($Q3$) and the first quartile ($Q1$):

$$IQR = Q3 - Q1$$

where:

- $Q3$ is the third quartile (75th percentile),
- $Q1$ is the first quartile (25th percentile).

IQR is effective for understanding the central portion of the data and is robust against outliers. It highlights the range within which the central half of the data lies, providing a measure that is not influenced by extreme values [13].

2.3.4.3 Pooled Standard Deviation

The pooled SD is used when combining SDs from multiple groups with the assumption of equal variances. It provides a weighted average of the variability across the groups. The formula for the pooled SD is:

$$s_p = \sqrt{\frac{\sum_{i=1}^{I}(n_i - 1)s_i^2}{\sum_{i=1}^{I}(n_i - 1)}}$$

where:

- s_i are SDs of the individual groups,
- n_i are the sample sizes of the individual groups,
- I is the number of groups.

The pooled SD is useful when comparing multiple datasets and provides an estimation of overall variability when the assumption of equal variance holds. By weighting SDs according to sample size, it offers a balanced view of variability across different groups [12].

3

Methods

This chapter provides a detailed overview of the methods used to achieve the objectives outlined in Section 1.3. The overall process is summarized in Figure 3.1, which illustrates each step involved, from initial data extraction to creating the benchmark.

We begin by extracting the queries and aggregating them by unique CQP values, counting their occurrences to prepare the dataset for further analysis. Next, we apply two successive levels of query abstraction, designed to retain essential structural elements while generalizing the data to facilitate pattern analysis. Following this abstraction, we conduct a pattern analysis to understand the structure and characteristics of user queries. The result of this analysis is the creation of ML features that capture key aspects of the queries. These features are then used for ML tasks, such as segmenting and classifying user queries to gain insights into user behavior patterns. Additionally, we evaluate query execution times and develop a benchmark for comparing the performance of corpora search systems, providing insights into the efficiency and reliability of query handling across different clusters and corpora.

Figure 3.1: Process flow of the methods.

3.1 Query Extraction and Aggregation

The process begins by extracting raw queries from the Korp server logs dated from October 2022 to January 2024, which represent the complete set of logs available to us. We then aggregate these queries based on unique exact string matches of CQP values and count their occurrences. This initial aggregation is important for mapping the overall distribution of queries, providing a foundational dataset for subsequent analysis.

We do not attempt to determine if one query is a subset or variation of another. Our aggregation is based solely on exact matches of the query strings as they appear in the logs. This approach does not account for potential overlaps or inclusions between different queries beyond their exact string representation.

3.2 Queries Abstraction

After aggregation, we abstract the CQP queries into a more generalized format using a two-level abstraction process. This method retains essential structural elements while simplifying the data, facilitating pattern analysis. The purpose of this two-level approach is to ensure the accuracy of pattern analysis by breaking the process into manageable steps and allowing for manual verification at each stage. This enhances the precision of pattern detection and feature extraction, leading to more meaningful clustering and better ML outcomes. By incrementally reducing the complexity of queries, this method ensures that the identified patterns are both accurate and reliable, providing a solid foundation for further analysis.

3.2.1 Level 1 Abstraction

The first level of abstraction transforms specific query elements into a more generalized format. This process includes:

- **Attribute Pattern Identification:** Specialized regexes are used to extract attribute patterns, which consist of attribute parameters, operators, and their corresponding values. This step isolates the components of the query that are structurally significant.
- **Generalization of Attribute Values:** Non-regex special characters within attribute values are replaced with a generic placeholder `'j'`, which abstracts the specific content while preserving regex special characters that are essential for maintaining the query's functional integrity. This differentiation ensures that the structural and operational elements of the query, governed by regex syntax, are preserved, allowing for accurate analysis of the query's logic and complexity. The list of special characters includes:

  ```
  .  ^ $ * + ?  { } [ ] \ | ( ) - = !  < > , :  ; / & % # @ ‘ ~" ’
                \s \S \w \W \d \D \b \n \r \t \0 \v
  ```

  For example, consider the attribute value in the following token:

  ```
  [phrase = "error\.detected.*"]
  ```

  In this case, the non-regex characters `'error'`, `'detected'`, and the escaped dot `'\.'` are replaced with the placeholder `'j'`, resulting in `"j.*"` while retaining the non-escaped dot and asterisk `'.*'`. This transformation abstracts the specific words while preserving the regex special characters that contribute to the query's operational logic.
- **Numeric Value Generalization:** Specific numeric values within the queries, which often specify quantities or define conditions, are substituted with generic placeholders (`'x'`, `'y'`, `'z'`), where `'z'` is used for standalone numbers and `'x'` and `'y'` for ranges. This includes modifying numeric quantifiers and ranges to standardized forms, such as replacing explicit numbers in quantifiers like `{3,6}` (which dictates matching between 3 and 6 instances of a preceding element) and `{3,}` (which dictates matching instances of a preceding element with a

minimum of 3) with `{x,y}`. This generalization abstracts the quantity-specific data while preserving the logical structure of the query, maintaining the integrity of the query's operational logic without revealing precise numerical details.

- **Corpus Identifier Generalization:** References to specific corpora within the queries are replaced with a non-specific placeholder `'l_c'`, abstracting direct references to corpus data.

This level of abstraction standardizes query elements into a more generalized format, facilitating detailed analysis while preserving essential structural components.

3.2.2 Level 2 Abstraction

Building upon the first level, the second level of abstraction further simplifies and generalizes the query strings by reducing them to their core structural components:

- **Placeholder Substitution for Attributes:** Each attribute part, including parameter, operator, and value, is replaced with a generic placeholder `'a_p'`, removing residual specificity from the first level of abstraction.
- **Removal of Non-Essential Characters:** This process strips all whitespaces and quotation marks, simplifying the query into a streamlined form that emphasizes structural patterns over lexical content.

The second level of abstraction is designed to distill the query down to its fundamental structural components, emphasizing patterns and relationships within the data without the distractions of specific content.

Together, these two levels of abstraction provide a robust method for transforming raw query data into a form suitable for comprehensive analysis, focusing on structural insights while ensuring data integrity.

3.2.3 Examples of Query Abstraction Process

To illustrate the abstraction process, we consider a few concrete examples:

- **Example Query 1:** The following query searches for one to five consecutive occurrences of a noun starting with `'int'`:

```
[word = "int.*" & pos = "NN"]{1,5}
```

1. **Level 1 Abstraction:** In the first level of abstraction, non-regex special characters within the attribute values are generalized. The word `'int.*'` is replaced with a placeholder `'j.*'`, `'NN'` is replaced with `'j'`, and the range `{1,5}` becomes `{x,y}`, resulting in the following transformed query:

```
[word = "j.*" & pos = "j"]{x,y}
```

2. **Level 2 Abstraction:** Building on the first abstraction, Level 2 further simplifies the query by replacing all attribute parts with a generalized

placeholder. Each of the attributes `'word = "j"'` and `'pos = "j"'` is replaced with `'a_p'`, leading to the following form:

```
[a_p&a_p]{x,y}
```

- **Example Query 2:** The following query searches for lemmas containing `'become'`, ignoring case and diacritics, in texts dated after January 1, 1800:

```
[lemma contains "become" %cd & int(_.text_datefrom) > 18000101]
```

1. **Level 1 Abstraction:** The lemma `'become'` is replaced with a placeholder `'j'`, and the date condition is abstracted by substituting the specific date with a placeholder `'z'`, resulting in the following:

```
[lemma contains "j" %cd & int(_.text_datefrom) > z]
```

2. **Level 2 Abstraction:** Further abstraction replaces each attribute with a generic placeholder. Thus, `'lemma contains "j"'` and `'int(_.text_datefrom) > z'` become `'a_p'`, leading to the following:

```
[a_p%cd&a_p]
```

- **Example Query 3:** The following query targets words matching either a pattern that ends with `'me.'` or starts with `'be'`, where `'\w+'` matches one or more word characters, and it is specified for the SVEN-SV and EUROPARL-EN corpora:

```
[word = "\w+(me\.  | be.*)"]:LINKED_CORPUS:SVEN-SV|EUROPARL-EN
```

1. **Level 1 Abstraction:** The regex patterns within the attribute values are preserved, but specific elements like `'me\.'` and `'be'` are generalized. The corpus identifiers are also abstracted to a non-specific placeholder. Thus, the query is transformed into the following form where `'j'` replaces `'me\.'` and `'j.*'` replaces `'be.*'`, and `'l_c'` replaces specific corpus identifiers:

```
[word = "\w+(j | j.*)"]:LINKED_CORPUS:l_c
```

2. **Level 2 Abstraction:** The abstraction process further reduces specificity by replacing the attribute with a placeholder `'a_p'`, simplifying the query to the following form where all distinctive characteristics of the patterns are removed, highlighting the structure of the query without revealing the exact content:

```
[a_p]:LINKED_CORPUS:l_c
```

3.3 Query Analysis

This section outlines the process of analyzing abstracted queries and examining the use of attributes within those queries. The procedure involves identifying and quantifying various syntactic and operational patterns, as well as integrating detailed counts of attribute parameters and operators. This structured approach is designed

to provide a framework for understanding the complexity and structure of user queries, as well as the specific attributes they employ.

3.3.1 Analysis of Abstracted Queries

The abstracted queries are analyzed to identify and quantify various syntactic and operational patterns using pattern recognition techniques. This analysis is performed to assess the types of queries processed by the system. The result of this analysis is the creation of ML features, each represented as a positive integer. These features are used in subsequent ML tasks, such as algorithmic analysis and modeling. Below is a summary of how different query components are identified and quantified:

- **Quantifiers:** regexes are used to detect quantifiers like `'*'`, `'+'`, `'?'`, `'{n}'`, `'{n,}'`, and `'{n,m}'`. The frequency of these elements is calculated to understand how users define the scope of their queries, both within attribute values and the broader query structure.
- **Logical Operators:** Logical operators such as `'&'`, `'|'`, and `'!'` are identified to construct complex logical conditions. The analysis quantifies the frequency of these operators to better understand query complexity.
- **Ignore Case and Diacritics Flags:** Flags like `'%c'`, `'%d'`, and `'%cd'`, which modify text matching in terms of case sensitivity and diacritics, are counted to assess how users manage text matching criteria.
- **Zero-Width and Lookaround Assertions:** This measure quantifies the use of zero-width assertions, including lookaheads and lookbehinds, which are important constructs in regexes. These assertions allow pattern checking around a specific point in the string without consuming characters. By measuring the use of these assertions, we gain insights into the complexity and specificity of pattern matching.
- **Grouping and Nested Complexity:** Grouping symbols `'( )'` are used to capture subexpressions and measure the complexity of queries. By analyzing the frequency and depth of these nested structures, we assess the intricacy of the user's query logic.
- **Sentence and Phrase Boundaries:** Sentence (`'<s>'`, `'</s>'`) and noun phrase (`'<np>'`, `'</np>'`) boundary markers are quantified to reflect how users specify linguistic structures in their searches.
- **Attribute Comparisons and Operators:** Attribute comparisons and operators such as equality, inequality, and range specifications are counted to evaluate how users specify their search criteria.
- **Special Characters and Wildcards:** The use of special characters and regex wildcards, including general pattern matchers (`'\s'`, `'\S'`, `'\w'`, `'\W'`, `'\d'`, `'\D'`), is detected and quantified to understand their influence on pattern matching behavior.

- **Target Markers and Attribute Parts:** These elements are analyzed to understand how users target specific attributes within their queries, providing insights into the granularity of query specifications.

- **Group Labeling:** The use of labels within grouping constructs is quantified to assess how users organize and reference complex query patterns.

- **Within Restrictions:** The frequency of the `'within'` keyword usage is analyzed to understand how users specify the contextual bounds of their searches.

- **Linked Corpus:** The use of linked corpus specifications is quantified to understand how users integrate multiple corpora into a single query, reflecting cross-corpus search capabilities.

This analysis offers a detailed understanding of the types and complexities of queries processed by the system, providing valuable insights into user interaction patterns. The features, fully listed in Appendix A, are quantified aspects of user queries and are defined as positive integers. These features are used in ML tasks, ensuring consistency in data type for algorithmic analysis. This structured approach aids in documenting query complexity and facilitates advanced analytical applications by leveraging quantitative data to predict and model user behavior.

3.3.2 Detailed Attribute Analysis

Following the analysis of abstracted queries, the dataset is further enriched by integrating detailed counts of each attribute parameter and operator identified in the queries. This stage adds quantitative features to the dataset, providing deeper insights into the structure and usage patterns of the queries.

During this phase, each query is examined to identify and record the occurrences of each attribute parameter and its associated operator. For example, in the query below, the attribute `'word'` is counted twice, as it is used with two different operators: `'='` and `'!='`:

```
[word = "hello"] [word != "world"]
```

From this analysis, a new feature called `'cqp_n_a_p_word'` is generated, where `'n'` stands for number, `'a'` for attribute, `'p'` for parameter, and `'word'` because the parameter `'word'` was found. Additionally, two other features, `'cqp_n_a_o_='` and `'cqp_n_a_o_!='`, are created for the operators `'='` and `'!='`, respectively.

This precise counting highlights not only the frequency of attribute usage but also the complexity and variety of the logical conditions applied by users in their queries. The resulting data from this analysis provides a set of ML features, each represented as a positive integer. These features enhances our understanding of user query behavior and for further algorithmic analysis.

3.4 Clustering of Analysis Data

This section outlines the method used for clustering the CQP queries, aimed at analyzing and understanding the patterns, complexities, and distribution of user interactions. The workflow supports the identification of common user trends and enables detailed analysis of the data through clustering techniques and statistical methods.

3.4.1 Data Scaling

The clustering process begins with scaling the numerical features extracted from the queries. To effectively handle potential outliers and non-uniform data distributions, we use `RobustScaler` from the `Scikit-learn` library [27].

`RobustScaler` is well-suited for data that may contain outliers or is not normally distributed. Unlike standard scaling methods that subtract the mean and divide by the variance, `RobustScaler` uses the median and the Interquartile Range (IQR) for scaling, making it less sensitive to outliers.

This method ensures that the scaled features are less affected by outliers, which could otherwise distort the results of analysis algorithms. By using the median and IQR, we ensure that the scale transformation is determined by the central data points, thereby preserving the integrity of data clusters.

3.4.2 Adaptive Clustering with Iterative Hyperparameter Optimization

Using the HDBSCAN clustering algorithm implemented within the HDBSCAN* framework, we then group these queries based on the scaled features. HDBSCAN is effective for this application due to its ability to handle varying data densities and to automatically adapt cluster formation without needing to predefine the number of clusters. This feature is important for managing the diverse nature of query data in digital humanities, where standard clustering parameters may not be sufficient.

The minimum cluster size hyperparameter of the HDBSCAN algorithm was adjusted through an iterative process to improve clustering results. This optimization was guided by evaluating a fast approximation of the DBCV score, which ranges from 0 to 1. The approximation method, provided by the `hdbscan` Python library, differs from the traditional DBCV score, which relies on the all-points minimum spanning tree, by using the mutual-reachability minimum spanning tree instead. This modification enhances computational efficiency, allowing for more frequent adjustments during the tuning process. It is important to note that this score does not provide an absolute measure of clustering quality but serves as a comparative tool for assessing the relative effectiveness of different hyperparameter configurations [28].

Using this relative validity score enabled the precise tuning of the algorithm, ensuring that the clusters formed are coherent and stable.

3.5 Performance Evaluation of Query Execution Times

This section describes the process of evaluating the performance of query execution times in CWB. This process involves executing the clustered queries on selected corpora to measure and document their performance. By analyzing execution times across various corpora, we aim to understand CWB's efficiency in handling different types of queries, identifying performance benchmarks and potential bottlenecks.

3.5.1 Selection of Corpora

To evaluate the execution time of the clustered queries, 14 diverse corpora were downloaded from Språkbanken. These corpora were selected based on their varied text topics and manageable sizes, ensuring compatibility with the computational capacity of the local machine used for testing. This selection balances the need for comprehensive linguistic data with practical constraints of time and resource availability for this thesis. Appendix B provides detailed descriptions and token counts for each chosen corpus, illustrating the range of linguistic data included in the evaluation.

3.5.2 Conversion Process of Corpora

The process of preparing and converting the downloaded text corpora into a format compatible with CWB involves several steps. Initially, corpora are often stored in compressed formats, requiring decompression before further processing. The steps include:

1. **Decompression:** Extracting files from `tar` and `bz2` archives to obtain Extensible Markup Language (XML) files containing the corpus data.
2. **XML Parsing and Correction:** Decompressed XML files are parsed to identify and correct any imbalances in XML tags, ensuring that all tags are properly closed and nested to maintain data integrity.
3. **Conversion to VeRticalized Text (VRT) Format:** Parsed XML files are converted into VRT format. This format is a plain text representation where each line corresponds to a single token, annotated with relevant linguistic information. Text and metadata from the XML files are structured into this tabular format.
4. **Corpus Encoding:** The VRT files are encoded using the CWB tools. This process involves converting the VRT files into a binary format that supports efficient querying. Attributes and structural annotations are defined and encoded to facilitate complex queries.
5. **Index Building:** After encoding, binary index files are created to support fast querying and efficient data retrieval.

6. **Token Stream Compression:** The token streams in the corpus are compressed to save disk space and enhance access speeds.
7. **Index File Compression:** Finally, the index files are compressed to further optimize storage and improve query performance.

3.5.3 Measurement of Query Execution Times

The measurement of query execution times was conducted to assess the performance of the clustered queries on the processed corpora in CWB. The following subsections describe the measurement process in detail, explaining the rationale behind each step. This process is visually summarized in Figure 3.2 for clarity.

3.5.3.1 Filtering of Clustered Queries

The first step involved filtering the queries to exclude noise data. In this context, noise data refers to irrelevant or uninformative queries that result from the clustering process. Filtering out noise data ensures that only meaningful queries are included in the measurement, providing a more accurate assessment of query performance.

3.5.3.2 Parallel Query Execution

To optimize resource utilization and efficiency, the queries were executed in parallel using 50% of the available logical processors. For the local machine used in this measurement, which is described in Section 3.7.2, this meant utilizing 4 out of 8 logical processors. This approach prevented 100% Central Processing Unit (CPU) utilization, leaving the remaining processors available for other essential processes. Limiting CPU usage to 50% mitigated the risk of resource contention, ensuring that the measured execution times reflect the true performance of the queries without being inflated by system overhead.

3.5.3.3 Generating and Executing Corpus Query Processor Commands

For each query, a CWB script file was generated and executed three times. The choice of three executions aimed to balance the need for reliable performance data with the constraints of available resources and time. Running each script three times accounts for minor variations in execution time, providing a reliable average without excessively prolonging the testing process.

The script file contained commands necessary to execute the query on the specified corpus, focusing on returning the number of matches rather than the actual text results. Focusing on the number of matches allows for accurate measurement of the query's execution time without the additional overhead associated with handling and displaying results. Each script included:

1. Setting the corpus with `'CORPUS_NAME;'`.
2. Assigning the query to a variable `'A'` with `'A = CQP_QUERY;'`.
3. Calculating the size of the result set with `'size A;'`.

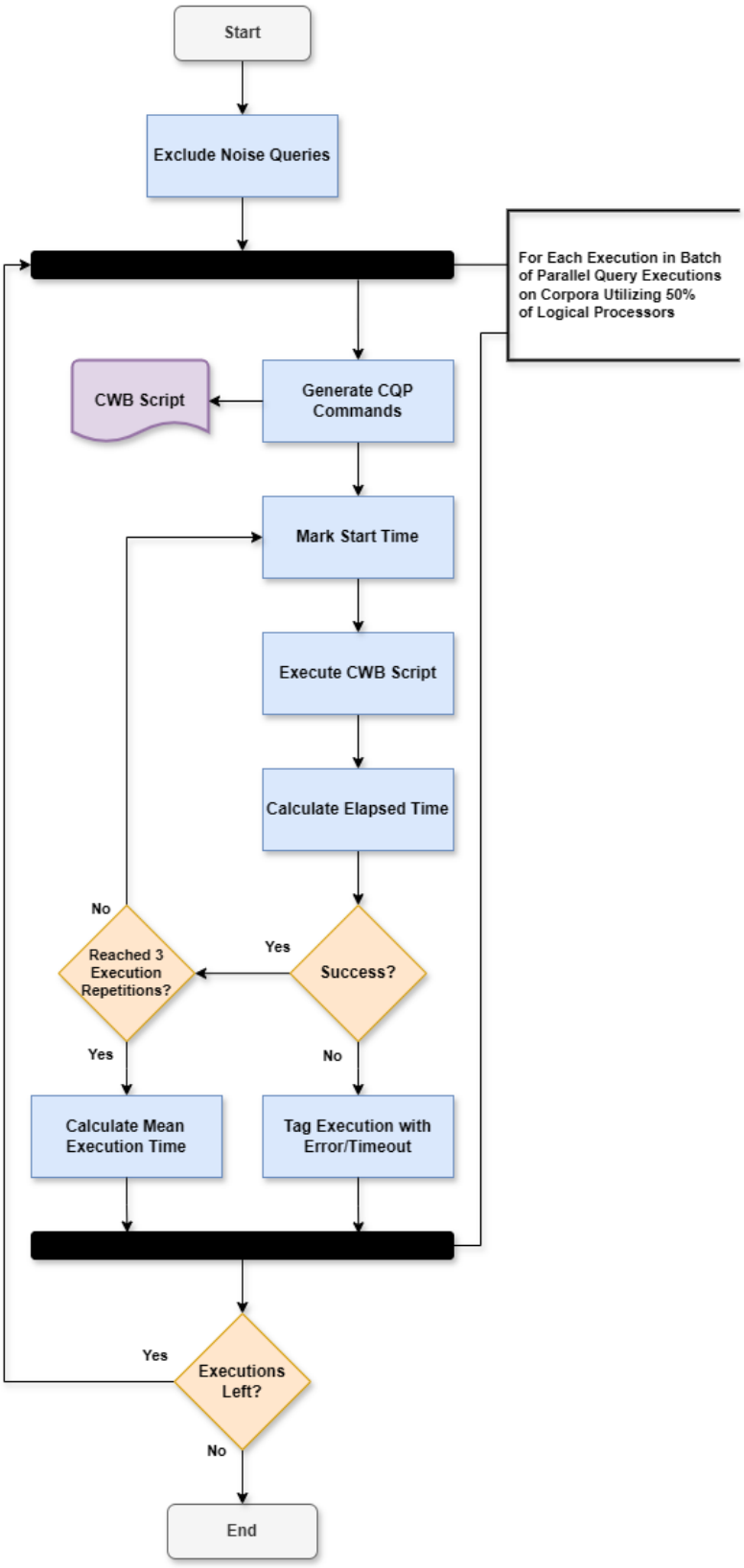

Figure 3.2: Flowchart illustrating the process of measuring query execution times.

3.5.3.4 Execution Timing and Management

The execution time for each query was measured using the `timeit` module in Python, which provides a precise way to time small bits of Python code [29]. The `timeit` module is particularly suited for timing short code snippets as it selects the best available timer for the platform and runs the code multiple times to reduce the impact of other system activities on the timing results. For each query execution, a timer was started before running the command, the command was executed, the output was captured, and then the timer was stopped to calculate the elapsed time. These steps were repeated three times, and the results were averaged to obtain the mean execution time.

A 60-second timeout was enforced for each execution to prevent prolonged execution times and potential blocking of the testing process. This timeout duration was chosen to allow sufficient time for complex queries to complete while avoiding unnecessary delays. If a timeout occurred or a CWB error was encountered during any of the three executions, the remaining executions were skipped. This approach ensured efficient use of time and resources. In cases where an error or timeout occurred, the mean execution time was not calculated; instead, a tag was added to indicate the issue. Errors were prioritized over timeouts because they typically occur more quickly and provide a clearer indication of issues compared to reaching the 60-second timeout.

This approach provided insights into the performance of the clustered queries on the processed corpora, identifying potential issues and contributing to the performance analysis. For brevity, the term "mean execution time" will be referred to simply as "execution time" throughout the remainder of the thesis.

3.6 Benchmark Development

To establish a robust benchmark for assessing the performance of corpora search systems, we applied a statistical approach, focusing on query executions across different clusters. Each cluster corresponds to a distinct level of query structure complexity, serving as a specific reference point in our evaluation.

The execution success rate and the Coefficient of Variation (CV) for each cluster evaluate our trust in the reliability and consistency of the queries it contains. A high execution success rate indicates effective query handling within the cluster, while a lower CV suggests more consistent execution times across queries.

CV of the execution time for each query within a cluster measures the reliability of the queries. The query with the lowest CV across different corpora is considered the most representative and is identified as the representative query for that cluster. This indicates that the selected representative query demonstrates consistent performance, making it an effective benchmark measure. Additionally, the average of all execution times for the representative query was used to establish a reference value that characterizes the performance standard for the cluster.

Following the establishment of these metrics, we introduced a refinement step. This

involves manually analyzing and merging clusters with structurally similar queries to eliminate redundancy and ensure a concise benchmark. By focusing on structural similarities and selecting the most stable cluster based on the lowest CV, this refinement ensures that the final benchmark is both representative and efficient, providing a more streamlined and effective evaluation tool for assessing corpora search systems.

3.6.1 Evaluating Execution Time Performance: Variability and Success Rates

The performance of queries across different clusters is evaluated using two metrics: the CV and the execution success rate. These metrics provide an understanding of both the performance stability (through CV) and the reliability (through execution success rate) of the clusters.

3.6.1.1 Execution Success Rate

The execution success rate measures the reliability of query executions within each cluster. It is calculated as the ratio of valid query executions to total executions, including errors and timeouts. A high success rate indicates reliable execution, while a low success rate highlights issues such as frequent errors or timeouts. This metric provides insight into the completeness and reliability of query handling within each cluster.

3.6.1.2 Evaluating Variability Using Coefficient of Variation

CV is used to assess the variability of execution times within each cluster and across queries. The process of evaluating execution time variability is illustrated in Figure 3.3, which visualizes the procedure for one cluster. The steps for evaluating variability and identifying representative queries are as follows:

1. Extract data specific to each cluster, focusing on valid execution times (excluding errors and timeouts). If a cluster does not have at least two valid execution times, we skip calculations for that cluster to avoid misleading results due to insufficient data.
2. Calculate the SD and mean execution time for the entire cluster. If the mean is zero, we skip the calculations for that cluster because the CV would be undefined.
3. Use these values to calculate the CV for the cluster as a whole. This consolidates the variability of execution times by normalizing the SD relative to the mean, providing a measure of performance consistency.
4. For valid clusters, calculate the SD and mean execution time for each query within the cluster, then compute the CV based on these values.
5. Identify the query with the lowest CV within the cluster and mark it as the representative query.

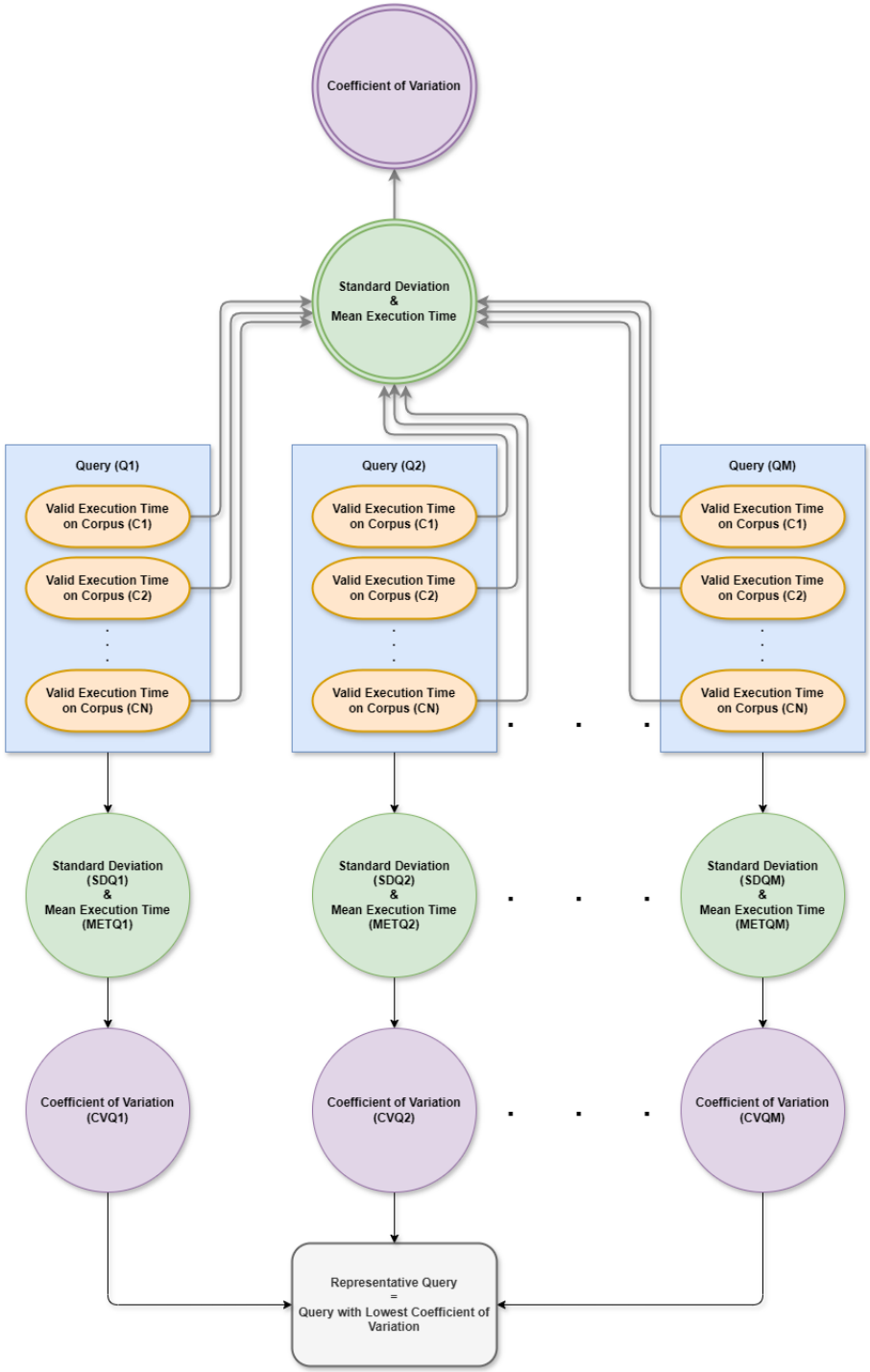

Figure 3.3: Illustration of the process for evaluating execution time variability using the coefficient of variation for both the entire cluster and individual queries within the cluster. The query with the lowest coefficient of variation is selected as the representative query.

CV adjusts for differences in mean execution times across clusters and individual queries, allowing for a fair comparison of variability. This helps identify which queries have more stable performance and which clusters show greater consistency in execution times.

3.6.2 Benchmark Refinement

After developing the benchmark, we undertake a manual refinement phase to optimize it. This step focuses on grouping clusters based on the structural similarity of the queries they contain.

We manually analyze the representative query from each cluster, identifying those with similar structural patterns, even if they differ in attribute value lengths. By disregarding these differences, we can merge structurally similar clusters, resulting in a more compact and representative benchmark. This process ensures that the final clusters are not fragmented based on minor variations, leading to a clearer and more consistent evaluation framework.

To maintain the robustness of our evaluation, we combine clusters and select the one with the lowest CV. This approach ensures that the chosen cluster represents the most consistent and reliable set of queries within the merged group. The representative query from this cluster, characterized by the lowest variability in execution times, is then selected for the final benchmark.

This refinement step enhances the clarity and reliability of our evaluation framework, ensuring that the final benchmark provides a focused and consistent tool for assessing the performance of corpora search systems.

3.7 Implementation Details

This section provides an overview of the methods and tools used in this thesis, covering data manipulation, preprocessing, clustering, visualization, timing, and parallelization operations, as well as details about the local machine and software installations.

3.7.1 Overview of Tools and Libraries

The methods employed in this thesis are implemented using Python, a widely used programming language known for its extensive library ecosystem.

Data manipulation and preprocessing tasks are performed using the `Pandas` and `NumPy` libraries. `Pandas` offers efficient, high-level data structures and operations for manipulating numerical tables and time series, making it suitable for handling the datasets extracted from the Korp server logs [30]. `NumPy` complements this by providing mathematical functions, random number generators, and array operations essential for scientific computing [31].

For clustering, we use `hdbscan`, a library that implements the HDBSCAN algorithm,

which is effective in handling varying data densities—an important factor given the diverse nature of the text data in our dataset [28]. The `Scikit-learn` library is used for feature scaling, which standardizes dataset features to ensure that the model is not biased or overly sensitive to the scale of measurements [27].

Visualizations play a big role in our analysis, helping to present complex data clearly. We use the `Seaborn` and `Matplotlib` libraries for this purpose [32], [33]. `Seaborn` extends `Matplotlib`'s capabilities by providing a high-level interface for creating statistical graphics, while `Matplotlib` is used for generating detailed charts that illustrate clustering results and feature distributions across different query types.

The `timeit` module is used to measure the execution time of queries. This module is well-suited for timing short code snippets as it selects the most accurate timer available on the platform and executes the code multiple times to minimize the impact of system activities on the results [29].

For parallelization, we use the `concurrent.futures` module, which provides high-level interfaces for executing callables asynchronously. This module supports both multi-threading and multi-processing, enabling us to efficiently distribute computational tasks and utilize multi-core processors. Using `concurrent.futures` helps reduce the execution time of data processing tasks, thereby improving the overall performance and scalability of our methods [34].

3.7.2 Local Machine Specifications

The processes and testing on the corpora were conducted on a local machine with the specifications listed in Table 3.1.

Table 3.1: Specifications of the local machine used for corpora processing and testing.

Component	**Specification**
Processor	Intel(R) Core(TM) i7-4720HQ CPU, 2.60 GHz, with 8 logical processors
RAM	16 GB, 1600 MHz
Storage	1 TB HDD
Operating System	Windows 10 Home, version 22H2
Additional Software	WSL with Ubuntu 22.04, Python 3.12.2

3.7.3 Installation of Corpus Workbench

To ensure compatibility and functionality for the processes involved in preparing and querying text corpora, CWB was downloaded from `https://cwb.sourceforge.io/` and installed in two versions on the local machine. This was necessary to address specific compatibility issues encountered with the native Windows version.

The native Windows version, CWB Core version 3.4.34, was initially installed due to its straightforward installation process. However, this version encountered issues with essential commands needed for converting the downloaded corpora into

a format compatible with CWB. Specifically, some commands related to corpus encoding, index building, token stream compression, and index file compression did not function properly, requiring the installation of an alternative version to ensure full operational capability.

To overcome these limitations, CWB Core version 3.5.0, designed for Linux, was installed on Windows Subsystem for Linux (WSL). WSL allows for the execution of Linux-specific commands and software within a Windows environment, ensuring compatibility with all CWB features. This version successfully handled the problematic commands, enabling accurate corpus encoding, efficient index building, and reliable compression processes.

The dual installation approach, utilizing both the native Windows version and the Linux version on WSL, provided a robust and functional environment for performing the required processes, ensuring seamless conversion and querying of the downloaded corpora.

4

Results

This chapter presents the findings from the analysis of query data within the Korp server logs, as outlined in Chapter 3. We employed the HDBSCAN clustering algorithm to segment user queries into distinct clusters, identifying common patterns and notable outliers. The results include cluster validation, general statistics about the queries, insights from the cluster analysis, and an examination of query execution times. Additionally, we present an analysis of the created benchmark used to evaluate corpora search systems. The specific results are discussed in the following sections.

4.1 General Statistics of Queries

The analysis of the Korp server logs shows that 462,227 queries were processed, reflecting significant interaction with the system. Of these, 119,844 were unique, representing 25.92% of the total. These unique queries are defined as exact string matches of the CQP values, as explained in Section 3.1. The remaining 342,383 queries, which make up 74.08% of the total, were repetitive. This suggests patterns of frequent searches or routine information requests, indicating common user behaviors.

Further analysis of the processed queries reveals specific searches that occur frequently. For example, the following query appeared 10,272 times, representing 2.22% of all processed queries:

```
[word = "100-årsminnet"]
```

Similarly, the following query was executed 5,018 times, accounting for 1.09% of all queries:

```
[word = "robots.txt"]
```

The following query with an empty word attribute was executed 2,628 times, further highlighting patterns in user search behavior across the processed data:

```
[word = ""]
```

4.2 Validation of Clustering Process

The HDBSCAN algorithm achieved a DBCV score of 0.736, indicating a good level of clustering stability and coherence, which is important for reliable data analysis. The best clustering performance was observed when the minimum cluster size hyperparameter was set within a range calculated from 0.1% to 0.9% of the total data points, with increments of 0.1%. The optimal result was at the lower end of this range, corresponding to 0.1% of the total data points or a minimum of 119 data points per cluster.

This range was chosen to balance the need for a sufficient number of clusters to capture diverse data patterns with the goal of minimizing noise within each cluster. By starting at a threshold of 0.1% and extending up to 0.9%, the hyperparameter settings allow for the evaluation of various cluster sizes to determine which provides the most coherent and stable grouping under different scenarios. The incremental adjustments enable a detailed examination of how cluster stability evolves as the minimum size requirement increases, ensuring that the final selection of parameters effectively segments large datasets while preserving the essential relationships and patterns among the data points. This methodical hyperparameter tuning is important for leveraging the full potential of HDBSCAN in data segmentation.

4.3 Analysis of Clusters

Building on the segmentation achieved through HDBSCAN, this section presents the results of the cluster analysis. It explores how these clusters differentiate various user search behaviors recorded in the Korp server logs, highlighting the diversity and specificity of the queries.

4.3.1 Overview of Clustering Results

The clustering process using HDBSCAN on unique queries resulted in 161 distinct clusters. Cluster -1, which represents noise, includes 19,754 queries, accounting for approximately 16.48% of all analyzed queries. This cluster mainly consists of queries that deviate significantly from common patterns or do not meet the minimum cluster size, thereby filtering out less relevant data.

Figure 4.1 shows the distribution of data points across all clusters (excluding Cluster -1). This bar chart illustrates the number of data points in each cluster, highlighting the variation in cluster sizes. The majority of data points are concentrated in a few large clusters, while many smaller clusters represent more specialized queries. This distribution emphasizes the centralization of frequent query types and the range of less common searches.

The largest cluster comprises 4,848 queries, representing 4.05% of all unique queries, and reflects prevalent search patterns likely aligned with common user requirements. The top 10 largest clusters together account for over 30% of all queries, indicating

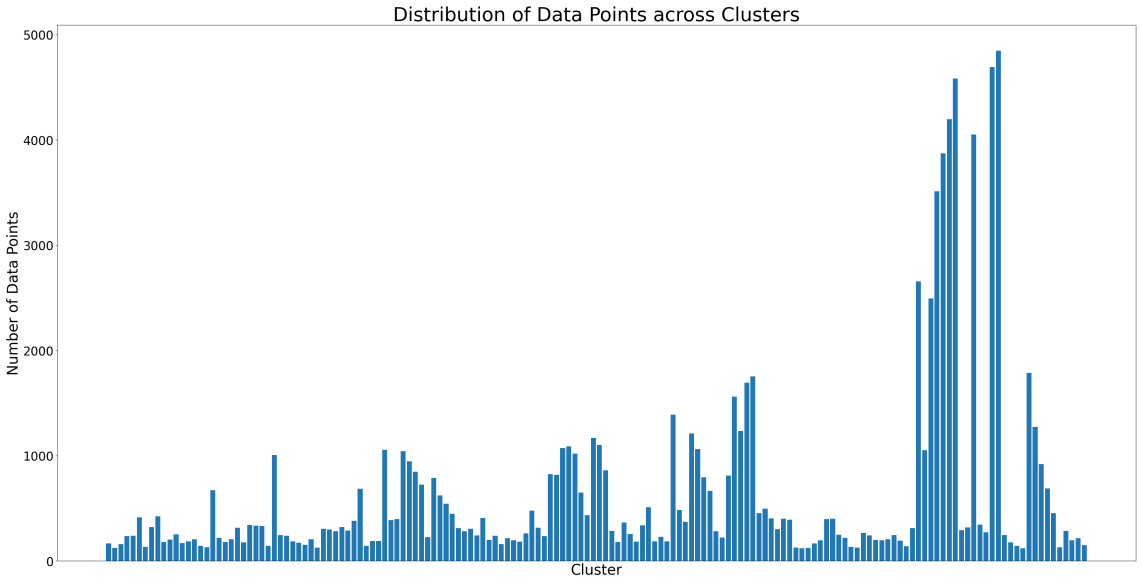

Figure 4.1: Distribution of data points across clusters (excluding Cluster -1).

a concentration of the most frequent query types. This suggests that a significant number of queries are focused on common themes among users.

In contrast, the 10 smallest clusters contain between 120 to 133 queries each, roughly 1% of the total, representing more specialized interests. This demonstrates the system's ability to cater to both general and highly specific information needs.

4.3.2 Complexity and Homogeneity across Clusters

Clusters exhibit significant variation in query complexity, specificity, and homogeneity, reflecting distinct user behaviors and needs. These variations are visualized in Figure 4.2, which focuses on the most relevant data points by filtering out lower-value intersections. The heatmap displays standardized mean values for various features across the identified clusters (excluding Cluster -1). By applying a threshold of 2, only clusters and features with standardized mean values above this level are included. This approach highlights the most significant data points, enhancing clarity and interpretability. The heatmap facilitates the quick identification of the most prevalent or unique features within specific clusters, thereby deepening our understanding of user search behaviors within the dataset.

The largest clusters are characterized by simpler query structures, with lower averages of logical operators and quantifiers. Conversely, smaller clusters typically feature more complex queries, indicating specialized information needs or advanced user knowledge.

Table 4.1 presents a selection of identified clusters, each with example queries that illustrate key aspects discussed in our analysis. These clusters are highlighted for their significance in demonstrating varying levels of complexity, specificity, and homogeneity. For clusters where only one example query is provided, it suggests that the remaining queries in that cluster are structurally similar, making additional ex-

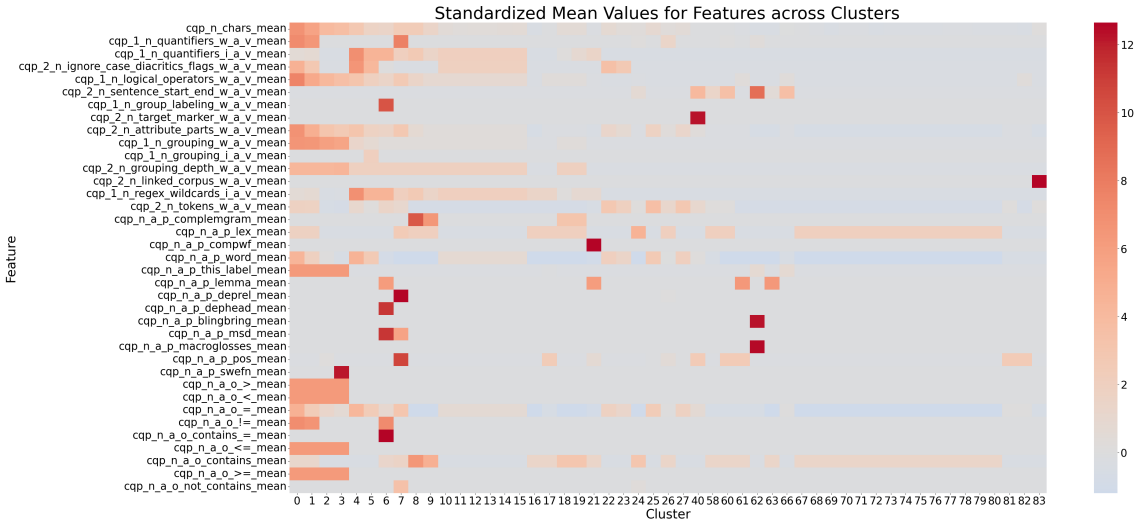

Figure 4.2: Heatmap of filtered standardized mean values for features across clusters (excluding cluster -1). This heatmap includes only the features and clusters with standardized mean values exceeding a threshold of 2, focusing on the most significant data points to enhance clarity and interpretability.

amples unnecessary. This concise presentation aids in a clearer interpretation of our clustering results.

4.3.2.1 Query Structure and Complexity

Larger clusters demonstrate minimal use of logical operators and quantifiers. For example, clusters with high query counts show an average of nearly zero logical operators and quantifiers, indicating that these clusters primarily serve users with straightforward search requirements. This simplicity suggests that a significant portion of the user base engages in basic searches that do not require advanced search techniques.

In contrast, smaller clusters often exhibit higher averages of logical operators and quantifiers, reflecting users with more complex information-seeking behaviors who employ advanced search techniques. This complexity suggests a higher level of search sophistication and specificity, indicating that these users navigate more intricate information landscapes and require precise, refined search criteria.

Another distinguishing factor is the use of case insensitivity or diacritic ignoring flags. Larger clusters exhibit less frequent use of these features, suggesting that their users operate within more standard linguistic contexts where such nuances are less critical. Additionally, it is likely that many users are simply unaware of these flags and therefore do not use them. In contrast, smaller clusters show greater use of these flags, indicating adaptation to diverse linguistic or regional search contexts. This adaptation caters to users who might require more nuanced search parameters due to language variations, reflecting the system's versatility in handling different linguistic needs.

Table 4.1: A selection of identified clusters with example queries, showcasing clusters discussed in our analysis. **C**: Cluster.

C	Example Queries
3	• [word = "spridde" & ((int(_.text_datefrom) = 18950101 & int(_.text_timefrom) >= 094500) | (int(_.text_datefrom) > 18950101 & int(_.text_datefrom) <= 19051231)) & (int(_.text_dateto) < 19051231 | (int(_.text_dateto) = 19051231 & int(_.text_timeto) <= 235959))]
4	• [(word = "för.*" %c | word = ".*för" %c)] [(word = "olika.*" %c | word = ".*olika" %c)] [(word = "aktiviteter.*" %c | word = ".*aktiviteter" %c)] [(word = "i.*" %c | word = ".*i" %c)] [(word = "projektet.*" %c | word = ".*projektet" %c)]
5	• [(word = "har.*" %c | word = ".*har" %c)] [(word = "bra.*" %c | word = ".*bra" %c)] [(word = "sikt.*" %c | word = ".*sikt" %c)]
8	• [(lex contains "tät\.\.av\.1" | complemgram contains "tät\.\.av\.1\+.*" | complemgram contains ".*\+tät\.\.av\.1\+.*" | complemgram contains ".*\+tät\.\.av\.1:.*")]
62	• <sentence> [_.text_subject = "Kemi"] • [_.ne_ex = "TIMEX"]{1,1} • [_.blog_sex = "female"] [] • [_.ne_type = "LOC"]
64	• []dela • [^namn] • "[kK]att" • <date=2012> • f= year
66	• [word = "Ni" %c & _.text_year = "1920-1930"] • [word = "är" & _.text_approximate_level = "Avancerad"] • [word = "eller" & _.text_city = "Åbo"] </sentence> • [word = "kvart" & _.ne_type = "TME"] []
83	• []:LINKED_CORPUS:ASPACSVEN-EN [word = "dog"] • [word = "will"]:LINKED_CORPUS:ASPACSVEN-SV|EUROPARL-SV []
138	• [word = "eftersom"] • [word = "komplett"]

4.3.2.2 Homogeneity of Clusters

Homogeneity within clusters is another important aspect of the analysis. Homogeneous clusters, where the structure of queries is more similar and often even identical, are more reliable in reflecting user behavior for specific types of searches. In contrast, heterogeneous clusters, where there is significant variability in query structures, might not reliably represent a single user behavior or need.

The analysis reveals that Clusters 62 and 64 are particularly heterogeneous, with Cluster 64 being the most heterogeneous. This high level of heterogeneity can be attributed to the diverse range of query structures within these clusters.

- **Cluster 64** shows significant heterogeneity with a wide range of unique query structures, indicating that it encompasses a very diverse set of queries. For instance, examples from Cluster 64 include queries like the following ones which vary significantly in structure and complexity:

```
"ge"
<f="date 2012">
count hen by lemma
```

- **Cluster 62** also demonstrates a considerable degree of variability, but to a lesser extent compared to Cluster 64. Examples from Cluster 62 include queries such as the following ones, indicating searches that utilize specific text titles, macroglossary terms, and subject filters:

```
[_.text_title = "s"]
[macroglosses contains "INSTR"]
<sentence> [_.text_subject = "Biologi"]
```

On the other hand, the remaining clusters are relatively homogeneous, exhibiting consistent query structures. This homogeneity makes these clusters more reliable for understanding specific user needs, as they reflect a narrower range of search behaviors and requirements.

4.4 Query Execution Time Analysis

This section presents the findings from the analysis of query execution times, including error rates, timeouts, and the distribution of execution times.

Out of 1,401,260 query executions, 98.92% were successful, 1.07% encountered errors, and 0.01% resulted in timeouts.

Execution times varied, ranging from a minimum of 0.05 seconds to a maximum of 59.30 seconds. The average execution time across all queries was 0.48 seconds, with an SD of 1.61 seconds. This variation in execution times suggests that while most queries were processed quickly, some required longer processing times due to factors such as query complexity or the volume of data being processed.

The total execution time summed to 185 hours. However, with parallel execution, the actual processing time was reduced to 144 hours. This reduction may have

been influenced by factors such as the creation of CWB script files and associated input/output operations.

4.4.1 Distribution of Execution Times

The distribution of query executions across different execution time ranges is depicted in Figure 4.3. The majority of query executions (91.02%) were completed in less than one second, indicating that most queries were processed quickly. A smaller percentage of queries took slightly longer, with 5.84% completing between one to five seconds, 1.33% between five to ten seconds, and the remaining queries extending beyond ten seconds. This distribution suggests that while most queries are straightforward, a significant portion requires more complex processing.

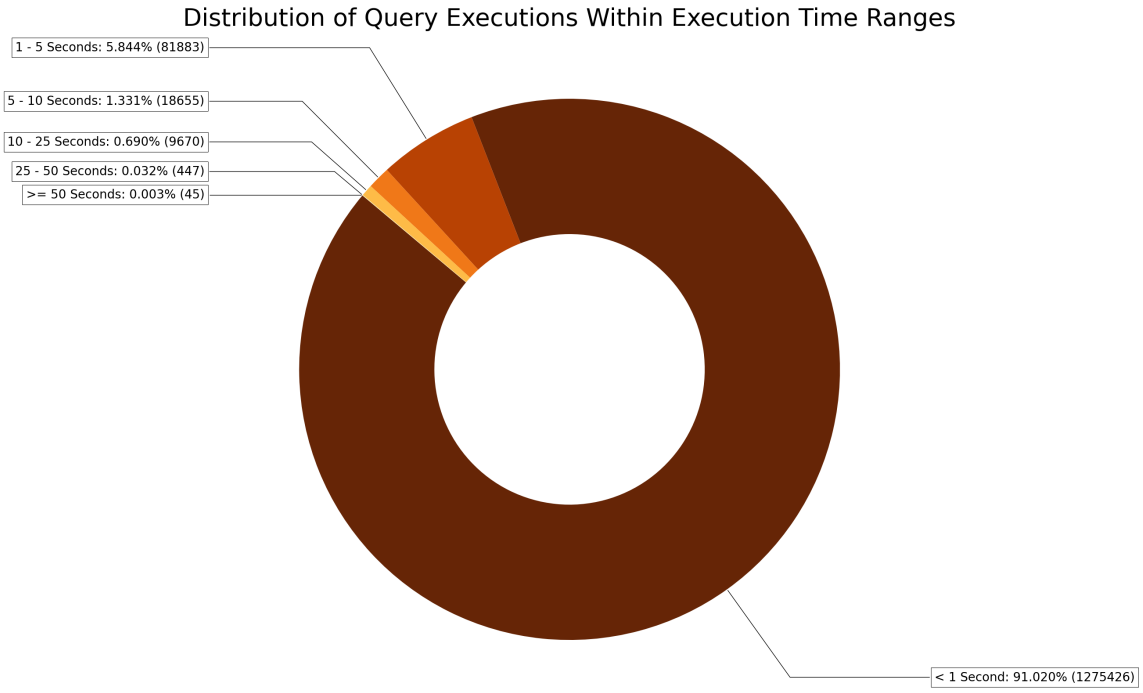

Figure 4.3: Distribution of query executions within execution time ranges. The labels show the percentage and number of query executions in each time range.

4.4.2 Cluster-Specific Execution Time Analysis

The analysis of cluster-specific execution times identified clusters with the highest and lowest mean execution time. The top 10 clusters with the highest mean execution time had an average of 3.65 seconds, while the top 10 clusters with the lowest mean execution time averaged 0.08 seconds. Cluster 62 had the highest mean execution time at 9.19 seconds, indicating that queries in this cluster were particularly complex or resource-intensive. In contrast, Cluster 138 had the lowest mean execution time at 0.08 seconds, reflecting efficient query execution, likely due to the simplicity of the queries.

Clusters with high mean execution time, such as Cluster 62, often involve more intricate query structures that utilize multiple attribute parameters, logical operators, and other complex elements. These clusters tend to be more heterogeneous, encompassing a variety of query types. For instance, queries in the slowest clusters, as detailed in Table 4.2, frequently include case insensitivity or diacritic ignoring flags and logical conditions, reflecting diverse search strategies and requirements. We provide one example query for each of these clusters because the remaining queries are structurally similar. Besides, additional examples have already been provided in Table 4.1.

Table 4.2: Top 10 clusters with the highest mean execution time, showcasing example queries that represent the structural characteristics of each cluster. **C**: Cluster, **MET**: Mean execution time of the entire cluster.

C	MET	Example Query
62	9.1902	`[_.text_date = "1820"]`
4	4.2800	`[(word = "lär.*" %c | word = ".*lär.*" %c | word = ".*lär" %c)] [(word = "sig.*" %c | word = ".*sig.*" %c | word = ".*sig" %c)] [(word = "skriva.*" %c | word = ".*skriva.*" %c | word = ".*skriva" %c)]`
7	3.5874	`[pos = "VB" & lex contains "läsa\.\.vb\.1"] [pos = "JJ" & pos = "AB"]{0,3} [(msd = ".*NN\.UTR\.SIN\.IND\.NOM.*" | msd = ".*NN\.NEU\.SIN\.IND\.NOM.*") & deprel = "OO"]`
22	3.4118	`[word = "i" %c] [word = "ett" %c] [word = "retoriskt" %c] [word = "samband" %c] [word = "med" %c]`
8	3.1850	`[(lex contains "udd\.\.nn\.1" | complemgram contains "udd\.\.nn\.1\+.*" | complemgram contains ".*\+udd\.\.nn\.1\+.*" | complemgram contains ".*\+udd\.\.nn\.1:.*")]`
23	2.9068	`[word = "precis" %c] [word = "som" %c] [word = "i" %c] [word = "fjol" %c]`
5	2.8753	`[(word = "flyga.*" %c | word = ".*flyga.*" %c | word = ".*flyga" %c)] [(word = "till.*" %c | word = ".*till.*" %c | word = ".*till" %c)]`
64	2.6349	`[katt]`
61	2.2733	`[pos = "NN" & lemma contains "eftervärld"]`
17	2.1156	`[pos = "PC" & lex contains "säga..vb.1"]`

Conversely, clusters with low mean execution time, such as Cluster 138, typically feature straightforward queries that search for specific words without additional conditions or complex patterns. These clusters exhibit a high degree of similarity among the queries, indicating homogeneity. For example, queries in the fastest clusters, as shown in Table 4.3, primarily consist of simple word searches without complex operators or additional parameters. Additionally, within each of these clusters, the queries have identical lengths. We provide one example query for each of these clusters because the remaining queries in each cluster are structurally similar, making additional examples unnecessary.

Table 4.3: Top 10 clusters with the lowest mean execution time, showcasing example queries that represent the structural characteristics of each cluster. **C**: Cluster, **MET**: Mean execution time of the entire cluster.

C	MET	Example Query
138	0.0801	`[word = "klassisk"]`
137	0.0804	`[word = "katastrof"]`
141	0.0805	`[word = "sätta"]`
134	0.0806	`[word = "nyliberalsim"]`
144	0.0806	`[word = "jämföra"]`
135	0.0806	`[word = "vattenflöde"]`
136	0.0806	`[word = "återkoppla"]`
145	0.0807	`[word = "vänlig"]`
151	0.0812	`[word = "karakteristisk"]`
150	0.0814	`[word = "ursprungligen"]`

4.4.3 Corpora-Specific Execution Time Analysis

The analysis of corpora-specific execution times reveals a clear relationship between data volume and execution time, as depicted in Figure 4.4. The data shows that as corpus size increases, there is a corresponding rise in the mean execution time of the entire corpus, highlighting the impact of corpus size on computational demand.

For example, the `flashback_livsstil` corpus, the largest in our study with 110,639,813 tokens, exhibits the highest mean execution time at 1.27 seconds. This suggests that the volume of data necessitates more processing, resulting in longer execution times. On the other hand, the `romii` corpus, with 4,304,271 tokens, has a mean execution time of 0.16 seconds, benefiting from its smaller size and faster processing.

This trend is consistent across the corpora examined. For instance, the `bloggmix2009` and `familjeliv_allmanna_noje` corpora, which are among the larger datasets with 75,116,677 and 90,124,289 tokens, respectively, show significantly higher mean execution time compared to smaller corpora like `gigaword_1960_69` or `sv_covid_19`. However, it is noteworthy that `familjeliv_allmanna_noje`, despite its large size, exhibits a lower mean execution time (0.93 seconds) than `bloggmix2009` (1.01 seconds), suggesting that factors beyond corpus size may also influence processing efficiency.

4.4.4 Error and Timeout Analysis

Error analysis in clusters, as visualized in Figure 4.5, revealed that Clusters 8, 62, 64, 66, and 83 exhibited the highest error counts, with Cluster 83 experiencing a 100% error rate. Cluster 8 had a high error rate of 28.84% due to the use of the `complemgram` attribute parameter, which is not supported by all the corpora in our analysis. Clusters 62 and 66 exhibited high error rates of 69.82% and 77.14%, respectively, due to queries containing corpus-specific attributes denoted by the pattern

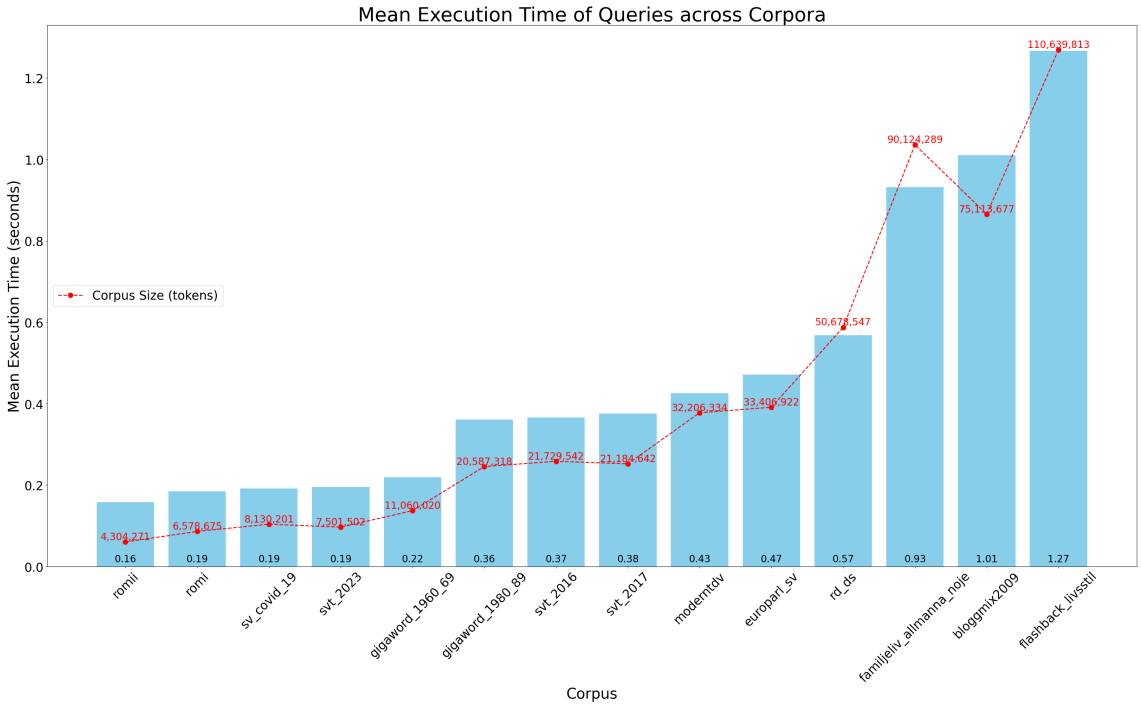

Figure 4.4: Bar chart illustrating the mean execution time (in seconds) of queries across the different corpora used in the performance evaluation, sorted by ascending size. Each bar represents a specific corpus, with the numeric value annotated at the bottom of each bar. The red points connected by a dashed line indicate the corpus sizes (in tokens) corresponding to each corpus.

'_.', which were not present in the corpora used, leading to frequent errors. Cluster 64 had an error rate of 86.16%, primarily due to the presence of malformed queries within the cluster. While the heterogeneous nature of the queries may contribute to the complexity, the main issue lies in the structure of the queries themselves, leading to a high incidence of errors. Finally, Cluster 83's 100% error rate was attributed to queries involving `':LINKED_CORPUS:'`, which specify particular corpora that were not available in our collection.

Figure 4.5: Number and percentage of errors in clusters containing one or more errors. The exact number of errors is shown above each bar, with the percentage (representing the proportion of errors relative to the total number of queries in that cluster) displayed above the number.

Similarly, the timeout analysis in clusters, presented in Figure 4.6, showed that timeouts were predominantly observed in Clusters 3, 4, and 5, with Cluster 3 having the highest timeout count of 72. Cluster 3 had a timeout rate of 2.17% because its queries require long processing times due to their length and complexity. These queries often involve retrieving results within specific date ranges (e.g., `int(_.text_timefrom) >= 094500`) and using multiple logical operators. Clusters 4 and 5 had timeout rates of 1.16% and 0.77%, respectively, because they both include long queries with many specific regex patterns, in addition to using multiple logical operators and case insensitivity or diacritic ignoring flags.

For a detailed, cluster-level breakdown of execution counts, errors and timeouts, and their respective rates for all affected clusters, refer to Appendix C.

4.5 Benchmark Analysis

This section presents the results and analysis of the statistical approach detailed in Section 3.6. The analysis includes metrics such as execution success rates and the

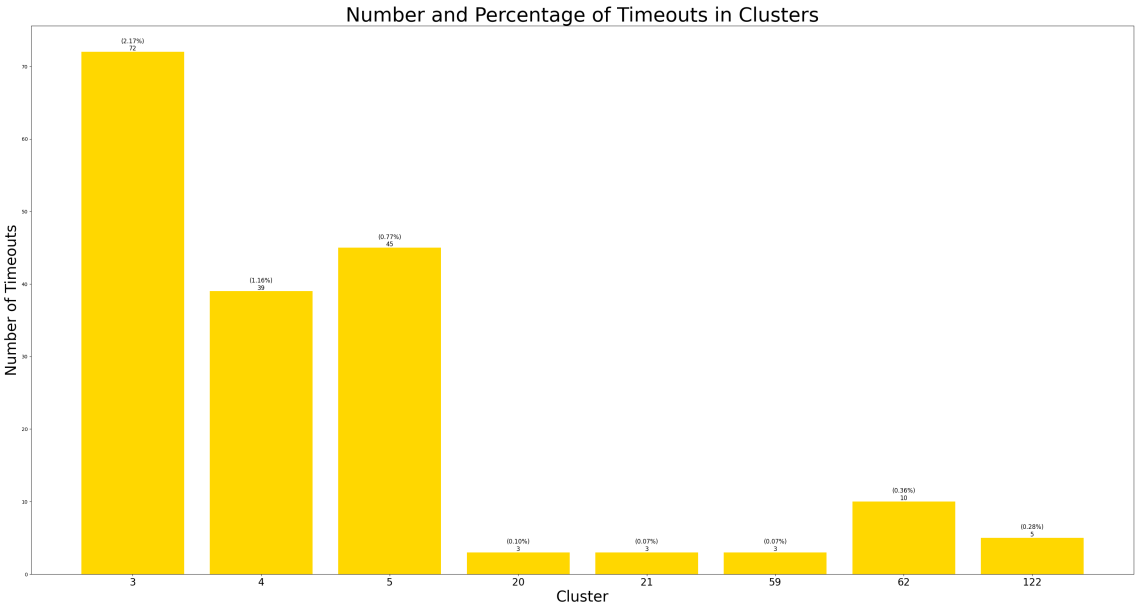

Figure 4.6: Number and percentage of timeouts in clusters containing one or more timeouts. The exact number of timeouts is shown above each bar, with the percentage (representing the proportion of timeouts relative to the total number of queries in that cluster) displayed above the number.

Coefficient of Variation (CV) of execution times for clusters and their representative queries. These metrics provide insight into how corpora search systems handle various queries in terms of efficiency and reliability.

The results of the benchmark development process, calculated before the manual refinement process, are detailed in Appendix D. To aid visual analysis, Figure 4.7 illustrates the CV for the execution times per cluster with its representative query. This visualization highlights the variability in performance across different clusters.

4.5.1 Evaluation of Execution Success Rates

As shown in Figure 4.8, the analysis of query execution success rates for each cluster revealed that 118 clusters achieved a perfect execution success rate of 100%, indicating that all queries within these clusters were executed successfully. Clusters with high execution success rates included 30 clusters with a success rate between 90% and 100%, 4 clusters with a success rate between 80% and 90%, and 4 clusters with a success rate between 70% and 80%. Clusters with lower execution success rates were less common. Specifically, Clusters 62 and 66 had success rates of approximately 30% and 23%, respectively, while Cluster 64 exhibited a success rate of around 14%. Notably, Cluster 83 had a 0% execution success rate, as all queries failed to execute, as discussed in Section 4.4.4.

These findings indicate that most clusters achieved high success rates, with a significant number attaining flawless execution, while a small fraction faced more substantial execution challenges.

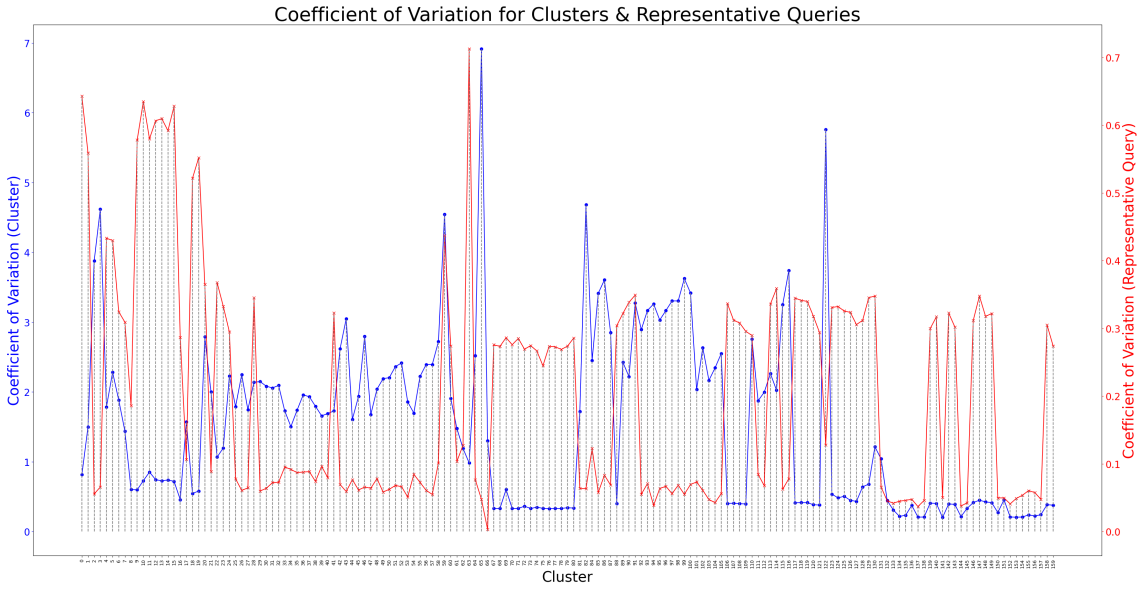

Figure 4.7: The coefficient of variation for the execution times per cluster (blue line) and the coefficient of variation of the representative query within each cluster (red line). The blue line on the left vertical axis reflects the overall variability in cluster performance, while the red line on the right vertical axis indicates the consistency of execution times for the representative query.

4.5.2 Analysis of Coefficient of Variation for Clusters

As shown in Figure 4.9, the analysis of the CV for clusters provides insights into the variability of query execution times. The distribution of clusters based on their CV shows that 72 clusters have a CV under 1, indicating relatively consistent performance. Additionally, 32 clusters fall between a CV of 1 and 2, suggesting moderate variability in execution times. A larger set of 50 clusters exhibit a CV between 2 and 4, reflecting higher variability. Finally, there are 5 clusters with a CV greater than 4, indicating significant execution time variability within these clusters.

Notably, Cluster 65 stands out with the highest CV of 6.91, indicating extreme variability in execution times. This cluster has a mean execution time of 0.2 seconds and an SD of 1.38 seconds, suggesting that queries in this cluster experience highly inconsistent performance. On the other hand, Cluster 141 has the lowest CV of 0.2, reflecting highly consistent execution times with a mean of 0.08 seconds and an SD of 0.02 seconds.

These findings indicate that the majority of clusters exhibit consistent and stable query execution times, reflecting low variability. However, a minority of clusters show higher variability, indicating unpredictable or inconsistent performance in those cases.

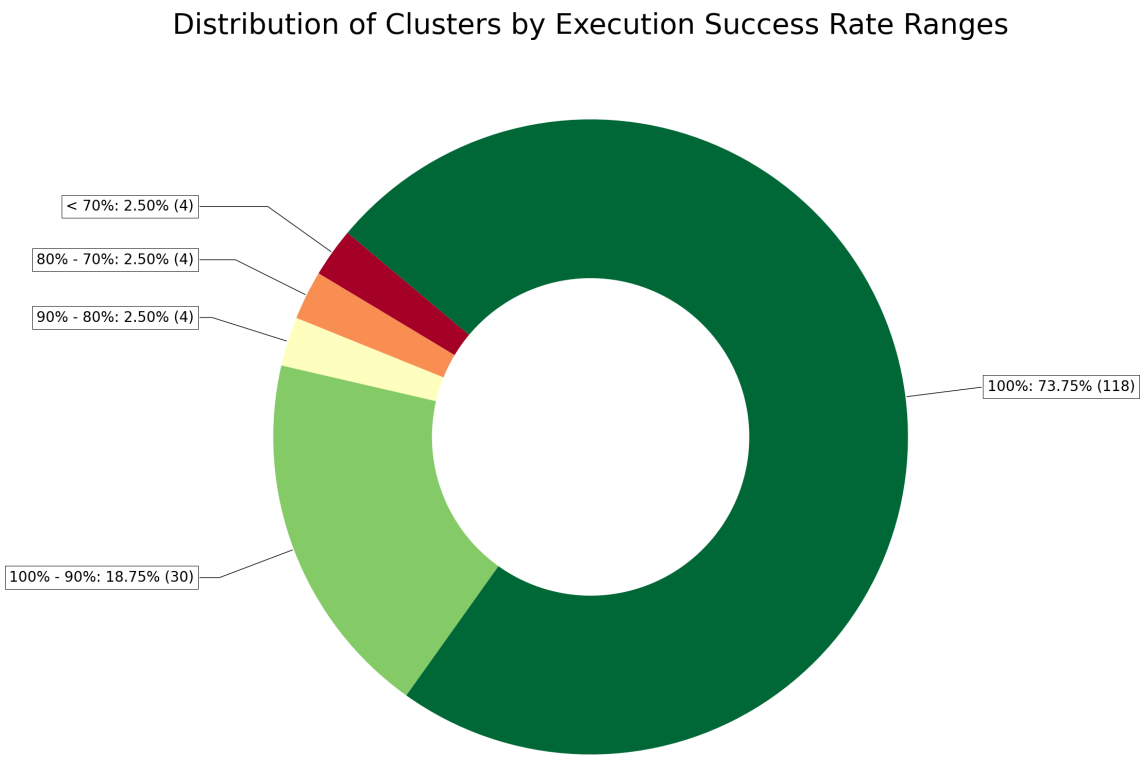

Figure 4.8: Distribution of clusters based on execution success rate ranges. The labels indicate the success rate range, the percentage of total clusters within that range, and the corresponding number of clusters.

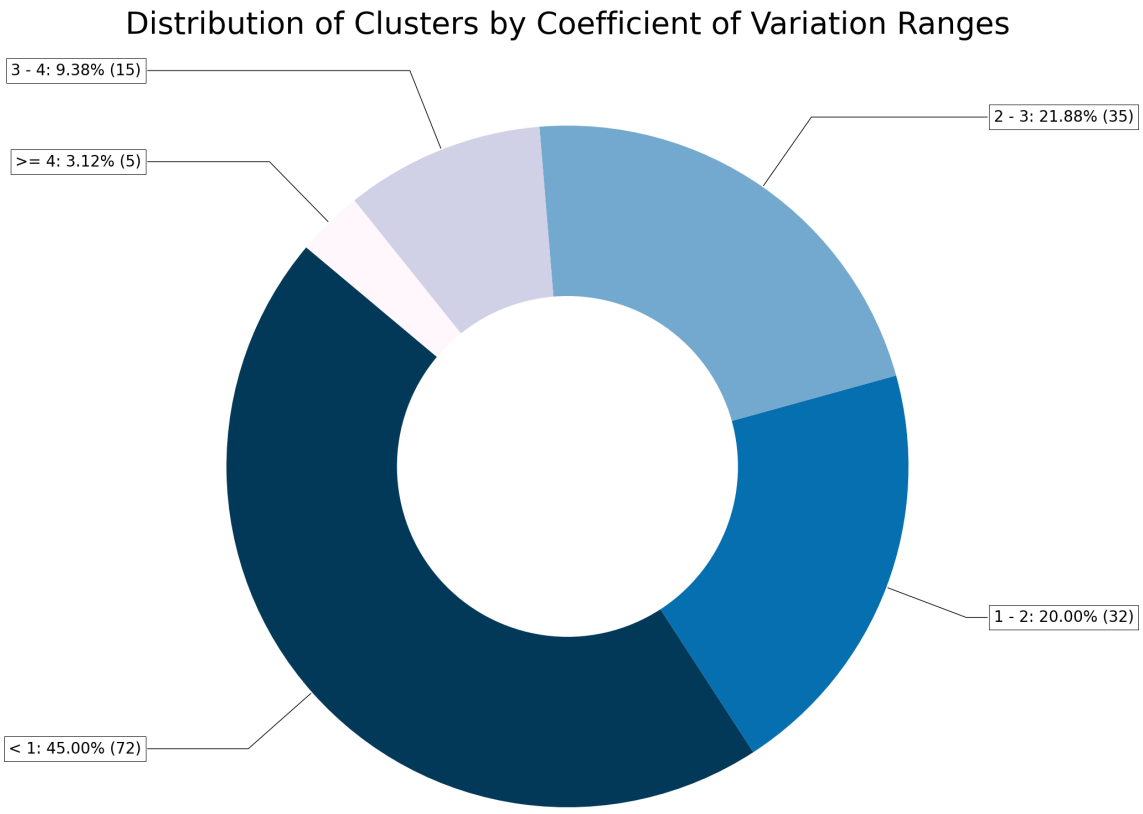

Figure 4.9: Distribution of clusters based on coefficient of variation ranges. The labels indicate the coefficient of variation range, the percentage of total clusters within that range, and the corresponding number of clusters.

4.5.3 Analysis of Coefficient of Variation for Representative Queries

As shown in Figure 4.10, the CV for representative queries also shows a wide range of variability. Seventy-nine representative queries have a CV under 0.1, indicating highly consistent execution times. Sixty-eight queries fall within the range of 0.1 to 0.5, suggesting generally stable performance with minor variability. Twelve representative queries exhibit a CV between 0.5 and 0.71, pointing to moderate variability in execution times. No representative queries were found to have a CV greater than 0.71, indicating that extreme variability is rare among the representative queries.

The representative query from Cluster 63 has the highest CV of 0.71, with a mean execution time of 1.48 seconds and an SD of 1.06 seconds, indicating higher variability in performance. In contrast, the representative query from Cluster 66 has the lowest CV, with a value of 0.003. This query shows a mean execution time of around 0.1 seconds and an SD of 0.0003 seconds, reflecting near-perfect consistency.

These findings suggest that most representative queries exhibit consistent execution times, with only a small fraction experiencing slightly higher variability.

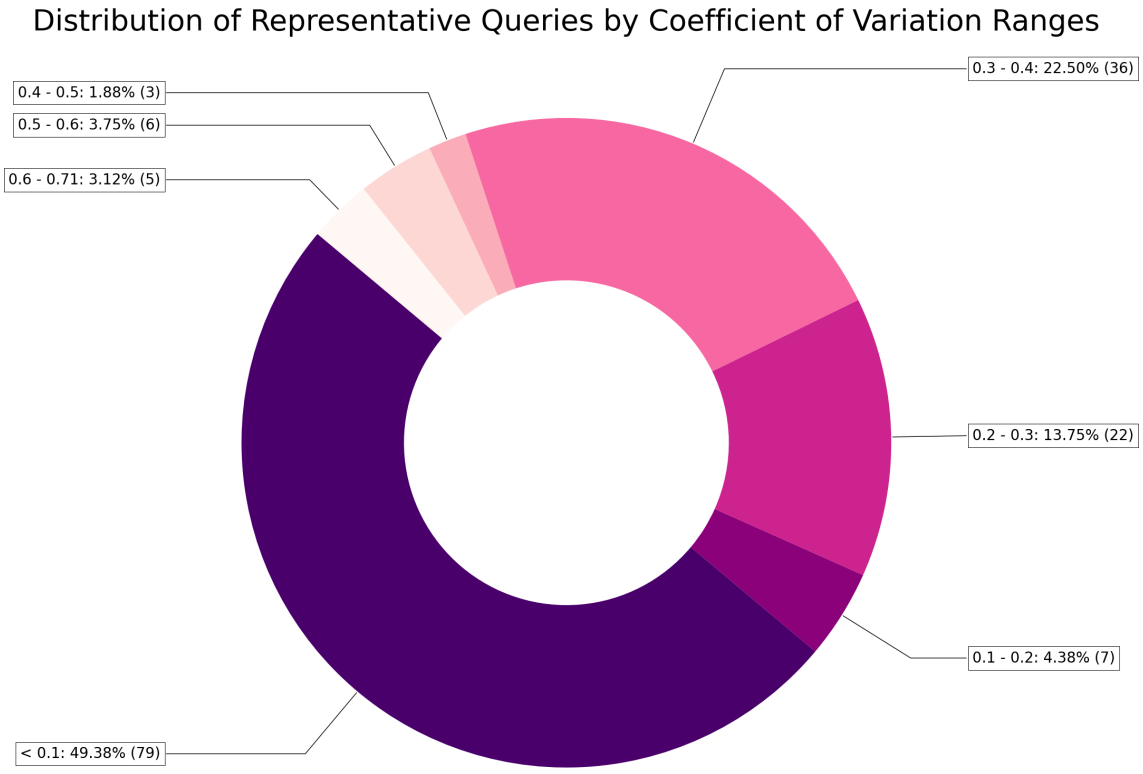

Figure 4.10: Distribution of representative queries based on coefficient of variation ranges. The labels indicate the coefficient of variation range, the percentage of total queries within that range, and the corresponding number of queries.

4.5.4 Analysis of Benchmark Refinement

This section presents the results of our analysis, which aimed to group clusters with similar query structures to enhance the clarity and compactness of our benchmark, as outlined in Section 3.6.2. By consolidating clusters with minor differences—such as varying attribute value lengths but similar overall structures—we achieved a more streamlined and representative benchmark.

Initially, we analyzed the entire set of 160 clusters. Four clusters (62, 64, 66, and 83) were excluded due to high heterogeneity and error rates, as detailed in Sections 4.3.2.2 and 4.4.4. Of the remaining clusters, 124 were merged into 12 groups based on structural similarities, while 32 clusters were left ungrouped due to their distinct characteristics, resulting in 44 final groups.

Rather than using all 160 clusters in the benchmark, the grouping reduced the number of entities used in the final benchmark to 44 groups, a 72.5% reduction in the number of clusters represented. The final refinement retained 77.5% of the clusters in grouped form, while 20% remained as standalone entities. The exclusion of the four outliers constituted only 2.5% of the original data, ensuring minimal loss of information.

Table 4.4 provides a summary of cluster groupings based on token and attribute structures. This table categorizes clusters by the number of tokens and the characteristics of the attribute parts, including parameters, operators, and the use of regex or ignore case flags. Additionally, the table includes clusters that were not part of any group but are assigned unique identification numbers for clarity and reference purposes.

Table 4.4: Summary of cluster groupings based on token and attribute structures. The table categorizes clusters by the number of tokens and the characteristics of the attribute parts, including parameters, operators, and the use of regular expressions or ignore case flags. Individual clusters that were not included in any grouping are presented together in a compact format in the last row of the table. The "Grouped Clusters" column lists the original cluster identification numbers, and these are assigned new identification numbers (12-43) in the same order. For clusters in this row, structural information is intentionally omitted, denoted by a dash `'-'`.

ID	Grouped Clusters	Token Structure	Attribute Structure
0	131-8, 141, 144-5, 150-7	1 token of 1 attribute part	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values
1	88, 91, 106-9, 117-130, 148-9	1 token of 1 attribute part	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values with regex

Continued on next page

Table 4.4 – continued from previous page

ID	Grouped Clusters	Token Structure	Attribute Structure
2	139-140, 142-3, 146-7, 158-9	1 token of 1 attribute part	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values with ignore case flag `'%c'`
3	92-105, 111-2, 115-6	2 tokens of 1 attribute part each	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values
4	84, 86	2 tokens of 1 attribute part each	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values with regex
5	89-90, 113-4	2 tokens of 1 attribute part each	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values with ignore case flag `'%c'`
6	43-57	3 tokens of 1 attribute part each	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values
7	29-39	4 tokens of 1 attribute part each	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values
8	65, 87	1 token of 2 attribute parts connected by logical operator	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values
9	10-15	1 token of 3 attribute parts connected by 3 logical operators	`'word'` as attribute parameter, `'='` as attribute operator, varying attribute values with ignore case flag `'%c'` and regex
10	16, 67-80	1 token of 1 attribute part	`'lex'` as attribute parameter, `'contains'` as attribute operator, varying attribute values
11	18-9	1 token of 2 attribute parts connected by logical operator	1st: `'lex'` as attribute parameter, `'contains'` as attribute operator; 2nd: `'complemgram'` as attribute parameter, `'contains'` as attribute operator, varying attribute values with regex
12-43	**0-9, 17, 20-28, 40-42, 58-61, 63, 81-82, 85, 110**	-	-

The analysis of the grouped clusters revealed consistently high execution success rates. The CVs exhibited a relatively narrow range, with very few clusters showing slightly higher CVs, indicating greater variability within those clusters. Similarly, the mean execution time for the representative queries varied slightly across clusters within their groups, with some taking a bit longer on average. The CV for the representative queries also displayed minimal variability, reflecting almost no differences in the stability of execution times. These observations suggest that our decision to select the representative cluster for each group based on the lowest CV does not compromise the overall quality of the benchmark, as the metrics exhibit similar values across the grouped clusters.

The refined benchmark, presented in Appendix E, provides a more representative evaluation of the performance of corpora search systems.

5

Conclusion

This chapter summarizes the key findings and implications of our study on developing a standardized benchmark for evaluating corpora search systems in digital humanities, based on data from Korp server logs.

5.1 Discussion

The development of a benchmark for evaluating corpora search systems in digital humanities was guided by the need to align with current researcher requirements and system functionalities. Analyzing Korp server logs provided empirical insights into user query patterns, informing the design of this benchmark. This section discusses the motivations behind key decisions and addresses findings not extensively covered in previous chapters.

5.1.1 Leveraging Korp Logs for Benchmarking

The focus on Korp server logs was motivated by their capacity to provide a snapshot of real-world usage. Korp offered a dataset relevant to developing a benchmark that reflects current research needs. The decision to aggregate queries based on exact string matches, rather than identifying subsets or variations, was influenced by the need to maintain the specificity and integrity of user queries. This approach aimed to accurately represent distinct search behaviors.

However, relying on Korp server logs for query analysis has certain limitations. The data, collected from October 2022 to January 2024, represents a specific time frame, and the types of queries captured within this period may lack diversity. Consequently, the findings and the resulting benchmark may not fully encompass the entire range of query types and complexities that researchers might encounter. As research trends evolve and new methodologies emerge, the relevance of these query patterns may change, potentially affecting the benchmarks applicability over time. Ongoing updates and analyses of query logs will be necessary to ensure that the benchmark remains current and relevant to contemporary research needs.

5.1.2 Query Abstraction: Balancing Complexity and Generalizability

The two-level abstraction process for CQP queries was a decision aimed at balancing complexity and generalizability. The first level of abstraction retained essential structural elements while simplifying the data, facilitating detailed pattern analysis. This step was important for preserving the functional integrity of the queries, particularly the regex patterns, which are useful for understanding the logical structure of user queries. The second level of abstraction further generalized the queries, focusing on core structural components and the connections within and between them. This approach aimed to ensure accuracy and reliability in pattern detection, which is necessary for clustering and ML feature extraction. By applying regex patterns at specific abstraction levels, as discussed in Appendix A, we enhanced the extraction of ML features while managing the complexity of the patterns involved.

5.1.3 Choosing Hierarchical Density-Based Spatial Clustering of Applications with Noise for Robust Clustering

The selection of HDBSCAN for clustering was guided by its ability to handle varying data densities and its hierarchical clustering approach. This algorithm was well-suited for the diverse nature of the query data, allowing for the identification of stable clusters without needing predefined cluster numbers. The iterative optimization of the minimum cluster size hyperparameter, guided by the DBCV score, was important for achieving robust clustering performance. This decision highlighted the need to adapt clustering parameters to the datasets characteristics, ensuring coherent and meaningful groupings.

5.1.4 Insights from Query Execution on Corpus Workbench: Evaluation and Challenges

The performance evaluation of queries in CWB using selected corpora provided insights into the efficiency and reliability of CWB. The variation in execution times across different corpora highlighted the impact of corpus size and complexity on query performance. While the selected corpora provided a basis for testing, it is important to note that many real-world corpora have significantly larger token sizes than those used in our testing. These larger corpora require more processing power and time, potentially exacerbating the performance issues observed in this study.

Additionally, many corpora search systems allow users to select multiple corpora for a single CQP query, which increases the processing load as the system must handle the combined size and complexity of all chosen corpora. This aspect was not fully captured in our testing environment, where queries were evaluated against individual corpora. Benchmarking queries across multiple corpora simultaneously is important for a more comprehensive evaluation of corpora search systems, as it more accurately reflects actual usage patterns.

5.1.5 Challenges with Execution Errors and Timeouts

The analysis of errors and timeouts in query executions revealed clusters with notable challenges, such as Clusters 62, 64, 66, and 83. These clusters exhibited high error rates due to the use of corpus-specific attributes and linked corpus specifications, which were absent in the selected corpora. This insight underscores the importance of including a broad range of corpora in future benchmarking efforts to ensure a more comprehensive evaluation and identification of potential system limitations.

5.1.6 Motivation for Using Coefficient of Variation for Cluster and Query Evaluation

In evaluating query execution times, CV is a useful statistical measure because it normalizes variability relative to the mean, allowing comparisons across clusters with different scales. Unlike SD, which measures absolute variability, CV provides a ratio of SD to the mean, offering insight into relative variability. This is particularly helpful for comparing clusters where the mean execution times differ.

By using CV, we can better identify clusters with consistent query execution times, which is important for selecting representative queries for benchmarking. Queries with a low CV indicate more stable performance across different corpora, making them reliable for evaluating system efficiency. This approach ensures that the benchmark reflects not only the overall performance but also the consistency of query execution times across clusters.

5.1.7 Motivation for Refining the Benchmark through Cluster Grouping

The decision to refine the benchmark by grouping clusters with similar query structures was driven by the need to enhance the clarity and efficiency of our evaluation framework. Our initial analysis revealed that while individual clusters offered valuable insights into specific query types, many clusters differed only in minor structural aspects, such as attribute value lengths. This redundancy unnecessarily complicated the evaluation process without providing meaningful distinctions, underscoring the importance of consolidating clusters based on significant structural similarities.

To address this, we grouped clusters that shared core structural characteristics, despite superficial differences. This consolidation resulted in a more streamlined and representative benchmark, reducing data fragmentation and making the evaluation process more meaningful and easier to interpret.

5.1.8 Alternative Evaluation Methods for Corpora Search Systems

The development of a benchmark for evaluating corpora search systems was a key aspect of this study. The final benchmark focused on query execution times as the sole metric for assessing system performance. While this approach provides valuable

insights into the speed of the systems, it is important to acknowledge that other evaluation metrics, such as precision and user satisfaction, could have been considered to offer additional perspectives. These alternative metrics would complement the current framework by assessing accuracy and user experience, thereby providing a more comprehensive evaluation of the system's overall performance.

Overall, while the Korp server logs provide insights for benchmark development, ongoing attention and adaptation will be necessary to maintain the benchmarking framework's relevance and effectiveness in the evolving landscape of digital humanities.

5.2 Conclusion

This study developed a benchmarking framework for evaluating corpora search systems within the digital humanities context. By analyzing real-world queries from the Korp server logs, this thesis provided insights into user search behaviors, query complexities, and system performance.

The clustering of user queries using the HDBSCAN algorithm revealed variations in query structures and complexities, highlighting the need for tailored benchmarks that account for both common and specialized user needs. The analysis of query execution times across selected corpora emphasized the impact of corpus size and complexity on performance, underlining the importance of scalability in corpora search systems.

The development of a benchmark for corpora search systems focused on assessing the efficiency of these systems. The use of query execution times as a metric provided a clear perspective on system performance, facilitating comparisons across different query complexities and corpora.

Overall, this thesis contributes to an understanding of the challenges and opportunities in evaluating corpora search systems, aiming to improve the development of digital research tools in the humanities.

5.2.1 Future Work

Future work should focus on several key areas to enhance the benchmarking framework:

First, incorporating a wider range of corpora, including those with significantly larger token sizes, will provide a more comprehensive evaluation of corpora search systems. This expansion will also facilitate the benchmarking of queries executed across multiple corpora simultaneously, reflecting real-world usage scenarios more accurately.

Second, integrating user experience metrics alongside computational metrics will offer a holistic view of corpora search systems effectiveness, addressing the practical needs of humanities scholars. Additionally, exploring advanced NLP techniques for query analysis and categorization could improve the precision of benchmark results.

Furthermore, the reliance on Korp server logs for query analysis may limit the scope of findings to systems using CQP as a query language. Future research should consider expanding the dataset to include other corpora search systems and query languages, broadening the benchmark's applicability.

The diversity of digital humanities queries and corpora search systems presents challenges in standardizing performance metrics. Continued efforts to refine these metrics will aim to ensure equitable evaluations across varied systems and architectures. Lastly, as the field evolves, the benchmarking framework will require updates to remain relevant and useful in evaluating corpora search systems' performance.

Bibliography

[1] D. M. Berry, "Introduction: Understanding the digital humanities," in *Understanding Digital Humanities*. London: Palgrave Macmillan UK, 2012, pp. 1–20, ISBN: 978-0-230-37193-4. DOI: 10.1057/9780230371934_1. [Online]. Available: https://doi.org/10.1057/9780230371934_1.

[2] C. L. Borgman, "The digital future is now: A call to action for the humanities," *Digital humanities quarterly*, vol. 3, no. 4, 2010.

[3] J. Kekäläinen, *The effects of query complexity, expansion and structure on retrieval performance in probabilistic text retrieval*. Citeseer, 1999.

[4] C. Biemann, G. R. Crane, C. D. Fellbaum, and A. Mehler, "Computational Humanities - bridging the gap between Computer Science and Digital Humanities (Dagstuhl Seminar 14301)," *Dagstuhl Reports*, vol. 4, no. 7, C. Biemann, G. R. Crane, C. D. Fellbaum, and A. Mehler, Eds., pp. 80–111, 2014, ISSN: 2192-5283. DOI: 10.4230/DagRep.4.7.80. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/DagRep.4.7.80.

[5] W. Gao, Y. Zhu, Z. Jia, *et al.*, "Bigdatabench: A big data benchmark suite from web search engines," *arXiv preprint arXiv:1307.0320*, 2013.

[6] B. McGillivray, T. Poibeau, and P. Ruiz Fabo, "Digital humanities and natural language processing je taime," *Moi non plus.(Alliance of Digital Humanities, 2020)*, 2020.

[7] L. Borin, M. Forsberg, and J. Roxendal, "Korp the corpus infrastructure of språkbanken," in *Proceedings of LREC 2012. Istanbul: ELRA*, vol. Accepted, 2012, pp. 474–478.

[8] S. Evert and A. Hardie, "Twenty-first century corpus workbench: Updating a query architecture for the new millennium," in *Proceedings of the Corpus Linguistics 2011 conference*, Citeseer, 2011, pp. 1–21.

[9] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: An introduction," *Journal of the American Medical Informatics Association*, vol. 18, no. 5, pp. 544–551, 2011.

[10] S. Badillo, B. Banfai, F. Birzele, *et al.*, "An introduction to machine learning," *Clinical pharmacology & therapeutics*, vol. 107, no. 4, pp. 871–885, 2020.

[11] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining*, Springer, 2013, pp. 160–172.

[12] D. Moore, G. McCabe, and B. Craig, *Introduction to the Practice of Statistics*. W. H. Freeman, 2014, ISBN: 9781464133633. [Online]. Available: https://books.google.se/books?id=pX1_AwAAQBAJ.

[13] Y. Dodge, *The concise encyclopedia of statistics*. Springer Science & Business Media, 2008.

[14] S. T. Gries and A. L. Berez, "Linguistic annotation in/for corpus linguistics," in *Handbook of Linguistic Annotation*, N. Ide and J. Pustejovsky, Eds., Dordrecht: Springer, 2017, pp. 379–409. [Online]. Available: https://api.semanticscholar.org/CorpusID:15981453.

[15] T. Nygren, A. Foka, and P. I. Buckland, "The status quo of digital humanities in sweden: Past, present and future of digital history," *H-Soz-Kult*, 2014.

[16] S. Evert and T. C. D. Team, *The ims open corpus workbench (cwb) - cqp interface and query language manual*, Online, Accessed: 2024-04-11, 2022. [Online]. Available: https://cwb.sourceforge.io/files/CQP_Manual.pdf.

[17] J. Nivre, M.-C. De Marneffe, F. Ginter, *et al.*, "Universal dependencies v1: A multilingual treebank collection," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 2016, pp. 1659–1666.

[18] A. Kilgarriff, V. Baisa, J. Buta, *et al.*, "The sketch engine: Ten years on," *Lexicography*, vol. 1, no. 1, pp. 7–36, 2014.

[19] T. Krause and A. Zeldes, "ANNIS3: A new architecture for generic corpus query and visualization," *Digital Scholarship in the Humanities*, vol. 31, no. 1, pp. 118–139, Oct. 2014, ISSN: 2055-7671. DOI: 10.1093/llc/fqu057. eprint: https://academic.oup.com/dsh/article-pdf/31/1/118/21518095/fqu057.pdf. [Online]. Available: https://doi.org/10.1093/llc/fqu057.

[20] L. Anthony, "Antconc: Design and development of a freeware corpus analysis toolkit for the technical writing classroom," in *IPCC 2005. Proceedings. International Professional Communication Conference, 2005.*, 2005, pp. 729–737.

[21] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–51, 2015.

[22] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, and J. Sander, "Density-based clustering validation," in *Proceedings of the 2014 SIAM international conference on data mining*, SIAM, 2014, pp. 839–847.

[23] and others, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, vol. 1, 1967, pp. 281–297.

[24] L. Rokach and O. Maimon, "Clustering methods," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. Boston, MA: Springer US, 2005, pp. 321–352, ISBN: 978-0-387-25465-4. DOI: 10.1007/0-387-25465-X_15. [Online]. Available: https://doi.org/10.1007/0-387-25465-X_15.

[25] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14, MIT Press, 2001. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2001/file/801272ee79cfde7fa5960571fee36b9b-Paper.pdf.

[26] P. J. ROUSSEEUW and C. CROUX, "Alternatives to the median absolute deviation," *Journal of the American Statistical Association*, vol. 88, no. 424, p. 1273, 1993.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] L. McInnes, J. Healy, and S. Astels, "Hdbscan: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, no. 11, Mar. 2017. DOI: 10.21105/joss.00205. [Online]. Available: https://doi.org/10.21105%2Fjoss.00205.

[29] A. Martelli, A. Ravenscroft, and D. Ascher, *Python cookbook*. " O'Reilly Media, Inc.", 2005.

[30] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[31] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[32] M. L. Waskom, "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. DOI: 10.21105/joss.03021. [Online]. Available: https://doi.org/10.21105/joss.03021.

[33] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[34] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

A

Machine Learning Features for Corpus Query Processor Analysis

This appendix details the ML features used in the analysis of CQP queries. These features are important for understanding the interactions within the Korp server logs, as listed in Table A.1.

The naming conventions for the IDs listed in the table are systematically structured as follows:

- The prefixes **'cqp'**, **'cqp__1'**, and **'cqp__2'** indicate that the statistic is derived from the original CQP, Level 1 abstraction of CQP, and Level 2 abstraction of CQP, respectively.
- The insertion of **'n'** denotes that the statistic's value is a numerical count, specifically a positive integer.
- The suffixes **'w__a__v'** and **'i__a__v'** clarify how attribute values are factored into the counts:
  - **'w__a__v'** (without attribute values): This suffix indicates that the statistic is calculated by excluding the contents within each attribute's value, focusing instead on elements outside of the attribute values.
  - **'i__a__v'** (in attribute values): This suffix specifies that the statistic is calculated exclusively from the contents within each attribute's value, focusing on these elements.

Table A.1: Comprehensive list of quantified machine learning features derived from the analysis of Korp queries.

ID	Description
cqp__n__chars	Total count of characters from the original CQP.
cqp__1__n__quantifiers__w__a__v	Total count of quantifiers including specific repetition ranges (`{d,d}`, `{d,}`, `{d}`, `{x,y}`), and symbols (`*`, `+`, `?`) at abstract level 1, excluding attribute values.

Continued on next page

Table A.1 – continued from previous page

ID	Description
cqp_1_n_quantifiers_i_a_v	Total count of quantifiers including specific repetition ranges (`{d,d}`, `{d,}`, `{d}`, `{x,y}`), and symbols (`*`, `+`, `?`) at abstract level 1, derived from attribute values only.
cqp_2_n_ignore_case_diacritics_flags_w_a_v	Total count of ignore case and diacritics flags (`%c`, `%d`, `%cd`, `%dc`) at abstract level 2, excluding attribute values.
cqp_1_n_lookaround_i_a_v	Total count of regex positive and negative lookahead and lookbehind assertions at abstract level 1, derived from attribute values only.
cqp_1_n_logical_operators_w_a_v	Total count of logical OR `'|'`, AND `'&'`, and NOT `'!'` operators at abstract level 1, excluding attribute values.
cqp_1_n_logical_operators_i_a_v	Total count of logical OR `'|'`, AND `'&'`, and NOT `'!'` operators at abstract level 1, derived from attribute values only.
cqp_2_n_sentence_start_end_w_a_v	Total count of sentence start and end markers (`<s>`, `</s>`, `<sentence>`, `</sentence>`) at abstract level 2, excluding attribute values.
cqp_2_n_noun_phrase_start_end_w_a_v	Total count of noun phrase start and end markers (`<np>`, `</np>`) at abstract level 2, excluding attribute values.
cqp_2_n_within_restrict_w_a_v	Total count of `'within'` keyword used for restrictions at abstract level 2, excluding attribute values.
cqp_2_n_zero_width_assertions_w_a_v	Total count of zero-width assertions, i.e., patterns including `'[:'`, at abstract level 2, excluding attribute values.
cqp_1_n_group_labeling_w_a_v	Total count of group labeling, i.e., patterns including `':['`, at abstract level 1, excluding attribute values.

Continued on next page

Table A.1 – continued from previous page

ID	Description
cqp_1_n_group_labeling_i_a_v	Total count of group labeling, i.e., patterns including `':['`, at abstract level 1, derived from attribute values only.
cqp_2_n_target_marker_w_a_v	Total count of target markers, i.e., patterns including `'@['`, at abstract level 2, excluding attribute values.
cqp_2_n_attribute_parts_w_a_v	Total count of attribute parts, i.e., patterns including `'a_p'`, at abstract level 2, excluding attribute values.
cqp_1_n_grouping_w_a_v	Total count of groupings within parentheses `'( )'` at abstract level 1, excluding attribute values.
cqp_1_n_grouping_i_a_v	Total count of groupings within parentheses `'( )'` at abstract level 1, derived from attribute values only.
cqp_2_n_grouping_depth_w_a_v	The number representing the maximum depth of nested grouping within parentheses `'( )'` at abstract level 2, excluding attribute values.
cqp_2_n_linked_corpus_w_a_v	Total count of linked corpus, i.e., patterns including `':LINKED_CORPUS:'`, at abstract level 2, excluding attribute values.
cqp_1_n_regex_wildcards_i_a_v	Total count of regex wildcards and special characters (`.`, `^`, `$`, `*`, `+`, `?`, `{,` `}`, `[`, `]`, `\`, `|`, `(`, `)`, `-`, `=`, `!`, `<`, `>`, `,`, `:`, `;`, `/`, `&`, `%`, `#`, `@`, `‘`, `~`, `"`, `’`, `\s`, `\S`, `\w`, `\W`, `\d`, `\D`, `\b`, `\n`, `\r`, `\t`, `\0`, `\v`) at abstract level 1, derived from attribute values only.
cqp_2_n_tokens_w_a_v	Total count of CQP tokens, i.e., patterns including `'['`, at abstract level 2, excluding attribute values.

B

Corpora Utilized for Evaluating Query Execution Times

This appendix offers an overview of the various corpora employed to evaluate the execution times of all clustered CQP queries in CWB. Each corpus is accompanied by a concise description, providing clarity on its specific content and relevance, as listed in Table B.1. All the corpora were downloaded from the resources page of the Språkbanken website, accessible at `https://spraakbanken.gu.se/en/resources`.

Table B.1: Full list of the 14 corpora used to measure query execution times in Corpus Workbench.

ID	Tokens	Description
bloggmix2009	75,113,677	`Blog mix 2009`—Material from a selection of the most visited Swedish blogs in 2009.
europarl_sv	33,406,922	`Europarl: Swedish`—The Swedish part of European Parliament Proceedings Parallel Corpus.
familjeliv_allmanna_noje	90,124,289	`Familjeliv: General Threads Entertainment`—Material from the the Entertainment section in the Familjeliv internet forum.
flashback_livsstil	110,639,813	`Flashback: Lifestyle`—Material from the Lifestyle section in the Flashback internet forum.
gigaword_1960_69	11,060,020	`The Swedish Culturomics Gigaword Corpus 1960-1969`—A Swedish reference dataset for NLP for the years 1960 to 1969.
gigaword_1980_89	20,587,318	`The Swedish Culturomics Gigaword Corpus 1980-1989`—A Swedish reference dataset for NLP for the years 1980 to 1989.
moderntdv	32,206,334	`Judgements`—A contemporary Swedish language corpus consisting of court judgments and legal texts.
rd_ds	50,678,547	`Riksdagen's Open Data: Department Series`—Investigations from government departments.
romi	6,578,675	`Bonnier novels I (197677)`—A corpus of 69 Bonnier novels from 197677.

Continued on next page

Table B.1 – continued from previous page

ID	Tokens	Description
romii	4,304,271	`Bonnier novels II (198081)`—A corpus of 60 Bonnier novels from 198081.
sv_covid_19	8,130,201	`Swedish Covid-19`—A collection of Swedish news texts, scientific and popular science articles and articles from certain blogs and social media such as Flashback and Twitter, which started to be published at the beginning of the coronavirus pandemic (early 2020).
svt_2016	21,729,542	`SVT News 2016`—News texts from `svt.se` in 2016.
svt_2017	21,184,642	`SVT News 2017`—News texts from `svt.se` in 2017.
svt_2023	7,501,502	`SVT News 2023`—News texts from `svt.se` in 2023.

C

Execution Error and Timeout Metrics for Clusters with Recorded Failures

This appendix provides, for each cluster that experienced execution failures, the total number of executions, error and timeout counts, and their corresponding rates. Table C.1 summarizes these metrics by cluster identifier.

Table C.1: Summary of execution volumes, error counts, and timeout counts for clusters with recorded failures. **C**: Cluster, **TE**: Total number of executions, **EC**: Error count, **ER**: Error rate, **TC**: Timeout count, **TR**: Timeout rate.

C	TE	EC	ER	TC	TR
0	2338	334	0.1429	0	0
1	1736	248	0.1429	0	0
2	2240	324	0.1446	0	0
3	3318	482	0.1453	72	0.0217
4	3360	0	0	39	0.0116
5	5810	28	0.0048	45	0.0077
6	1862	56	0.0301	0	0
8	5922	1708	0.2884	0	0
9	2520	720	0.2857	0	0
17	9394	336	0.0358	0	0
18	3080	880	0.2857	0	0
19	2506	716	0.2857	0	0
20	2870	2	0.0007	3	0.0010
21	4438	56	0.0126	3	0.0007
24	4690	14	0.0030	0	0
40	5320	42	0.0079	0	0
59	4256	14	0.0033	3	0.0007
62	2800	1955	0.6982	10	0.0036
63	3332	28	0.0084	0	0
64	2226	1918	0.8616	0	0
65	3038	14	0.0046	0	0

Continued on next page

Table C.1 – continued from previous page

C	TE	EC	ER	TC	TR
66	2730	2106	0.7714	0	0
81	12054	56	0.0046	0	0
83	2520	2520	1.0	0	0
86	2548	28	0.0110	0	0
87	4760	14	0.0029	0	0
91	2590	14	0.0054	0	0
92	19446	14	0.0007	0	0
102	21868	14	0.0006	0	0
118	5628	14	0.0025	0	0
121	1862	14	0.0075	0	0
122	1778	112	0.0630	5	0.0028
124	3402	14	0.0041	0	0
128	3458	14	0.0040	0	0
131	4382	42	0.0096	0	0
134	34902	14	0.0004	0	0
141	56728	14	0.0002	0	0
145	67872	14	0.0002	0	0
150	25004	14	0.0006	0	0
151	17850	5	0.0003	0	0
153	9660	14	0.0014	0	0
156	3990	28	0.0070	0	0

D

Results of the Benchmark Development Process Before the Refinement Step

This appendix presents the results from the benchmark development process, as outlined in Section 3.6, prior to the manual refinement step. These results evaluate the performance and variability of query executions across different clusters.

Table D.1 provides key performance metrics, including execution success rate, mean execution times, SDs, and CVs at both the cluster and representative query levels. These metrics offer an overview of query stability and reliability across various corpora.

Moreover, Table D.2 lists the representative queries for each cluster, illustrating the query structures analyzed during this stage.

Table D.1: Performance metrics of the benchmark development process prior to manual refinement. **C**: Cluster, **ESR**: Execution success rate, **MC**: Mean execution time of cluster, **SDC**: Standard deviation of cluster, **CVC**: Coefficient of variation of cluster, **MQ**: Mean execution time of representative query, **SDQ**: Standard deviation of representative query, **CVQ**: Coefficient of variation of representative query.

C	ESR	MC	SDC	CVC	MQ	SDQ	CVQ
0	0.8571	2.0119	1.6355	0.8129	2.6646	1.7129	0.6428
1	0.8571	1.2522	1.8766	1.4986	0.3245	0.1814	0.5591
2	0.8554	0.2349	0.9105	3.8765	0.0672	0.0037	0.0554
3	0.8330	1.1099	5.1271	4.6195	0.0682	0.0045	0.0657
4	0.9884	4.2800	7.6393	1.7849	30.1225	13.0432	0.4330
5	0.9874	2.8753	6.5582	2.2809	27.0139	11.6102	0.4298
6	0.9699	0.5163	0.9710	1.8809	0.1450	0.0470	0.3244
7	1.0	3.5874	5.1551	1.4370	0.2922	0.0903	0.3091
8	0.7116	3.1850	1.9140	0.6009	1.4487	0.2692	0.1858
9	0.7143	2.0849	1.2439	0.5966	2.1424	1.2388	0.5783
10	1.0	0.4808	0.3502	0.7284	0.5019	0.3186	0.6348
11	1.0	0.5833	0.4965	0.8512	0.8497	0.4928	0.5800

Continued on next page

Table D.1 – continued from previous page

C	ESR	MC	SDC	CVC	MQ	SDQ	CVQ
12	1.0	0.5077	0.3770	0.7425	0.2800	0.1698	0.6065
13	1.0	0.5072	0.3680	0.7256	0.6331	0.3860	0.6097
14	1.0	0.4690	0.3464	0.7386	0.2646	0.1566	0.5919
15	1.0	0.4478	0.3215	0.7179	0.3987	0.2504	0.6280
16	1.0	0.2381	0.1088	0.4570	0.2107	0.0604	0.2867
17	0.9642	2.1156	3.3222	1.5704	0.0623	0.0066	0.1062
18	0.7143	0.8992	0.4908	0.5458	0.8938	0.4664	0.5218
19	0.7143	1.4110	0.8171	0.5791	1.5022	0.8291	0.5520
20	0.9983	0.6475	1.8057	2.7889	0.1766	0.0644	0.3648
21	0.9867	1.6587	3.3213	2.0024	0.0785	0.0070	0.0887
22	1.0	3.4118	3.6358	1.0656	0.1871	0.0688	0.3675
23	1.0	2.9068	3.4786	1.1967	0.1747	0.0581	0.3326
24	0.9970	0.6247	1.3920	2.2284	0.2600	0.0766	0.2946
25	1.0	1.5341	2.7439	1.7886	0.0716	0.0056	0.0779
26	1.0	0.7121	1.6000	2.2468	0.0739	0.0045	0.0606
27	1.0	1.4793	2.5810	1.7448	0.0843	0.0054	0.0645
28	1.0	1.2130	2.5944	2.1389	0.2248	0.0776	0.3451
29	1.0	0.9670	2.0758	2.1468	0.0901	0.0054	0.0598
30	1.0	1.0414	2.1677	2.0815	0.0836	0.0053	0.0638
31	1.0	0.9703	1.9956	2.0567	0.0724	0.0052	0.0721
32	1.0	1.0182	2.1310	2.0929	0.0740	0.0054	0.0726
33	1.0	1.4480	2.5009	1.7272	0.1011	0.0096	0.0952
34	1.0	1.8686	2.8078	1.5026	0.0761	0.0070	0.0917
35	1.0	1.4152	2.4574	1.7365	0.0904	0.0079	0.0870
36	1.0	1.0720	2.0976	1.9568	0.0834	0.0073	0.0878
37	1.0	1.1768	2.2720	1.9307	0.0852	0.0075	0.0884
38	1.0	1.3682	2.4538	1.7934	0.0882	0.0065	0.0740
39	1.0	1.5540	2.5754	1.6572	0.0770	0.0074	0.0962
40	0.9921	1.8470	3.1214	1.6899	0.0923	0.0073	0.0795
41	1.0	1.4928	2.5785	1.7272	0.1867	0.0603	0.3230
42	1.0	1.0387	2.7187	2.6174	0.0934	0.0065	0.0694
43	1.0	0.3900	1.1887	3.0481	0.0852	0.0050	0.0590
44	1.0	1.6822	2.7020	1.6062	0.0749	0.0057	0.0766
45	1.0	1.1567	2.2376	1.9345	0.0897	0.0055	0.0613
46	1.0	0.5032	1.4065	2.7950	0.0843	0.0055	0.0656
47	1.0	1.5311	2.5653	1.6755	0.0701	0.0045	0.0639
48	1.0	1.0374	2.1147	2.0385	0.0905	0.0071	0.0781
49	1.0	0.9201	2.0149	2.1898	0.0839	0.0049	0.0582
50	1.0	0.8586	1.8921	2.2036	0.0866	0.0054	0.0623
51	1.0	0.7246	1.7119	2.3626	0.0784	0.0053	0.0680
52	1.0	0.6922	1.6707	2.4138	0.0801	0.0053	0.0662
53	1.0	1.2789	2.3763	1.8581	0.0883	0.0045	0.0510

Continued on next page

Table D.1 – continued from previous page

C	ESR	MC	SDC	CVC	MQ	SDQ	CVQ
54	1.0	1.4682	2.4900	1.6960	0.0821	0.0069	0.0845
55	1.0	0.8098	1.7984	2.2206	0.0873	0.0063	0.0726
56	1.0	0.6934	1.6586	2.3920	0.0648	0.0039	0.0608
57	1.0	0.6685	1.5996	2.3930	0.0819	0.0045	0.0547
58	1.0	0.5710	1.5529	2.7198	0.0918	0.0093	0.1016
59	0.9960	0.4425	2.0094	4.5414	0.1525	0.0666	0.4369
60	1.0	1.4753	2.8135	1.9071	0.2653	0.0727	0.2739
61	1.0	2.2733	3.3576	1.4770	0.0655	0.0068	0.1034
62	0.2982	9.1902	10.9381	1.1902	24.1065	3.0723	0.1274
63	0.9916	0.2814	0.2761	0.9813	1.4821	1.0563	0.7127
64	0.1384	2.6349	6.6430	2.5212	0.0687	0.0052	0.0763
65	0.9954	0.1999	1.3818	6.9136	0.0691	0.0033	0.0475
66	0.2286	0.1457	0.1895	1.3004	0.0958	0.0003	0.0029
67	1.0	0.1997	0.0664	0.3326	0.2041	0.0563	0.2757
68	1.0	0.2039	0.0671	0.3293	0.2205	0.0603	0.2733
69	1.0	0.2634	0.1583	0.6012	0.2349	0.0672	0.2862
70	1.0	0.2096	0.0690	0.3291	0.2227	0.0614	0.2759
71	1.0	0.2069	0.0685	0.3312	0.2236	0.0638	0.2851
72	1.0	0.2428	0.0880	0.3626	0.2428	0.0654	0.2692
73	1.0	0.2194	0.0723	0.3297	0.2205	0.0605	0.2745
74	1.0	0.2382	0.0827	0.3474	0.2372	0.0633	0.2667
75	1.0	0.2272	0.0755	0.3323	0.2382	0.0583	0.2447
76	1.0	0.2235	0.0735	0.3288	0.2323	0.0635	0.2736
77	1.0	0.2160	0.0712	0.3297	0.2324	0.0634	0.2726
78	1.0	0.2126	0.0703	0.3309	0.2014	0.0542	0.2689
79	1.0	0.2347	0.0799	0.3405	0.2444	0.0669	0.2737
80	1.0	0.2304	0.0774	0.3358	0.2377	0.0679	0.2857
81	0.9954	1.9179	3.2979	1.7195	0.0685	0.0043	0.0635
82	1.0	0.2269	1.0622	4.6807	0.0653	0.0041	0.0633
83	-	-	-	-	-	-	-
84	1.0	1.2680	3.1092	2.4521	0.0796	0.0098	0.1228
85	1.0	1.0607	3.6196	3.4125	0.0890	0.0051	0.0576
86	0.9890	1.0380	3.7412	3.6043	0.0662	0.0055	0.0833
87	0.9971	0.4292	1.2239	2.8513	0.0892	0.0061	0.0685
88	1.0	0.1591	0.0635	0.3991	0.1588	0.0483	0.3038
89	1.0	0.5553	1.3466	2.4249	0.1772	0.0570	0.3220
90	1.0	0.8582	1.9037	2.2183	0.1640	0.0555	0.3385
91	0.9946	0.5175	1.6932	3.2722	0.1518	0.0530	0.3491
92	0.9993	0.4985	1.4417	2.8921	0.0767	0.0042	0.0550
93	1.0	0.2279	0.7216	3.1666	0.0875	0.0062	0.0707
94	1.0	0.2062	0.6720	3.2594	0.0682	0.0026	0.0385
95	1.0	0.3967	1.2017	3.0295	0.0911	0.0058	0.0634

Continued on next page

Table D.1 – continued from previous page

C	ESR	MC	SDC	CVC	MQ	SDQ	CVQ
96	1.0	0.3838	1.2147	3.1650	0.0798	0.0053	0.0668
97	1.0	0.3346	1.1051	3.3030	0.0732	0.0041	0.0558
98	1.0	0.2880	0.9515	3.3034	0.0911	0.0062	0.0683
99	1.0	0.1775	0.6431	3.6227	0.0652	0.0036	0.0549
100	1.0	0.1462	0.4996	3.4176	0.0803	0.0056	0.0694
101	1.0	1.1113	2.2595	2.0332	0.0785	0.0057	0.0731
102	0.9994	0.5948	1.5669	2.6344	0.0878	0.0053	0.0608
103	1.0	0.9770	2.1153	2.1652	0.0847	0.0040	0.0472
104	1.0	0.8224	1.9281	2.3444	0.0874	0.0037	0.0426
105	1.0	0.6578	1.6762	2.5480	0.0813	0.0046	0.0560
106	1.0	0.1624	0.0654	0.4026	0.1649	0.0555	0.3366
107	1.0	0.1638	0.0663	0.4046	0.1748	0.0546	0.3123
108	1.0	0.1605	0.0640	0.3987	0.1789	0.0551	0.3079
109	1.0	0.1596	0.0630	0.3944	0.1656	0.0490	0.2956
110	1.0	0.2436	0.6708	2.7537	0.1545	0.0447	0.2895
111	1.0	1.2697	2.3778	1.8727	0.0717	0.0060	0.0844
112	1.0	1.1754	2.3460	1.9959	0.0772	0.0052	0.0676
113	1.0	0.8693	1.9659	2.2616	0.1799	0.0604	0.3360
114	1.0	1.0566	2.1374	2.0228	0.1877	0.0674	0.3589
115	1.0	0.1271	0.4133	3.2513	0.0697	0.0044	0.0625
116	1.0	0.1428	0.5340	3.7406	0.0863	0.0067	0.0780
117	1.0	0.1644	0.0678	0.4122	0.1572	0.0542	0.3447
118	0.9975	0.1638	0.0679	0.4144	0.1666	0.0569	0.3413
119	1.0	0.1638	0.0677	0.4136	0.1572	0.0534	0.3398
120	1.0	0.1568	0.0602	0.3837	0.1643	0.0522	0.3178
121	0.9925	0.1571	0.0596	0.3791	0.1595	0.0468	0.2936
122	0.9342	0.5155	2.9694	5.7608	0.0688	0.0088	0.1281
123	1.0	0.1911	0.1024	0.5357	0.1671	0.0553	0.3309
124	0.9959	0.1754	0.0846	0.4821	0.1824	0.0606	0.3322
125	1.0	0.1847	0.0932	0.5046	0.1718	0.0559	0.3256
126	1.0	0.1715	0.0766	0.4469	0.1799	0.0582	0.3236
127	1.0	0.1698	0.0732	0.4313	0.1701	0.0520	0.3056
128	0.9960	0.1969	0.1260	0.6399	0.1706	0.0532	0.3118
129	1.0	0.2126	0.1444	0.6792	0.1945	0.0672	0.3454
130	1.0	0.2904	0.3523	1.2134	0.1796	0.0624	0.3477
131	0.9904	0.0913	0.0950	1.0411	0.0654	0.0043	0.0657
132	1.0	0.0817	0.0362	0.4425	0.0672	0.0030	0.0453
133	1.0	0.0836	0.0258	0.3081	0.0742	0.0031	0.0418
134	0.9996	0.0806	0.0174	0.2161	0.0862	0.0039	0.0447
135	1.0	0.0806	0.0189	0.2338	0.0679	0.0031	0.0460
136	1.0	0.0806	0.0302	0.3745	0.0683	0.0032	0.0475
137	1.0	0.0804	0.0166	0.2060	0.0722	0.0027	0.0369

Continued on next page

Table D.1 – continued from previous page

C	ESR	MC	SDC	CVC	MQ	SDQ	CVQ
138	1.0	0.0801	0.0166	0.2076	0.0680	0.0031	0.0458
139	1.0	0.1681	0.0685	0.4075	0.1636	0.0490	0.2998
140	1.0	0.1674	0.0673	0.4018	0.1760	0.0558	0.3170
141	0.9998	0.0805	0.0162	0.2007	0.0812	0.0041	0.0505
142	1.0	0.1658	0.0655	0.3948	0.1867	0.0602	0.3226
143	1.0	0.1637	0.0637	0.3892	0.1417	0.0428	0.3022
144	1.0	0.0806	0.0172	0.2138	0.0704	0.0026	0.0376
145	0.9998	0.0807	0.0268	0.3318	0.0667	0.0028	0.0424
146	1.0	0.1713	0.0710	0.4146	0.1644	0.0513	0.3118
147	1.0	0.1734	0.0777	0.4480	0.1659	0.0577	0.3478
148	1.0	0.1641	0.0696	0.4240	0.1589	0.0505	0.3177
149	1.0	0.1632	0.0673	0.4124	0.1498	0.0482	0.3220
150	0.9994	0.0814	0.0221	0.2716	0.0925	0.0046	0.0497
151	0.9997	0.0812	0.0369	0.4545	0.0665	0.0033	0.0495
152	1.0	0.0815	0.0169	0.2079	0.0644	0.0026	0.0408
153	0.9986	0.0816	0.0168	0.2054	0.0683	0.0033	0.0489
154	1.0	0.0816	0.0169	0.2074	0.0641	0.0034	0.0536
155	1.0	0.0828	0.0198	0.2391	0.0742	0.0045	0.0604
156	0.9930	0.0822	0.0182	0.2216	0.0655	0.0038	0.0575
157	1.0	0.0821	0.0198	0.2412	0.0685	0.0033	0.0476
158	1.0	0.1624	0.0623	0.3836	0.1688	0.0515	0.3050
159	1.0	0.1597	0.0603	0.3774	0.1619	0.0443	0.2738

Table D.2: Representative queries from the benchmark development process prior to manual refinement. **C**: Cluster.

C	Representative Query
0	`[(word = "i" %c | word = "j" %c | word = "ii" %c | word = "ij" %c | word = "ih" %c | word = "jh" %c) & ((int(_.text_datefrom) = 19600101 & int(_.text_timefrom) >= 000000) | (int(_.text_datefrom) > 19600101 & int(_.text_datefrom) <= 19791231)) & (int(_.text_dateto) < 19791231 | (int(_.text_dateto) = 19791231 & int(_.text_timeto) <= 235959))] [word != "i" %c]{0,3} [lex contains "anslutning\.\.nn\.1"] [(word = ".*till" | word = ".*til")]`
1	`[(word = "inom" %c | word = "jnom" %c) & ((int(_.text_datefrom) = 19400101 & int(_.text_timefrom) >= 000000) | (int(_.text_datefrom) > 19400101 & int(_.text_datefrom) <= 19591231)) & (int(_.text_dateto) < 19591231 | (int(_.text_dateto) = 19591231 & int(_.text_timeto) <= 235959))] [word != "inom" %c]{0,3} [lex contains "ram\.\.nn\.1"] [(word = ".*för" | word = ".*fär")]`

Continued on next page

Table D.2 – continued from previous page

C	Representative Query
2	[(word = "bedde" | word = "bedje") & ((int(_.text_datefrom) = 19500101 & int(_.text_timefrom) >= 000000) | (int(_.text_datefrom) > 19500101 & int(_.text_datefrom) <= 19601231)) & (int(_.text_dateto) < 19601231 | (int(_.text_dateto) = 19601231 & int(_.text_timeto) <= 235959))]
3	[word = "ha rent mjöl i påsen"] [((int(_.text_datefrom) = 18000101 & int(_.text_timefrom) >= 094500) | (int(_.text_datefrom) > 18000101 & int(_.text_datefrom) <= 18501231)) & (int(_.text_dateto) < 18501231 | (int(_.text_dateto) = 18501231 & int(_.text_timeto) <= 235959))]
4	[(word = "i.*" %c | word = ".*i.*" %c | word = ".*i" %c)] [(word = "svenska.*" %c | word = ".*svenska.*" %c | word = ".*svenska" %c)] [(word = "skolan.*" %c | word = ".*skolan.*" %c | word = ".*skolan" %c)]
5	[(word = "i.*" %c | word = ".*i.*" %c | word = ".*i" %c)] [(word = "sammanfattning.*" %c | word = ".*sammanfattning.*" %c | word = ".*sammanfattning" %c)]
6	[(msd = ".*JJ\.POS\.MAS\.SIN\.DEF\.NOM.*" | msd = ".*JJ\.POS\.MAS\.SIN\.DEF\.GEN.*" | msd = ".*JJ\.SUV\.MAS\.SIN\.DEF\.GEN.*" | msd = ".*JJ\.SUV\.MAS\.SIN\.DEF\.GEN.*")] [pos = "PM"]
7	[lex contains "skilja\.\.vb\.1" & pos = "VB"] [pos = "JJ" & pos = "AB"]{0,3} [(msd = ".*NN\.UTR\.SIN\.IND\.NOM.*" | msd = ".*NN\.NEU\.SIN\.IND\.NOM.*") & pos = "NN" & deprel = "OO"]
8	[(lex contains "forskning\.\.nn\.1" | complemgram contains "forskning\.\.nn\.1\+.*" | complemgram contains ".*\+forskning\.\.nn\.1\+.*" | complemgram contains ".*\+forskning\.\.nn\.1:.*") & _.text_year = "2022"]
9	[(lex contains "gå_åt\.\.vbm\.1" | complemgram contains "gå_åt\.\.vbm\.1\+.*" | complemgram contains ".*\+gå_åt\.\.vbm\.1\+.*")]
10	[(word = "flykting.*" %c | word = ".*flykting.*" %c | word = ".*flykting" %c)]
11	[(word = "sjuk.*" %c | word = ".*sjuk.*" %c | word = ".*sjuk" %c)]
12	[(word = "varsa,.*" %c | word = ".*varsa,.*" %c | word = ".*varsa," %c)]
13	[(word = "marknad.*" %c | word = ".*marknad.*" %c | word = ".*marknad" %c)]
14	[(word = "tilqwämd,.*" %c | word = ".*tilqwämd,.*" %c | word = ".*tilqwämd," %c)]
15	[(word = "engaledhis.*" %c | word = ".*engaledhis.*" %c | word = ".*engaledhis" %c)]

Continued on next page

Table D.2 – continued from previous page

C	Representative Query
16	[lex contains "fsvm--sar..pn.1"]
17	[pos = "okänd" & lex contains "vars..pn.1"]
18	[(lex contains "samband\.\.nn\.1" | complemgram contains "samband\.\.nn\.1\+.*")]
19	[(lex contains "katt\.\.nn\.1" | complemgram contains ".*\+katt\.\.nn\.1:.*")]
20	[word = "småhalv.*" & (word = ".*ande" | word = ".*ende")]
21	[word = "spänner"] [] [lemma contains ".*?av.*"]
22	[word = "enhet" %c] [word = "inom" %c] [word = "den" %c] [word = "sociala" %c] [word = "ekoknomin" %c]
23	[word = "riksdagsman" %c] [word = "AND" %c] [word = """kerstin " %c] [word = "hesselgren""" %c]
24	[lex contains "såväl_som\.\.knm\.1"] [lex contains "lycka\.\.nn\.2"]
25	[word = "Her"] [word = "sigx"] [word = "aff"] [word = "flores"] [word = "oc"] [word = "blanzafloor"]
26	[word = "determinerare"] [pos = "JJ"] [lex contains "finne\.\.nn\.1"]
27	[word = "mors"] [word = "moder"] [word = "och"] [word = "mors"] [word = "fader"]
28	[word = "president.*"] [word = ".*"] [word = "motorcykelklubb.*"]
29	[word = "stadi"] [word = "man"] [word = "find"] [word = "hitter"]
30	[word = "ankom"] [word = "den"] [word = "glada"] [word = "rapporten"]
31	[word = "Nyckeln"] [word = "sitter"] [word = "i"] [word = "löset"]
32	[word = "1857"] [word = "den"] [word = "nya"] [word = "diligencen"]
33	[word = "haka"] [word = "upp"] [word = "sig"] [word = "på"]
34	[word = "Arg"] [word = "som"] [word = "ett"] [word = "bi"]
35	[word = "drunknar"] [word = "i"] [word = "ett"] [word = "hav"]
36	[word = "Älska"] [word = "någon"] [word = "kan"] [word = "man"]
37	[word = "grälade"] [word = "på"] [word = "sig"] [word = "själv"]
38	[word = "prövar"] [word = "på"] [word = "sin"] [word = "tur"]
39	[word = "gotta"] [word = "sig"] [word = "åt"] [word = "att"]
40	[word = "kollektivet"] [word = "heter"] [pos = "JJ"]
41	[word = "hustru" %c] [word = "och" %c] [word = "man" %c]
42	[word = "nyheten"] [] [word = "på"] [word = "förstasidan"]

Continued on next page

Table D.2 – continued from previous page

C	Representative Query
43	[word = "vänskapskampen"] [word = "i"] [word = "intensitet"]
44	[word = "aad"] [word = "u"] [word = "yar"]
45	[word = "Sitt"] [word = "eget"] [word = "liv"]
46	[word = "servitris"] [word = "och"] [word = "servitör"]
47	[word = "sign"] [word = "me"] [word = "up"]
48	[word = "Sorget"] [word = "kan"] [word = "man"]
49	[word = "jesus"] [word = "resa"] [word = "själ"]
50	[word = "aa"] [word = "fem"] [word = "dagarna"]
51	[word = "verb"] [word = "sig"] [word = "adjektiv"]
52	[word = "utandning"] [word = "genom"] [word = "näsan"]
53	[word = "sänt"] [word = "en"] [word = "palm"]
54	[word = "Blek"] [word = "som"] [word = "en"]
55	[word = "bortom"] [word = "denna"] [word = "punkt"]
56	[word = "mællum"] [word = "rættara"] [word = "wara"]
57	[word = "knak"] [word = "eller"] [word = "knäppning"]
58	[word = "färgad"] [lex contains ""]
59	[(word = ".*mplatå.*" | word = "mplatå")]
60	[lex contains "allvarlig\.\.av\.1"] [pos = "NN"]
61	[pos = "nn" & lemma contains "avdelning"]
62	[_.forum_url = ""]
63	[lemma contains "sig"]
64	"date=2012"
65	[(word = "dröjer" | word = "fördröjer")]
66	[word = "vaccin" & _.forum_title = ""]
67	[lex contains "till_skillnad_från\.\.ppm\.1"]
68	[lex contains "till_förmån_för\.\.ppm\.1"]
69	[lex contains "fil\.\.nn\.2"]
70	[lex contains "på_fallrepet\.\.avm\.1"]
71	[lex contains "vad_beträffar\.\.ppm\.1"]
72	[lex contains "söka\.\.vb\.1"]
73	[lex contains "tyna_bort\.\.vbm\.1"]
74	[lex contains "kämpa\.\.vb\.1"]
75	[lex contains "omnibuss\.\.nn\.1"]
76	[lex contains "förbannad\.\.av\.1"]
77	[lex contains "lantbrukare\.\.nn\.1"]
78	[lex contains "dalinm--stad\.\.nn\.1"]
79	[lex contains "smärta\.\.nn\.1"]
80	[lex contains "vitrysk\.\.av\.1"]
81	[word= "hund skall"][pos= "NN"]
82	[word = "middagsstängt" & pos = "AB"]
83	-
84	[word = "ka"] [word = ".*sii.*"]

Continued on next page

Table D.2 – continued from previous page

C	Representative Query
85	[word = "roten"] [] [word = "ekvationen"]
86	[word = "maan"] [word = ".*o"]
87	[word = "Femton" & word = "Kvart"]
88	[word = "kandidattj&#65533;?¤nstg&#65533;?¶ring″]
89	[word = "refereras" %c] [word = "till" %c]
90	[word = "omvandla" %c] [word = "sig" %c]
91	[word = "'\w+ [ae] + ndes'"]
92	[word = "tappra"] [word = "tårar"]
93	[word = "resultaten"] [word = "vilkea"]
94	[word = "skändligt"] [word = "förtryck"]
95	[word = "bemötande"] [word = "mot"]
96	[word = "hängig"] [word = "förkyld"]
97	[word = "inbiten"] [word = "realist"]
98	[word = "suckade"] [word = "ljudligt"]
99	[word = "Australian"] [word = "Football"]
100	[word = "kontrollerade"] [word = "svaren"]
101	[word = "Plan"] [word = "på"]
102	[word = "jesus"] [word = "jesus"]
103	[word = "vinka"] [word = "åt"]
104	[word = "förmer"] [word = "än"]
105	[word = "ilska"] [word = "över"]
106	[word = "sk&#65533;?¤rning″]
107	[word = "seri&#65533;?¶sa″]
108	[word = "mysteri&#65533;?¶s″]
109	[word = "n&#65533;?¤ppeligen″]
110	[word = "informationsrymd.*" %c]
111	[word = "oj,"] [word = "oj"]
112	[word = "hå"] [word = "hå"]
113	[word = "löstes" %c] [word = "upp" %c]
114	[word = "värna" %c] [word = "sig" %c]
115	[word = "arbetssätten"] [word = "besvarar"]
116	[word = "stärkta"] [word = "synergieffekter"]
117	[word = "fortg&#65533;?¥″]
118	[word = "vr&#65533;?¥ng″]
119	[word = "arm&#65533;?©″]
120	[word = "beh&#65533;?¤rskning″]
121	[word = "omst&#65533;?¤ndighet″]
122	[word = "\w" %l]
123	[word = "pondus.*"]
124	[word = "sakunnig.*"]
125	[word = "Smittar.*"]
126	[word = "brottning.*"]

Continued on next page

Table D.2 – continued from previous page

C	Representative Query
127	[word = "copywriter.*"]
128	[word = "bysbo.*"]
129	[word = "Fast.*"]
130	[word = "NN-.*"]
131	[word = "Öl"]
132	[word = "dogo"]
133	[word = "Oro"]
134	[word = "vänsterbenet"]
135	[word = "svansföring"]
136	[word = "vÃlskrapad"]
137	[word = "högdragar"]
138	[word = "dagtinga"]
139	[word = "minnes" %c]
140	[word = "imkanal" %c]
141	[word = "Pryda"]
142	[word = "täckande" %c]
143	[word = "magaalada" %c]
144	[word = "fuffens"]
145	[word = "helsom"]
146	[word = "tahay" %c]
147	[word = "taga" %c]
148	[word = "nyttotrafik.*"]
149	[word = "preceptorlön.*"]
150	[word = "växthuseffekt"]
151	[word = "språkanvändare"]
152	[word = "marknadsskojare"]
153	[word = "oljekontaminerat"]
154	[word = "ägarförhållandena"]
155	[word = "modermålundervisning"]
156	[word = "nationalistledaren"]
157	[word = "Icke-diskriminering"]
158	[word = "fångenskap" %c]
159	[word = "missaktning" %c]

E

Refined Benchmark

This appendix introduces the refined benchmark for evaluating the performance of corpora search system, following the process outlined in Section 3.6.2. The benchmark consolidates similar clusters and provides a more streamlined and representative evaluation tool.

Table E.1 presents the refined benchmark, listing the mean execution times for each representative query and showing the identification numbers corresponding to both grouped and individual clusters from Table 4.4.

Table E.1: Refined benchmark for evaluating corpora search systems. **ID**: Identification number corresponding to both grouped and individual clusters from Table 4.4. **MQ**: Mean execution time for the representative query.

ID	MQ	Representative Query
0	0.0812	`[word = "Pryda"]`
1	0.1595	`[word = "omst&#65533;?¤ndighet″]`
2	0.1619	`[word = "missaktning" %c]`
3	0.0717	`[word = "oj,"] [word = "oj"]`
4	0.0796	`[word = "ka"] [word = ".*sii.*"]`
5	0.1877	`[word = "värna" %c] [word = "sig" %c]`
6	0.0749	`[word = "aad"] [word = "u"] [word = "yar"]`
7	0.0761	`[word = "Arg"] [word = "som"] [word = "ett"] [word = "bi"]`
8	0.0892	`[word = "Femton" & word = "Kvart"]`
9	0.3987	`[(word = "engaledhis.*" %c | word = ".*engaledhis.*" %c | word = ".*engaledhis" %c)]`
10	0.2323	`[lex contains "förbannad\.\.av\.1"]`
11	0.8938	`[(lex contains "samband\.\.nn\.1" | complemgram contains "samband\.\.nn\.1\+.*")]`

Continued on next page

Table E.1 – continued from previous page

ID	MQ	Representative Query
12	2.6646	[(word = "i" %c | word = "j" %c | word = "ii" %c | word = "ij" %c | word = "ih" %c | word = "jh" %c) & ((int(_.text_datefrom) = 19600101 & int(_.text_timefrom) >= 000000) | (int(_.text_datefrom) > 19600101 & int(_.text_datefrom) <= 19791231)) & (int(_.text_dateto) < 19791231 | (int(_.text_dateto) = 19791231 & int(_.text_timeto) <= 235959))] [word != "i" %c]{0,3} [lex contains "anslutning\.\.nn\.1"] [(word = ".*till" | word = ".*til")]
13	0.3245	[(word = "inom" %c | word = "jnom" %c) & ((int(_.text_datefrom) = 19400101 & int(_.text_timefrom) >= 000000) | (int(_.text_datefrom) > 19400101 & int(_.text_datefrom) <= 19591231)) & (int(_.text_dateto) < 19591231 | (int(_.text_dateto) = 19591231 & int(_.text_timeto) <= 235959))] [word != "inom" %c]{0,3} [lex contains "ram\.\.nn\.1"] [(word = ".*för" | word = ".*för")]
14	0.0672	[(word = "bedde" | word = "bedje") & ((int(_.text_datefrom) = 19500101 & int(_.text_timefrom) >= 000000) | (int(_.text_datefrom) > 19500101 & int(_.text_datefrom) <= 19601231)) & (int(_.text_dateto) < 19601231 | (int(_.text_dateto) = 19601231 & int(_.text_timeto) <= 235959))]
15	0.0682	[word = "ha rent mjöl i påsen"] [((int(_.text_datefrom) = 18000101 & int(_.text_timefrom) >= 094500) | (int(_.text_datefrom) > 18000101 & int(_.text_datefrom) <= 18501231)) & (int(_.text_dateto) < 18501231 | (int(_.text_dateto) = 18501231 & int(_.text_timeto) <= 235959))]
16	30.1225	[(word = "i.*" %c | word = ".*i.*" %c | word = ".*i" %c)] [(word = "svenska.*" %c | word = ".*svenska.*" %c | word = ".*svenska" %c)] [(word = "skolan.*" %c | word = ".*skolan.*" %c | word = ".*skolan" %c)]
17	27.0139	[(word = "i.*" %c | word = ".*i.*" %c | word = ".*i" %c)] [(word = "sammanfattning.*" %c | word = ".*sammanfattning.*" %c | word = ".*sammanfattning" %c)]
18	0.1450	[(msd = ".*JJ\.POS\.MAS\.SIN\.DEF\.NOM.*" | msd = ".*JJ\.POS\.MAS\.SIN\.DEF\.GEN.*" | msd = ".*JJ\.SUV\.MAS\.SIN\.DEF\.GEN.*" | msd = ".*JJ\.SUV\.MAS\.SIN\.DEF\.GEN.*")] [pos = "PM"]
19	0.2922	[lex contains "skilja\.\.vb\.1" & pos = "VB"] [pos = "JJ" & pos = "AB"]{0,3} [(msd = ".*NN\.UTR\.SIN\.IND\.NOM.*" | msd = ".*NN\.NEU\.SIN\.IND\.NOM.*") & pos = "NN" & deprel = "OO"]

Continued on next page

Table E.1 – continued from previous page

ID	MQ	Representative Query
20	1.4487	[(lex contains "forskning\.\.nn\.1" | complemgram contains "forskning\.\.nn\.1\+.*" | complemgram contains ".*\+forskning\.\.nn\.1\+.*" | complemgram contains ".*\+forskning\.\.nn\.1:.*") & _.text_year = "2022"]
21	2.1424	[(lex contains "gå_åt\.\.vbm\.1" | complemgram contains "gå_åt\.\.vbm\.1\+.*" | complemgram contains ".*\+gå_åt\.\.vbm\.1\+.*")]
22	0.0623	[pos = "okänd" & lex contains "vars..pn.1"]
23	0.1766	[word = "småhalv.*" & (word = ".*ande" | word = ".*ende")]
24	0.0785	[word = "spänner"] [] [lemma contains ".*?av.*"]
25	0.1871	[word = "enhet" %c] [word = "inom" %c] [word = "den" %c] [word = "sociala" %c] [word = "ekoknomin" %c]
26	0.1747	[word = "riksdagsman" %c] [word = "AND" %c] [word = "" "kerstin" %c] [word = "hesselgren""" %c]
27	0.2600	[lex contains "såväl_som\.\.knm\.1"] [lex contains "lycka\.\.nn\.2"]
28	0.0716	[word = "Her"] [word = "sigx"] [word = "aff"] [word = "flores"] [word = "oc"] [word = "blanzafloor"]
29	0.0739	[word = "determinerare"] [pos = "JJ"] [lex contains "finne\.\.nn\.1"]
30	0.0843	[word = "mors"] [word = "moder"] [word = "och"] [word = "mors"] [word = "fader"]
31	0.2248	[word = "president.*"] [word = ".*"] [word = "motorcykelklubb.*"]
32	0.0923	[word = "kollektivet"] [word = "heter"] [pos = "JJ"]
33	0.1867	[word = "hustru" %c] [word = "och" %c] [word = "man" %c]
34	0.0934	[word = "nyheten"] [] [word = "på"] [word = "förstasidan"]
35	0.0918	[word = "färgad"] [lex contains ""]
36	0.1525	[(word = ".*mplatå.*" | word = "mplatå")]
37	0.2653	[lex contains "allvarlig\.\.av\.1"] [pos = "NN"]
38	0.0655	[pos = "nn" & lemma contains "avdelning"]
39	1.4821	[lemma contains "sig"]
40	0.0685	[word= "hund skall"][pos= "NN"]
41	0.0653	[word = "middagsstängt" & pos = "AB"]
42	0.0890	[word = "roten"] [] [word = "ekvationen"]
43	0.1545	[word = "informationsrymd.*" %c]