Bachelor Thesis

Deniz Hagelstein

Quantitative Evaluation of AI-based Literature Search in Comparison to Systematic Literature Search

June 27, 2025

supervised by:
Professor Sibylle Schupp
Antje Rogalla

Hamburg University of Technology (TUHH)
Technische Universität Hamburg
Institute for Software Systems
21073 Hamburg

Statutory Declaration[1]

I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance.

Date	Signature

[1] *Goethe University Frankfurt, Department of Economics*, 2025, `https://www.accounting.uni-frankfurt.de/fileadmin/user_upload/dateien_abteilungen/abt_rec/Statutory_Declaration.pdf`

Abstract

Efficient literature search is a key component for academic research in generating meaningful insights. The yearly growth of scientific publications increasingly complicates this process. Systematic literature search approaches provide a structured methodology but are often time consuming and require expertise in the topic. Although AI tools could potentially provide benefits in speed and coverage to systematic approaches, their effectiveness is not well researched. This thesis aims to compare a systematic literature search with an AI-assisted search using Research Rabbit, both applied to a shared research topic. To compare the findings of the two approaches, three experiments have been conducted and evaluated to find the differences. The results of the experiment show that an AI-assisted approach leads to different results than a systematic approach, with minimal overlap in sources or quantitative metrics.

Contents

List of Figures

List of Algorithms

1 Introduction

With the number of scientific publications increasing each year and no database covering them all [3], it becomes increasingly challenging to assess all the information. This trend is particularly strong in the field of computer science where publication output is growing rapidly [7]. A common way of discovering literature is by conducting a systematic literature review (SLR). However, while SLR approaches provide a structured methodology, they are labor-intensive [13]. These challenges increase the need for solutions to discover academic literature. A potential solution for this can be the usage of AI tools that are specialized in literature search [13]. In this thesis Research Rabbit is used for AI driven literature search.
Research Rabbit uses citation based networks and other proprietary methods for finding literature and creating citation network graphs [11].
The goal of this work is to analyze the differences between AI-based literature searches with a systematic approach. To achieve this, both approaches are applied to a shared research topic using an initial paper to search literature.
This paper uses the SLR approach of Bonfanti et al. [2] and adapts it to focus on the literature searching aspect. However, instead of coming up with keywords, we use KeyBERT [8] to generate keywords out of an initial paper, which is manually selected before starting the queries. The query results are combined into a single file and preprocessed for further analysis.
To compare the two approaches, we have conducted three experiments. Experiment 1 and 2 focus on the comparison of an approach using Research Rabbit and a systematic literature search (SLS) approach. While experiment 1 analyzes the overlapping data results, experiment 2 analyzes quantitative metrics. The last experiment aims to highlight the differences in results when selecting different input data for Research Rabbit. Research Rabbit's limitations, which are discussed in later chapters, get further highlighted by the experiments.
The Experiments and observed limitations of Research Rabbit, indicate that the AI-assisted approach is not yet a replacement for SLS processes.
This thesis is structured as follows. Chapter 2 provides a background and talks about the limitations of AI and systematic literature search, the literature sources and the BibTeX format which is used for storing bibliographic data. Chapter 3 describes in detail how the AI and systematic literature search are implemented. Additionally, chapter 3 presents the implementation of quantitative evaluation metrics. This chapter is followed by experiments in Chapter 4 which are used for evaluating and comparing the approaches. Afterwards, we will discuss the results by providing a summary of the findings and suggestions for future research options.

2 Background

This chapter provides an overview of the key concepts and tools that are necessary for this thesis. We introduce the process of an SLS, which ensures a structured and reproducible process. Moreover, we cover the databases that are used and the file format for saving and handling bibliographic data. Additionally, we will describe AI-based literature search tools and elaborate on the tool used in this thesis.

2.1 Systematic Literature Search

A systematic literature search is a structured method of searching literature to identify and evaluate existing publications on a specific topic. It is used to give an overview of a topic that one would like to research by querying databases and performing quantitative analysis on the data.
In this thesis, we adapt the approach of an SLR by Bonfanti et al. [2] and focus on the search aspect.
A SLR process involves defining clear research questions by querying multiple databases using keywords and boolean operators. SLR help to minimize bias and ensure reproducibility of findings and support evidence-based conclusions by formulating specific research questions [2]. The SLR by Bonfanti et al. is briefly summarized as follows. The process is divided into two main steps:

1. Paper collection

2. Classification and analysis

The first step is the paper collection, where Bonfanti et al. use keywords to perform search queries in databases. Multiple queries are performed with different keywords to obtain results which are expected but were not found. The results are then saved as .bib files and merged for the next step, which is the classification and analysis. In the classification and analysis step, Bonfanti et al. first pre-processed the merged data to filter out duplicates and normalize entries [2]. The process continues by manually deleting non-relevant data such as non-peer reviewed or topic unrelated publications. The SLR is then completed by answering preformed research questions by analyzing the data with quantitative and qualitative metrics.
In this thesis however, we are not interested in the review of a specific topic. Our goal is to compare an AI-based literature search tool with a systematic search approach. For this reason, we do not formulate and answer specific research questions, which are necessary for an SLR. Instead, we perform two search approaches and compare the outcomes. Moreover, we use quantitative metrics and perform three experiments for the comparison. These metrics will be covered in detail in the next chapter.
This work uses three databases for the paper queries: **Scopus**, **DBLP** and **OpenAlex**. All selected databases include conference papers (InProceedings), Journals,

Books. Some databases also include other types of publications, which we are not interested in such as editorial, blog posts, or other non academic content. The reason for including these three publication types is that we want to focus on published and peer reviewed works. We are only interested in peer reviewed publications, as peer review is an indicator for the quality of a publication [4] [2]. However, not all databases contain only peer reviewed publications, therefore extra filtering has to be implemented for these databases.
The databases and their limitations are presented in the following subsections.

2.1.1 Scopus

The first database used for the SLS is Scopus. Scopus is a well established commercial database owned by Elsevier that contains literature from different fields including science, technology, medicine and many other fields.
Scopus is one of the largest scientific publication databases with over 60 million records and over 21500 peer reviewed journals [2]. This makes Scopus essential for a broad data search query to gather literature. Scopus requires a paid subscription; however, it does provide free services via institutional access. However the number of API requests is limited to 20000 calls per week and 5000 total items per query.
The Scopus search allows us to search keywords in the abstract, title and author keywords of publications. The keywords can be separated by boolean operators "AND" or "OR" to combine keywords. Moreover, the Scopus search allows the usage of specific filters or sorting options. All the entries of Scopus come from peer reviewed journals, thus no additional filtering for peer review is necessary.

2.1.2 DBLP

DBLP (Digital Bibliography & Library Project) is an open, specialized database in the field of computer science created by the University of Trier. DBLP offers detailed bibliographic information on major journals, conferences, and workshops. The database focuses on computer science related literature and thus ensures thorough coverage of research in that domain. Unlike Scopus, DBLP does not contain peer reviewed publications only. However, filtering out non peer reviewed papers is possible by using the publication type after the query. DBLP has a restrictive searching API, it is only possible to search for words in the title of publications. The maximum number of results per query is 1000, there is also a limit on the number of API calls. The number of maximum API calls is not disclosed, if a certain threshold is exceeded, the API will send an error message. DBLP allows searching for keywords only in the title of the publications. The DBLP search allows the boolean operators "AND" or "OR" to combine keywords. However, since we only search in the title the query will be structured differently. The building of the search query will be presented in detail in the next chapter.
One other limitation of DBLP is that the data received by the database does not include citation counts.

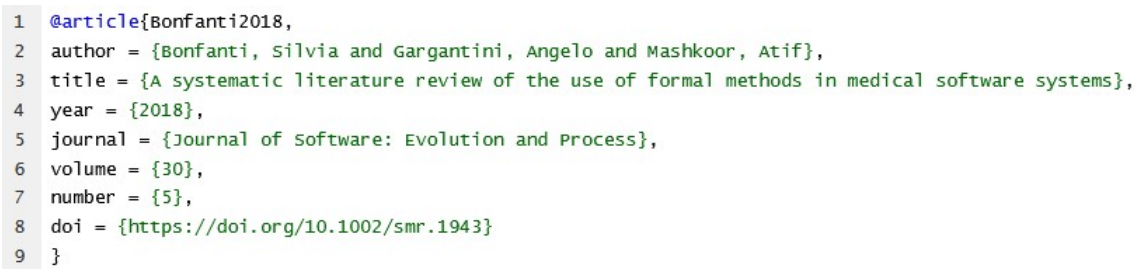
2.1.3 OpenAlex

OpenAlex is an open, comprehensive database that aggregates metadata from a wide range of scholarly resources created by OurResearch. It offers free and structured access to the bibliographic data with a free API connection. The number of API requests is limited to 100,000 calls per day, The OpenAlex API lets us set multiple filters like primary locations, open access and many others, while also allowing us to search for multiple keywords in conjunction. Keywords can be searched in the title and abstract of the database entries separated by the boolean operators "AND" or "OR". OpenAlex allows to search for single works by using the DOI (Digital Object identifier) or name of a publication which is later used for enriching data from other sources.
Similar to DBLP, OpenAlex does not only provide peer reviewed publications. Since we are only interested in peer reviewed publications, we have to set filters, however OpenAlex does not directly provide a filter for peer reviewed works.
We use the filter: "has_oa_accepted_or_published_version:true" to find peer reviewed works. This filter ensures that, for works that undergo peer review, an open access copy exists somewhere[10]. By applying that filter, we get peer reviewed journals and InProceedings articles, but that does not hold for books or other types [10]. Hence we also have to restrict the query by adding another filter so that we receive only journal and InProceeding articles.

2.2 BibTeX

To save the results from a systematic literature search, a file format needs to be chosen. In this thesis BibTeX files have been chosen with the file extension .bib. BibTeX is a reference management tool that store, and format bibliographic references in LaTeX documents. It uses a structured file format to store citation information such as authors, titles, journals, number of citations and publication years. The information that is saved will be explained in detail in the approach section to showcase which data is used and saved. The use of .bib files in academic writing provides several advantages. It ensures consistency in citations and reference lists by automatically formatting them according to specific bibliographic styles. It also simplifies the process of updating references: once a .bib file is modified, the changes are reflected throughout the document, thus improving the maintainability. Additionally, BibTeX files promote collaboration and reproducibility, as researchers can share .bib files alongside their LaTeX documents to ensure transparency in citation practices. These benefits make BibTeX a standard tool in academic writing, particularly in fields such as computer science, engineering, and mathematics, where LaTeX is widely used.

The structure of a .bib file entry can be seen in Fig. 2.1. A BibTeX entry starts with an "@" followed by the entry type such as @article, @inproceedings,@book. After the entry type there will be the citation key which is a unique identifier used to reference that entry. After entry type and key, the bibliographic data is stored by a list of



```
@article{Bonfanti2018,
author = {Bonfanti, Silvia and Gargantini, Angelo and Mashkoor, Atif},
title = {A systematic literature review of the use of formal methods in medical software systems},
year = {2018},
journal = {Journal of Software: Evolution and Process},
volume = {30},
number = {5},
doi = {https://doi.org/10.1002/smr.1943}
}
```

Figure 2.1: .bib File Entry

predefined entry fields such as "year", "title","author", and more. Every entry type has a set of mandatory entry fields; in this thesis, we saved all data as entry type @article. This was done to have the mandatory fields: author, title, journal and year and thus ensure uniform formatting. Other entry fields that have been used for data saving in this thesis are described in chapter 3.1.2.

2.3 AI Literature Search: Research Rabbit

This section explains the AI literature search tool Research Rabbit. We explain the choice of Research Rabbit and its limitations.
Research Rabbit is an AI-powered innovative research discovery tool designed to help explore and build networks of related publications [11]. Research Rabbit as an AI tool that takes an initial paper as input and analyzes it to find similar work [11]. The similar work is then visualized in a citation graph, which maps the citations and co-citations. It is also possible for collaborators to work on collections together and share graphs and similar work [11]
. We can export a .bib file with the similar work found by Research Rabbit. The .bib files contain entries with the following entry fields: **Title** , **DOI**, **Author**, **Journal** and **Year** and other fields that are not important for a comparison such as **abstract**. However, Research Rabbit has two major limitations:

1. missing data for quantitative evaluation
2. potentially non-peer reviewed work

Fields like citations and type of publication are not provided by Research Rabbit but are necessary for quantitative evaluation metrics explained in chapter 3.3. For this limitation, we can fetch the missing data from another database with the DOI or title of the entries.
The second limitation of Research Rabbit is that there is no indication that the work found is peer reviewed. As a result, the relevance or scholarly credibility of some suggested works may vary.

3 Systematic vs. AI Literature Search

This thesis is mainly divided into two search process approaches: an SLS process and an AI search process with Research Rabbit.
This chapter will cover the structure and implementation of the algorithms and processes implemented. We will start with the structure and implementation of the SLS process. The implementation part covers all steps of the SLS process with the help of code examples. After the SLS process, this section will cover the implementation of the AI literature search process and the necessary data enrichment. The last section will describe the quantitative metrics that are used to analyze the two approaches.
Before we start with the processes, we need to select a topic of interest and find an initial paper. In the SLS process, the initial paper is used to generate keywords with KeyBERT. KeyBERT is a keyword extraction tool that uses BERT embeddings and cosine similarity to identify relevant words or phrases that best represent a document [8]. The reason for selecting an initial paper to generate keywords from, is that we want the input for both search approaches to be as similar as possible. In the AI process, the initial paper is needed to search similar work. The similar work is exported and then enriched with data from other databases for the comparison.

3.1 Systematic Literature Search Process

The SLS process can be seen in Fig. 3.1 and starts with the generating of the keywords from the initial paper. The generated keywords are used to start search queries in the databases. The results will then be saved separately as a .bib file and merged into a combined .bib file. Since the metadata varies from the databases, normalization must be done which will be described in section 3.1.3.
The SLS process consists of four main components which will be discussed in the subsections of this section. First, we describe how search queries are built for the databases. Secondly, we explain the handling of the query result by stating the important factors that are to be managed in the implementation of data handling. The combination of these databases ensures a diverse dataset that utilizes the strengths of each to minimize bias and maximize coverage of the research topic. In the end of the SLS process, we want to have one .bib file with all the query results with uniform format and without duplicates.

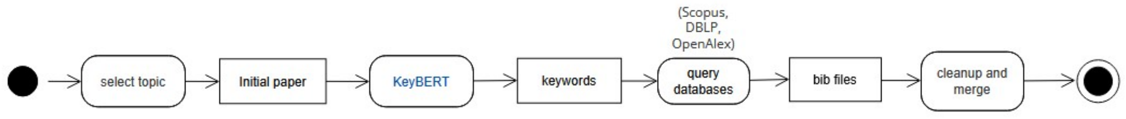


Figure 3.1: SLS process

3.1.1 Literature Search Queries

To receive data from the databases, we need to use the API from the databases and create an API call with different parameters depending on the database. These parameters are filters, number of search results and the keywords that are needed for the search query. Additionally, Scopus needs an API key to verify the access of the users as a parameter, DBLP and OpenAlex APIs are free to use and require no API key.
The literature search queries start after KeyBERT generated keywords from which the user chooses up to three keywords. For Scopus the API request consists of: the API URL, header parameters and query parameters.
The API URL is "https://api.elsevier.com/content/search/scopus" The header parameters are mandatory and contain the API key and the desired format of the response, which in our case is the JSON format. The query parameters are optional and depend on the desired output. We are using the following query parameters:

- **"query"** = "TITLE-ABS-KEY(**Keywords**)", which indicates that we want to search in the title abstract and author keywords of publications. **Keywords** is a text string that contains the keywords, multiple keywords can be combined with the "AND" or "OR" operators.

- **"start"** = "0" , **"count"** = "200", are needed for pagination, as per API response only up to 200 results can be received.

- **"sort"** = "citedby-count,relevancy", to sort the query results by citation count and then by relevancy.

Additionally, we cap the maximum number of results to 2000 to avoid receiving too many results.
The DBLP API uses the URL: "https://dblp.org/search/publ/api" followed by two query parameters:

- **"q"** = **Keywords**, which contains the keywords. **Keywords** is a text string which contains the keywords, multiple keywords are separated by spaces between words or the "|" operator, which indicates a logical OR.

- **"h"** = 1000, is the maximum number of results we want to receive.

It is important to note that the number of results receive is limited by DBLP to 1000 results per API request. Another important note is that DBLP searches in the title. This means that all of the keywords would need to be in the title of a publication to be found. For this reason, we decided to limit the amounts of keywords to one keyword of up to three words. The keywords could be separated by an "|" as a logical OR operator. As mentioned in the background chapter, we are only interested in peer reviewed publications, but DBLP does contain some non-peer reviewed literature. For that reason, we filter the results based on the types, taking into account only journals, books and InProceedings publications. However, without manually checking

all the entries from the response, there is always the possibility for non peer reviewed literature to appear in the results.
The OpenAlex API uses the URL: "https://api.openalex.org/works?search=" followed by the parameter:

- **"search"** = **Keywords**, which contains the keywords being searched in abstract and title of publications. **Keywords** are the same keywords with the same format as in the Scopus query.
- **filter=type_crossref:** = "journal-article|proceedings-article", indicates that we want only journal articles and InProceedings.
- **has_oa_accepted_or_published_version:** = "true", which allows us to filter for peer reviewed journal and InProceedings articles.
- **per-page** = "200" , **cursor** = "*", this is used for pagination, as per API response only up to 200 results can be received.

After creating the API URL with the parameters and filters, we create a GET request with the URL. From the response we select each entry and save it, this process is explained in the next section in detail.

3.1.2 Data Storage and File Handling

When receiving a response from the databases, we want to save the entries in a .bib file. In order to do that, we need to first define the fields that we want to save.
The entries that we save from the databases in .bib files are the following:

1. Title
2. DOI
3. Author
4. Type
5. Year
6. Journal
7. Note

Title is the title of the publication. This entry field is extracted from all databases, but some provide different formats. These formats could be abbreviations, having a dot at the end, or other differences. **DOI** is the DOI of the publication. This entry is not always extracted. Some publications do not have a DOI, or the database does not include it. The format of the DOI can also differ as some databases deliver the entry with the prefix "https://doi.org/" and other databases omit this prefix. **Author**

contains the authors of the publication. Since Scopus only delivers the first author, the other authors of DBLP and OpenAlex will be omitted, and we consider only the first author for all publications. This is done to make sure that all entries are uniform. **Type** is the type of publication, all the different databases have different names for publications. The publications are grouped into four categories: **InProceedings** for conference papers , **Journal** for journal papers, **Book** for books and book series and **Unknown** if the type is not known or none of the other types. Since the entries have different names in each of the databases, the types must be normalized, which will be discussed in the next section. **Year** contains the publication year. **Journal** is the name of the journal in which the publication is published. These entries also differ for the different databases, as they are sometimes abbreviated already. The **Note** entry is a flexible entry that can contain different types of information about the publications. The note field was used to store the number of citations that a paper has as an integer. An exception for this are all of the DBLP entries, since DBLP does not provide cited by count value. Thus, the note field is zero for all DBLP entries and all other entries that have not been cited by other publications.
The selected fields are extracted from the response of the database. This process is done for each of the databases separately and is showcased in Alg. 1. From the queries, we create multiple .bib files and combine to a single .bib file for an analysis. To perform this analysis, it is necessary to save the data without duplicates and normalize the entries.

Algorithm 1: Query_databases

```
   Input: API URL queryURL
   Output: Generated BibTeX file
1  response ← send GET request to queryURL and receive JSON data;
2  bibEntries ← empty list;
3  foreach entry in response do
4      DOI ← entry.DOI;
5      Title ← entry.title;
6      Note ← entry.cited_by_count;
7      Journal ← entry.journal;
8      Author ← entry.author;
9      Type ← entry.publication_type;
10     Create BibTeX entry with DOI, Title, Note, Journal, Author, Type;
11     Append BibTeX entry to bibEntries;
12 Save bibEntries as BibTeX file;
```

3.1.3 Normalizing Entries

To analyze the data, we ensure that the entries are in the same format. Most of the normalization is done immediately when the results are received from the database.

The **Year**, **Note**, **Journal** and **Title** entries are saved as received from the database and do not require normalization. For the DOI, OpenAlex database gives the whole URL address, while Scopus and DBLP only give the DOI string. Therefore, when saving the OpenAlex DOI entries, the DOI is shortened to save only the DOI string.
The **Type** and **Author** require normalization as they differ from database to database.
Starting with the **Type** entry of a publication, each database has their own names for publication types. As mentioned in the previous chapter, all types are grouped into four categories: Journal, Book, InProceedings and Unknown. For type normalization, a list with the mapping of types has been created where we map the retrieved type with our four categories. When an entry is fetched from the database, a function searches for the type in the mapping list and replaces the type name with one of the normalized type names.
For the **Author** we decided to save only the first author for all results of the query. This holds even for DBLP and OpenAlex which provide all the authors. The reason for this is to make sure that duplicates share the same author and that the .bib entries are uniform. The format for the author must also be normalized as there are inconsistencies between databases and even inside one database itself. Scopus always delivers the first author abbreviated; the author "Deniz Hagelstein", would be displayed as "Hagelstein D." for example. OpenAlex and DBLP return the first author in full length or sometimes abbreviated partially.
To normalize the authors, we decided to abbreviate the first author for OpenAlex and DBLP. There are many options on author name formats, which we need to cover if we want to abbreviate the author name. In the end, we want all the authors to be in the format of complete last name followed by initials of the first names with a ".". For this, the author name normalization Alg. 2 has been implemented. The Alg. 2 opens a .bib file and extracts the author from the .bib entry. The first step is to take the author string and separate it by name parts and initialize an empty list of abbreviated name parts. DBLP and OpenAlex always give the last name last, meaning we are assuming that the last name part is the last name. Then we check if the last name part of the name already has a "." indicating an abbreviation. In that case, we assume that the name is already abbreviated and go to the next author entry. If that is not the case, we check all the name parts except the last one for abbreviations. We then skip name parts if they are abbreviated, or abbreviate them if they are not and append them to a list of abbreviated name parts. When all the name parts have been iterated, we take the last name and conjunct it with the list of abbreviated name parts to build a new name. The last step is to update the .bib entry with the new author and save the new .bib file.
The correctness and limitations of this algorithm are discussed in section 4.1.

Algorithm 2: Normalize_authors

```
Input: BibTeX file
Output: Updated BibTeX file
foreach entry do
    author ← entry.author;
    name_parts ← split author by spaces or hyphen;
    abbr ← empty list;
    if name_parts[#parts - 1] contains "." then
        skip entry;
    for i ← 0 to length(name_parts)−2 do
        if name_parts[i] contains "." or length < 2 then
            skip name_parts[i];
        else
            Add first letter of name_parts[i] + "." to abbr;
    lname ← name_parts[#parts - 1];
    newAuthor ← lname + abbr;
    entry.author ← newAuthor;
Save BibTeX file;
```

3.1.4 Merging and Duplicate Handling

Now that all the entries are normalized, the next step is to merge the .bib files into a new .bib file containing all entries. After merging, the combined .bib file should not contain any duplicates as they could distort the analysis and comparison. Too many unnoticed duplicates will lead to false results and thus will lead to false assumptions in comparisons of quantitative metrics. For that reason it is important to be able to find and handle duplicates when merging the entries.
In order to find and handle duplicates we have implemented Alg. 3. First the algorithm opens all the .bib files and extracts the entries from the .bib files as a list. We also create an empty list for the unique entries. Then we iterate through all entries and check if they are already in the list of unique entries. For that we follow two steps.
Step one is to check if the current entry has a DOI that is already in one of the entries of the unique entries. In that case we will skip the entry and continue with the next entry. If that is not the case, then the entry will be further analyzed in step two.
Step two covers cases where the DOI is missing in either the list of unique entries or in the entry that is currently checked. Besides the DOI, there are three different entry fields that can indicate duplicates: These entry fields are: **Year**, **Author** and **Title**. If the **Year** of the new and the existing entries do not match, it will not be considered a duplicate. Therefore, if the year is not already in the merged entry list then the entry can be inserted into the merged list. If the **Year** matches, the author and title of the entry will be compared to the existing titles and authors. Since the titles and

authors can have a differences, such as special characters, we use the fuzzy matching methods by M. Bachmann [9]. The reason for using fuzzy matching methods and not exact matching, is that titles or authors can have variations caused by special characters, diacritics, and minor typographical errors. Therefore a fuzzy matching algorithm determines the similarity of the titles and authors and returns a number in the range of [0,100] depending on the similarity. The similarity score threshold of the title is set to be 90% , and the threshold for the authors is set to 75%. We lowered the threshold of the author similarity since the authors last name can be as short as 2 characters, meaning minor differences already lower the score of similarity significantly.
Two entries are considered a duplicate if the title and author thresholds are exceeded and the years are equal. The algorithm operates on a first-come, first-served basis, thus the first encounter of a duplicate is kept while the subsequent encounters are disregarded.
If all the entries are iterated, the merged entries will be saved in a separate merged.bib file to perform the experiments on.

Algorithm 3: Combine_bib_files

```
Input: Three BibTeX files: openalex.bib, DBLP.bib, Scopus.bib
Output: Merged BibTeX file: merged.bib
1  mergedEntries ← empty list;
2  allEntries ← load entries from openalex.bib, DBLP.bib, Scopus.bib;
3  foreach entry in allEntries do
4      if entry.DOI exists and DOI is already in mergedEntries then
5          skip entry;
6      else if entry.year exists in mergedEntries then
7          if fuzzy match(entry.title, mergedEntries.title) > 90% and
8          fuzzy match(entry.author, mergedEntries.author) > 90% then
9              skip entry;
10         else
11             Append entry to mergedEntries;
12     else
13         Append entry to mergedEntries;
14 Save mergedEntries as merged.bib;
```

3.2 AI Literature Search Process

In this section, we explain the Research Rabbit search process which can be seen in Fig. 3.2. Furthermore, we go over the solutions we implemented for the limitations and restrictions that come from using Research Rabbit.

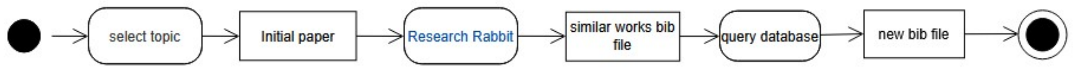


Figure 3.2: AI process

3.2.1 AI Search Query

The AI query starts by entering the initial paper into Research Rabbit. After inserting an initial paper, Research Rabbit searches for publications that are similar to the inserted initial paper. When Research Rabbit finds similar works, it creates a citation graph that show how the papers are connected. Additionally the number of similar works is displayed and they can be selected and exported as .bib file. The exact process for finding similar work is not disclosed by Research Rabbit.
The resulting export .bib file contains the similar works with data provided by Research Rabbit. The exported data contains the title, DOI, journal, year and the author of the similar works but not the cited by count or publication type.
Additionally, the data is incomplete in some cases, meaning that even if a field exists, it can have the value "null". These cases, and the fact that we are missing data like citation and publication type, make data enrichment necessary for a comparison with the SLS.

3.2.2 Enriching Research Rabbit Entries

To add the missing data, it was decided to perform an OpenAlex query for each of the Research Rabbit export entries. OpenAlex has been chosen for its high API call limit and because Research Rabbit itself is querying OpenAlex [11].

For the data enrichment we have implemented Alg. 4. The algorithm iterates through all the entries of the Research Rabbit export. We also initialize a new empy list matchedEntries in which we want to save all the new entries. For each entry we take the title and create a GET request for OpenAlex. The API URL looks like this: "https://api.openalex.org/works?filter=title.search:TITLE" where TITLE is the title of the export entry that is searched. We could also add the filters discussed in 3.1.1 to filter out non-peer reviewed entries. However, this would mean that we have to filter out all of the books from the export hence we decided to include possible non-peer reviewed entries.
By taking the title as a parameter, we will receive multiple items since the database also returns publications with similar titles. We take the search results and iterate through them to look for a correct match. Matches can be found in two ways:

1. exact match via DOI: both the original and result contain a DOI and they match.

2. Fuzzy match by Metadata: we check if the years are matching and whether the title and author fields have fuzzy string similarity greater than 90%.

If a match is found, it gets saved in a new .bib file and we continue with the next export entry until all entries are iterated. If no match is found the entry gets skipped. Since OpenAlex does not contain all publications, there is also the possibility titles are not found. For these cases, we do not save the old export data in the new .bib file to keep consistent and uniform data. Data enrichment is needed for the quantitative metrics which we discuss in the next section, however not all experiments we conduct need this enriched data. Saving the data in a new file makes sure that the export data from Research Rabbit is not lost during the process.

Algorithm 4: Enrich_RR_Entries

```
Input  : BibTeX file RRexport.bib
Output: Matches written to newRR.bib
Load entries from RRexport.bib into bibEntries;
foreach entry in bibEntries do
    Set foundMatch ← false;
    Query OpenAlex using entry.title, store results in results;
    if results is empty then
        Continue to next entry;
    foreach result in results do
        if entry.DOI exists and result.DOI exists and entry.DOI ==
          result.DOI then
            Write result to newRR.bib;
            Set foundMatch ← true;
            break;
        else if entry.year == result.year and
        fuzzyMatch(entry.title, result.title) > 90% and
        fuzzyMatch(entry.author, result.author) > 90% then
            Write result to newRR.bib;
            Set foundMatch ← true;
            break;
    if foundMatch == false then
        Continue to next entry;
```

3.3 Quantitative Evaluation

After saving the .bib files from either literature search process, an evaluation will be performed manually in one of the experiments. In order to do that, a quantitative analysis concept has been implemented which creates graphs/charts for 7 different metrics from the data:

1. **Data distribution**: Creates a graph that shows how many results are collected from the AI and SLS approach. This is used to highlight the distribution of the

data sources.

2. **Most cited publications**: Creates a table showing the top 10 most cited papers from the data set.

3. **Citation distribution**: Creates a plot of the citations of the publications by grouping them into bins. This helps to find a trend of research of the topic, if there are a lot of publications with a lot of citations, we can assume that the topic is researched on.

4. **Papers per author**: Creates a graph that shows how many authors have multiple publications which can indicate the author's performance.

5. **Publications per journal**: Creates a graph that shows the journals with the most publications. This can potentially highlight important journals for that specific topic.

6. **Yearly trend of publications**: Creates a plot that shows the year of publications and the number of publications to indicate the yearly trend and overall trend of research on that topic.

7. **Publication type**: This creates a pie chart that shows the distribution of publication types.

Having created the visualizations of the data and these seven metrics, we can further analyze the differences of the results of the AI-based search and SLS.
To further look for differences, an algorithm has been implemented that searches for identical papers to test if the results are similar from the two approaches. This algorithm uses a fuzzy matching function and compares all titles of two .bib files. Additionally the abstract of the top 10 most cited papers has been read through to see if the topics are matching the initial paper, which provides a small qualitative evaluation. However, the evaluation focuses mainly on quantitative evaluation methods by comparing the seven different metrics chosen.

4 Evaluation

In the previous chapter, we reviewed the quantitative evaluation metrics. This chapter will discuss the correctness of the implemented algorithms, define an experimental setup for the general process and compare the two search approaches with three experiments. Additionally we will make setups for keywords, queries and the comparisons. First, the implemented algorithms will be tested for correctness by reviewing test cases that have been created. Then the results of the queries will be discussed and visualized according to the processes we have defined in the previous chapters. In the end, the result will be analyzed and compared to each other.

4.1 Correctness

In this section, we will discuss the tests that have been done for the implemented algorithms in regard to their correctness. To do that, we have defined test cases for algorithms where a particular result is expected. In particular, we will take a look at the algorithms Alg. 2 and Alg. 3 and discuss the test cases for the correctness.
For Alg. 2, we defined eight test cases in a .bib file with various author names in each entry. After the execution of the algorithm, we expect each of these eight entries to be either correctly abbreviated or kept unchanged if the name is already abbreviated or not in the expected format. The test cases and results can be seen in table 4.1; the table showcases the input and the output after one execution of the algorithm. From the test cases, we see that the algorithm works for names that have the format of n first names and only one last name at the end. The abbreviation even works if a first name is already abbreviated or two first names are connected via a hyphen. If an author has multiple last names, the algorithm treats the name at the end as the only last name while the other last names will get treated as first names. This means that if the name is in a different format like last name and n first names, the output would be the nth first name followed by the initials of all the other name parts led by the initial of the last name which we would not consider a correct abbreviation.

Name before abbreviation	Name after abbreviation
Deniz Hagelstein	Hagelstein D.
Deniz Test Hagelstein	Hagelstein D.T.
Deniz-Test Hagelstein	Hagelstein D.T.
Deniz T. Hagelstein	Hagelstein D.T.
Hagelstein D.	Hagelstein D.
Deniz Hagelstein and Test Testcase	Hagelstein D.
D. Hagelstein	D. Hagelstein
Hagelstein	Hagelstein

Table 4.1: Author abbreviation test cases

In conclusion, the algorithm does work correctly for names with the expected format of first name followed by the last name or already abbreviated names, and does not work if there is only a singular name or a different format.

For Alg. 3 we have created two test .bib files with 15 entries each that will be merged with the algorithm. The algorithm should identify and omit duplicates and merge only unique entries. The .bib files contain 10 duplicates and 5 unique entries, in total the merged.bib file should have 20 entries.
Since duplicate detection happens by fuzzy matching the title, the author and matching the year or DOI (if present), we have created the duplicates in regards to cases where these fields have minor differences like hyphens, a dot at the end, or missing characters.
Since the DOI should only get compared if both entries have a viable DOI, a test case for this case was also concluded successfully.
One weakness of this algorithm is that if two entries share the same DOI but have different metadata, they are still considered a duplicate, meaning if one database gives out a wrong DOI, the entry could wrongfully be marked as a duplicate.
Another weakness is that the algorithm does also not find duplicates if the titles of the entries are short and different, as an example we take the two titles:

1. "new IT systems"
2. "new IT-systems."

For these two titles, the fuzzy match ratio function would get a score of 89,65%, the threshold however is 90% and thus these two titles would not be considered duplicates.
An edge case would be if titles are subsets of other titles and the entries have no DOI. That could mean the entries are duplicates because one title was abbreviated by a database, or they are not duplicates, they just share a similar title. In this case the algorithm would conclude that the two entries are not duplicates and both entries will be included in the merged.bib file.
This concludes that even when not solely relying on the DOI, not all duplicates can be found with the algorithm alone.

4.2 Experimental Setups

In the following sections, we want to create experiments for the processes implemented and evaluate them. The goal for these experiments is seeing how the results of the two approaches compare to each other. First, we will describe how the initial paper was selected and how the data was handled from the systematic and the AI approach. Then, three different experiments will be explained and evaluated. The first experiment discusses the two approaches and displays the differences and overlaps of the results. The second experiment will show the quantitative analysis results of the enriched Research Rabbit export and compare it to the result of the SLS. The third experiment is to see if a different input paper for Research Rabbit can yield different results. To show that, three additional papers are being queried separately in Research Rabbit and will then be compared to the AI query made with the initial paper and to the result of the SLS.

4.3 Experiment 1

Experiment 1 focuses on the SLS and AI approaches, we start by selecting a topic and finding an initial paper. In particular, we are interested in how many overlaps exist in the results of the two approaches. For this we perform an SLS and an AI search query.
We start by making the assumption that at least 10% of the results from the Research Rabbit export will overlap with the SLS results.
Therefore we define the match rate of an AI query with the SLS to be:

$$\text{Match Rate} = \frac{\text{Number of matches with SLS}}{\text{Total number of AI results}} \tag{4.1}$$

To find the initial paper, the topic "UPPAAL" was selected which is a model checker for real-time systems. A query on Scopus has been performed with "UPPAAL" as a keyword. From the results we selected the paper "Testing Real-Time Systems Using UPPAAL" [5] as the initial paper to start the processes.
The paper's abstract and title have been used as input for KeyBERT and keywords have been generated. We selected the two keywords: ["UPPAAL","real time"]. After selecting keywords, we performed the queries for the three databases first.
For DBLP the search query was done only with the Keyword "UPPAAL" since multiple keywords would change the output of the queries as described in the approach section. For Scopus and OpenAlex the query keywords were "UPPAAL AND real time". The results of the queries are displayed in table 4.2. As we can see in the table we have 1571 total results before merging. After the merge of the .bib files, we got 1395 results, meaning that we have found 176 duplicates in the query results. Out of these 176 duplicates, 162 (92%) have been solely via DOI matching and the remaining 14 (8%) have been found via the implemented matching algorithm.

The next step is to perform the AI query. For this we take the initial paper and insert it into Research Rabbit. When inserting the initial paper into Research Rabbit, we get 996 similar works which we then export. The results of both approaches can be seen in the Fig. 4.1.

Comparing the 996 export entries with the 1395 results from the systematic approach, we match 31 results. We match the results via fuzzy matching the title with a ratio of 90%, we also look for matches with the DOI. The matching with the DOI however, is lower with only 23 matches. This can be caused by the missing metadata from Research Rabbit which we get after data enrichment as the DOI is not always provided by Research Rabbit. Coming back to the assumption, we notice that we

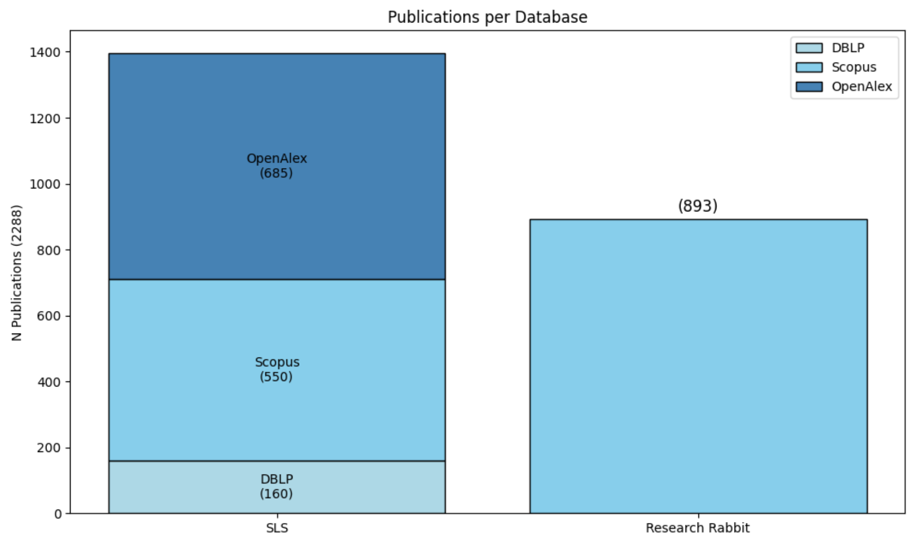


Figure 4.1: Paper Sources

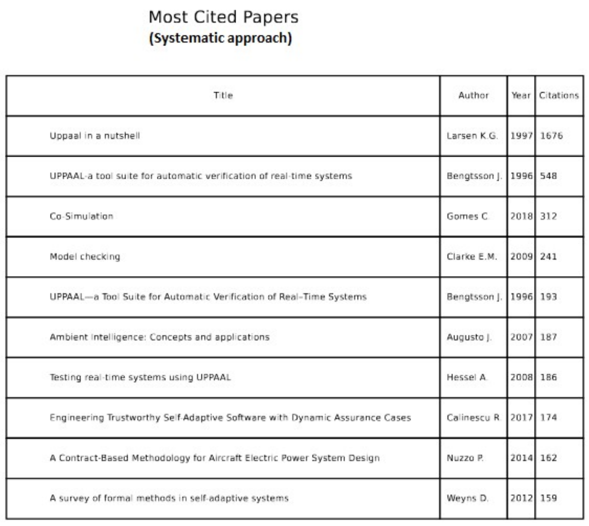
Data Source	Input	Results	Results without duplicates
Scopus	"UPPAAL" AND "real time"	555	550
DBLP	"UPPAAL"	238	160
OpenAlex	"UPPAAL" AND "real time"	778	685
Research Rabbit	Initial Paper	996	996

Table 4.2: Query Results

only match 31 results, meaning the match rate of the AI query with the initial paper is 3.11%, which is way lower than expected.
Possible reasons for the assumption not holding are bad choice of keywords for the SLS, bad choice of databases for the systematic search or the AI research tool was not the ideal choice. KeyBERT generated the keywords for the SLS, if keywords would be the issue, then KeyBERT would be a weakness and could require some improvement.

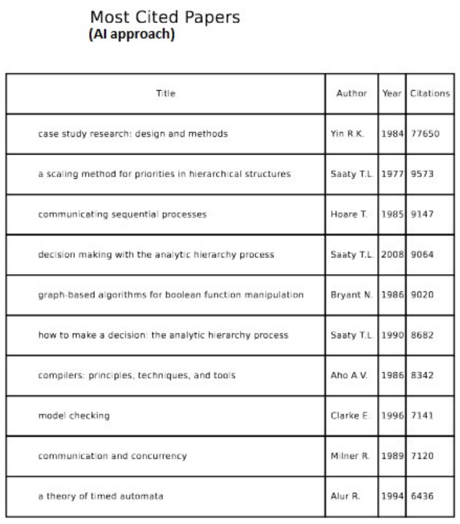
4.4 Experiment 2

In experiment 2 we will take a look at the results of the quantitative analysis and compare the graphs. We have performed a query with the initial paper in Research Rabbit and an SLS with keywords generated from the initial papers abstract and title in the previous Experiment. We use these two results now to perform quantitative analysis based on the metrics we defined in section 3.3. Now we need to enrich the Research Rabbit export data of the AI query. For that we use OpenAlex API as mentioned before.



Most Cited Papers
(Systematic approach)

Title	Author	Year	Citations
Uppaal in a nutshell	Larsen K.G.	1997	1676
UPPAAL-a tool suite for automatic verification of real-time systems	Bengtsson J.	1996	548
Co-Simulation	Gomes C.	2018	312
Model checking	Clarke E.M.	2009	241
UPPAAL—a Tool Suite for Automatic Verification of Real–Time Systems	Bengtsson J.	1996	193
Ambient intelligence: Concepts and applications	Augusto J.	2007	187
Testing real-time systems using UPPAAL	Hessel A.	2008	186
Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases	Calinescu R.	2017	174
A Contract-Based Methodology for Aircraft Electric Power System Design	Nuzzo P.	2014	162
A survey of formal methods in self-adaptive systems	Weyns D.	2012	159



Most Cited Papers
(AI approach)

Title	Author	Year	Citations
case study research: design and methods	Yin R.K.	1984	77650
a scaling method for priorities in hierarchical structures	Saaty T.L.	1977	9573
communicating sequential processes	Hoare T.	1985	9147
decision making with the analytic hierarchy process	Saaty T.L.	2008	9064
graph-based algorithms for boolean function manipulation	Bryant N.	1986	9020
how to make a decision: the analytic hierarchy process	Saaty T.L.	1990	8682
compilers: principles, techniques, and tools	Aho A.V.	1986	8342
model checking	Clarke E.	1996	7141
communication and concurrency	Milner R.	1989	7120
a theory of timed automata	Alur R.	1994	6436

Figure 4.2: Most Cited Publications

Initially, since Research Rabbit's data is not guaranteed to be peer reviewed, but the systematic approach is, we wanted to only compare peer reviewed data with each other. This means that when enriching the data from Research Rabbit we need to include filters so that the works that are not peer reviewed get filtered out. However of the 996 export entries, we failed to retrieve information for most of them and only got data from 207 entries. This does not necessarily mean that the other 789 results are not peer reviewed; it could be that OpenAlex does not contain or find them, or they are books and thus are filtered out.
Since it did not make sense to compare 1000+ works from the systematic query to only 207 works, we decided to take into account possible non-peer reviewed literature from Research Rabbit to see if the trends and graphs are similar.
Without the filter for peer reviewed works, we got 893 entries enriched from the 996 Research Rabbit export data, meaning 103 entries were not found in OpenAlex. With these 893 works, and the result from the systematic query, we now perform the quantitative analysis to go over the evaluation metrics mentioned in section 3.3.
Starting with the data distribution, we see in Fig. 4.1 the distribution of the databases for the SLS (after removing the duplicates) and for Research Rabbit. The next metric to be looked at is the most cited publications. From the Fig. 4.2 we can see that there are no overlaps of papers in the top 10 cited papers, except one journal paper "model checking" by Clarke E.M. which does not have the same publication year. When checking the entry, it was noticed that OpenAlex contained the paper twice with both of the publication years, meaning that OpenAlex could have a wrong entry or the journal paper was published twice.
The next metric to look at is the citation distribution. we see in Fig. 4.3 that the distribution of citations is very different. From systematic approach results we see that we only have few works that have been cited more than 100 times, and even less

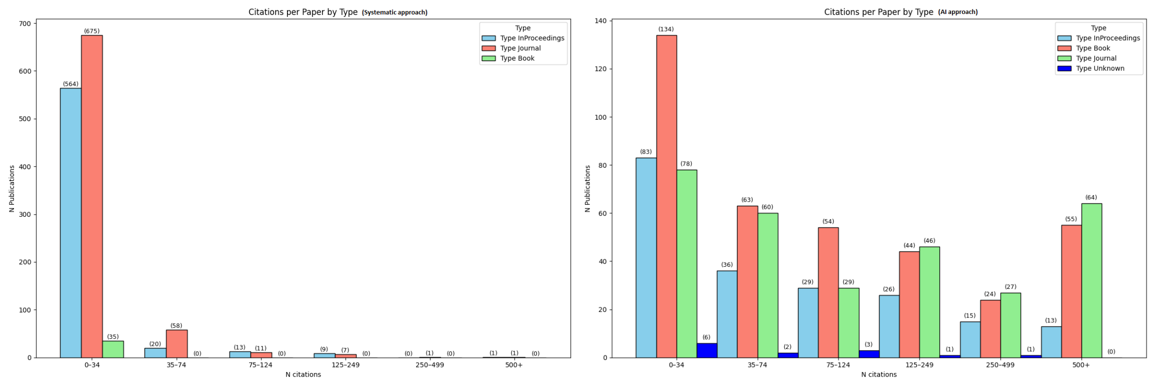


Figure 4.3: Citation Distribution

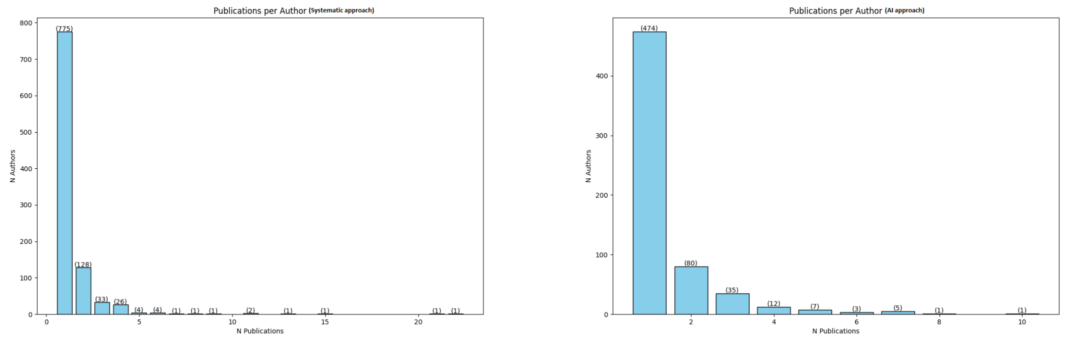


Figure 4.4: Publications per Author

works the higher the citations become. The AI approach results do not only contain more works in the range of 100+ citations but contain papers with more than 500 citations which the systematic. While most papers in the SLS have citation counts below 124, Research Rabbit publications tend to have more citations. This indicates that Research Rabbit focuses on citations for finding publications. In Fig. 4.4 we see the papers per author distribution, and can notice that both graphs show that most authors have only one publication, however we can also see that there are more authors with more publications gathered from the systematic approach than from the AI approach. In Fig. 4.5 we can see the papers per journal distribution which can indicate important journals. We can see in both of the approaches the LNCS (Lecture Notes in Computer Science) journal as the top journal as well as some other journals like theoretical computer science journal and IEEE Transactions on Software Engineering. While there are some journals that have a high number of publications in the systematic approach result that do not appear in the AI result. We also see that the number of unknown journals in the AI approach is 242, while the number of unknown journals in the systematic approach is only 77.

Fig 4.6 shows the yearly trend of publications by type. This figure can help to

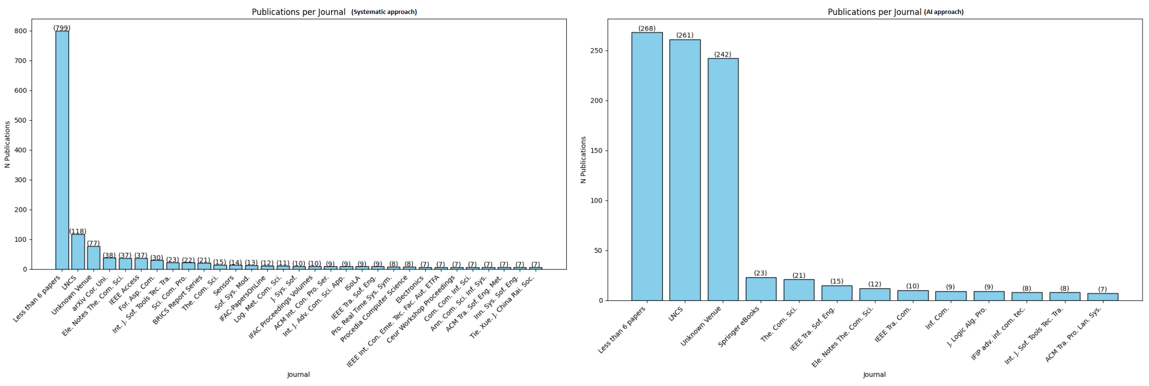


Figure 4.5: Publications per Journal

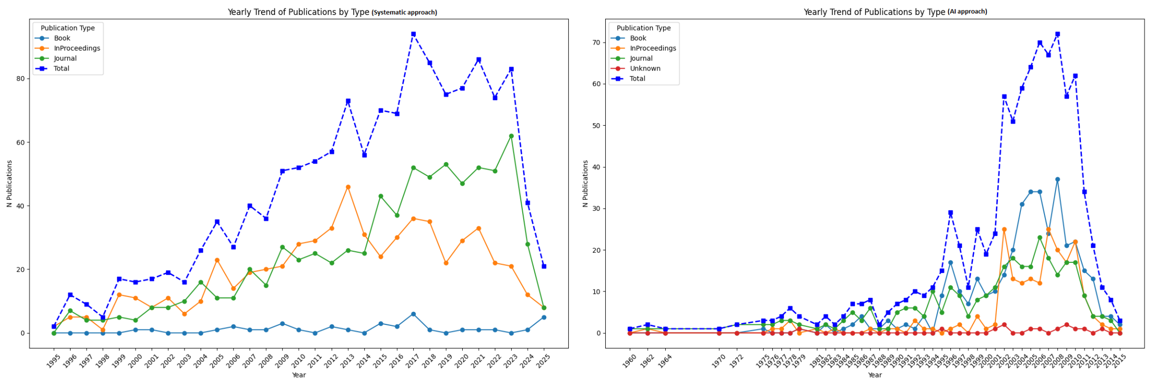


Figure 4.6: Yearly Trend

identify trends of publications to see when the most research on a given topic has been published. From the graphs we can see that both have a peak of total publications around 2009, but we also see that the publications from there on have decreased in the AI approach result while they have increased in the systematic approach result, which is clearly a mismatch between the two results. We can also see from the AI approach result that there was a peak of book publications in the years from 2004 to 2010, which is not seen in the systematic approach result. The last evaluation metric is the distribution of the types of the publications; the results are shown in Fig 5.7. From the pie charts we can clearly see a different set of distributions, the books in the systematic approach make up only 2.6% while they make up almost 42% of the results of the AI approach. This could be due to filtering of OpenAlex results from the systematic paper queries, as we wanted to omit non-peer reviewed papers for the systematic approach. It is also possible that the 103 that were not enriched and thus omitted in the quantitative analysis results, could be of type journal, book or InProceeding meaning the distribution could look slightly differently.
As the graphs show, there are more journal papers than there are InProceedings from both queries.

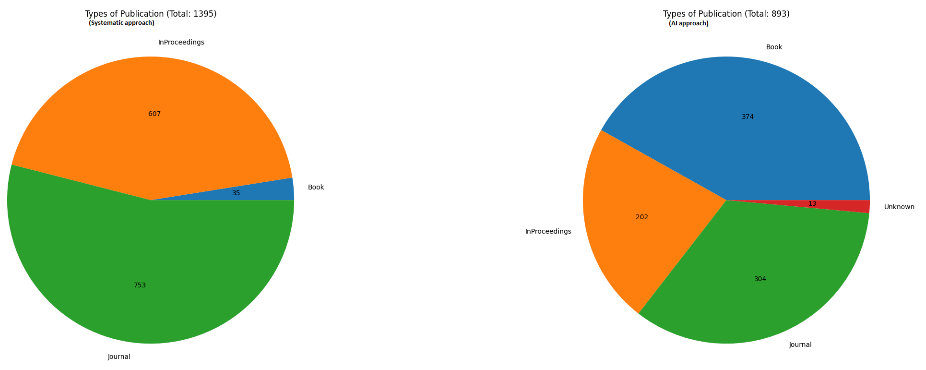


Figure 4.7: Publication Type Distribution

4.5 Experiment 3

Experiment 3 takes three different papers from the SLS results and uses them as input for Research Rabbit queries. We use the same result of the SLS and compare it to other AI queries with the 3 different papers. As in experiment one, we use the term "match rate" as the percentage of matches of the AI query result with the SLS divided by the total number of AI query result for that particular query.
We want to know much the selection of the initial paper changes the outcome of the results. Our hypothesis is that the match rates with the SLS are similar to the initial AI query (3.11%) from experiment one.
The new papers used for different queries have been chosen at random from the SLS result. and are the following:

1. the first one is "MDE-Based Verification of SysML State Machine Diagram by UPPAAL" by Huang X. et al. [6].
2. the second one is "Formal Verification of Device Discovery Mechanism using UPPAAL" by Arry S. et al. [12].
3. the third one is "Efficient Guiding Towards Cost-Optimality in UPPAAL" by Behrmann G. et al. [1].

The result of the new queries and their matching rate can be seen in table 4.3.
For the test cases, the cited by count does seem to influence the number of similar works found. The test cases have the following cited by counts: case 1 has been cited 5 times, case 2 has been cited 2 times and case 3 has been cited 39 times. These cited by counts are from the results of the SLS.

Our assumption holds in test case 1 with a match rate of 5.03%, which is higher than the 3.11%. For test case 2, our assumption does not hold as the match rate with the SLS results is 2.4% which is lower than 3.11%.
Test case 3 matches 42 entries with the systematic search results, yielding a match

Research Rabbit input	Results	Matches with SLS Query	Match Rate with SLS
Initial paper	996	31	3.11%
Test Case 1	159	8	5.03%
Test Case 2	127	3	2.4%
Test Case 3	313	42	13.41%

Table 4.3: AI Queries Comparison

rate of 13.41% which is significantly higher then the 2.2% match rate from the initial AI query.
One important note is that only test case 3 contains the initial paper that was used for the initial AI query and the SLS in the exported data while test case 1 and 2 did not.
This concludes the experiment indicating that the choice of the input paper for Research Rabbit does influence the match rate with the SLS. However, to verify this claim, more testing would have to be done while also not choosing the papers at random but trying to match the content of the papers.

5 Conclusion and Future Work

In this thesis we implemented processes for a systematic and an AI literature search approach and performed three experiments on the results with the goal to see how they compare. From the results of each experiment we can see that the results of the two approaches do not overlap significantly, and thus they are not similar in most aspects. It is, however, to be noted that one reason for this could be the choice of keywords being used; if we change the keywords the results may be different. Another reason could be the omitting of non-peer reviewed data when performing the SLS but having no way to do that for the AI literature search. While the experiments yielded initial results, definitive conclusions would require a broader range of topics and a more extensive set of test cases to ensure the robustness and generalization of the comparison.

For the future work, there are several areas where the two processes could be improved. One important area is the metadata of the systematic queries, as DBLP does not provide any citations. A way to add citations to the results from DBLP would increase the consistency and correctness of the data. One way to implement this could be via querying another database and enriching the data from DBLP, or use another database instead of DBLP. Web of Science is a big database that was considered for the SLS but during the time of implementation, the access was not approved, hence Web of Science was not used. This would also solve the problem of how the search is performed, as DBLP can only search in the title. A different database might be better suited for keyword searching as more keywords could be used to get more precise results.
Another part of the systematic approach that needs to be improved, is the keyword generation. The quality of the keywords generated by KeyBERT can vary depending on the input text; ideally one would like to enter a whole paper or even multiple papers. This was not implemented, as in the process of putting in a whole paper, the quality of the keywords drastically worsened as footnotes and headers of PDF files were included in the analysis. To improve the generation of keywords with KeyBERT there needs to be done more exploration and testing. However, adding an option for choosing whole papers could be a great improvement when having multiple choices of initial papers to have more concise topics.
For the merging step, when a duplicate is found, the duplicate will be omitted, which could mean that metadata that is not present in the current entry but is part of the duplicate, also gets omitted. In the future, it should be implemented that the data is not omitted but a new entry is merged out of the duplicates to get as much data as possible.
For the quantitative analysis, a way for journal abbreviation is needed, as the current way is just taking the strings and shortening them; there may be a way to handle this more efficiently. Some databases for example already give an abbreviated version as the journal, this means that if one database is giving a journal in an abbreviated form,

and another database uses the same journal but not abbreviated, they are not treated as the same journal and could potentially distort the graphs from the quantitative analysis.
The last aspect of the comparison with AI tools would be considering different AI tools that do not need initial papers for querying for papers or a tool with an API to automate the process. Another AI tool could also solve the the issue with the peer reviewed status of the papers, as Research Rabbit does not guarantee that a work is peer reviewed, so there may be a tool that does focus on only peer reviewed works.

Bibliography

[1] Gerd Behrmann, Ansgar Fehnker, Thomas S. Hune, Kim G. Larsen, Paul Pettersson, and Judi Romijn. Efficient guiding towards cost-optimality in uppaal. *BRICS Report Series*, 8, 2001.

[2] Silvia Bonfanti, Angelo Gargantini, and Atif Mashkoor. A systematic literature review of the use of formal methods in medical software systems. *Journal of Software: Evolution and Process*, 30:e1943, 2018.

[3] Lutz Bornmann and Rüdiger Mutz. Growth rates of modern science: A bibliometric analysis based on the number of publications and cited references. *Journal of the Association for Information Science and Technology*, 66:2215–2222, 2015.

[4] Yongfan Fu, Jian Luo, Guofang Nan, and Dahui Li. Peer review expert group recommendation: A multi-subject coverage-based approach. *Expert Systems with Applications*, 264:125971, 2025.

[5] Anders Hessel, Kim G. Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. *Testing Real-Time Systems Using UPPAAL*, pages 77–117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[6] Xiaopu Huang, Qingqing Sun, Jiangwei Li, and Tian Zhang. Mde-based verification of sysml state machine diagram by uppaal. In Yuyu Yuan, Xu Wu, and Yueming Lu, editors, *Trustworthy Computing and Services*, pages 490–497, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[7] Peder Olesen Larsen and Markus von Ins. The rate of growth in scientific publication and the decline in coverage provided by science citation index. *Scientometrics*, 84:575–603, 2010.

[8] Maarten Grootendorst. Keybert: Minimal keyword extraction with bert. 10.5281/zenodo.4461265, 2025. Accessed: 2025-06-20.

[9] Max Bachmann. Rapidfuzz. https://zenodo.org/records/15133267, 2025. Accessed: 2025-06-20.

[10] OpenAlex. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. `https://docs.openalex.org/api-entities/works/filter-works`, 2025. Accessed: 2025-06-20.

[11] ResearchRabbit. ResearchRabbit: AI-powered literature discovery tool. `https://www.researchrabbit.ai`, 2025. Accessed: 2025-06-20.

[12] Amardeep Kaur Shivangi Arry. Formal verification of device discovery mechanism using uppaal. *International Journal of Computer Applications*, 58:32–37, 2012.

[13] Davide Tosi. Comparing generative AI literature reviews versus human-led systematic literature reviews: A case study on big data research. *IEEE Access*, 13:56210–56219, 2025.