Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Master in Data Science

Master Thesis

SALTBot: Linking Software and Articles in Wikidata

Author: Jorge Bolinches Segovia

Madrid. July, 2024

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Master Thesis
Máster en ***Ciencia de Datos***

Title: **SALTBot: Linking Software and Articles in Wikidata**

Month, Year

Author: **Jorge Bolinches Segovia**

Supervisor:
Daniel Garijo Verdejo
ETSI Informáticos
Departamento de Inteligencia
Artificial (DIA)
UPM

Abstract

Research Software is becoming a recognized first-class citizen to support and reproduce the results of scientific investigations. However, the link between software and their corresponding articles is often absent from Knowledge Graphs like Wikidata, thus making it challenging to retrieve implementations of existing papers. The purpose of this project is to present an approach to improve linking between articles and software.

This approach is implemented in SALTbot, the Software and Article Linker Toolbot, an implementation able to join scientific articles and the software they implement in any Wikibase instance. In addition, SALTbot will automatically describe software and article entities with metadata.

Furthermore, this project will also aim to link software in repository datasets with their corresponding articles in Wikidata, provide insights about the results of said linking and make indications about how to use SALTbot to increase visibility and dissemination of scientific articles and their software in knowledge graphs.

Contents

1 Introduction

In the context of research, papers are generated to present findings, and these papers may include artifacts, including software. Therefore, the tangible programs, algorithms, and other tools developed during the investigation serve as artifacts that complement the scientific literature, which describes the innovations and contributions of the research.

Research Software refers to the scripts, tools or computational pipelines developed throughout an investigation to support the main findings described in a scientific publication [1] . Research Software is becoming increasingly recognized as a research product[1], and the scientific community has developed software citation principles [2] and citation formats [3] to recognize developers with the appropriate credit.

Although these two types of artifacts will be developed simultaneously during the majority of the investigation, it is common to see a split between both after the publishing of the results. While the scientific literature will be published in journals, conferences or in specific online repositories such as arxiv[2], software will most likely be stored in dedicated software repositories such as GitLab[3] and GitHub[4], that is, if the software is free and open source.

This separation presents a traceability problem when trying to find the associated software of a paper and vice versa, especially when articles don't present an explicit mention of where the software is stored or the software doesn't have a correct citation of the article. This traceability problem raises the following challenges when understanding and applying research, as well as when reusing software:

- **Lack of Context**: Without a clear link between the software and its associated scholarly article, users or other researchers may lack the necessary context to understand the purpose, functionality, and intended use of the software. The article provides the theoretical background, experimental results, and contributions, while the software offers a practical implementation of those ideas. Linking them ensures that users have a comprehensive understanding of the research and its practical implications.

[1] https://sfdora.org/read/
[2] https://arxiv.org/
[3] https://about.gitlab.com/
[4] https://github.com/about

- **Reproducibility Challenges:** Reproducibility is a cornerstone of scientific research. When software and its scholarly article are not properly linked, it becomes difficult for other researchers to reproduce the results or verify the claims made in the paper. The software implementation may include specific configurations, dependencies, or optimizations that are crucial for achieving the reported results. Without access to the software or clear instructions, it becomes challenging to reproduce the experiments or build upon the work.

- **Missed Opportunities for Improvement:** The scholarly article typically describes the innovations, methodologies, and results, while the software implementation provides practical details, optimizations, and potential extensions. By not linking the two, researchers may miss out on opportunities to improve or enhance the software. The software may contain valuable code, libraries, or techniques that can be reused or built upon, leading to further advancements in the field. Linking the software and the article allows for a more comprehensive understanding and promotes collaboration and innovation.

- **Ineffective Knowledge Transfer:** When software and scholarly articles are disconnected, it becomes challenging to transfer knowledge effectively from the researchers to the wider community. Other researchers, practitioners, or industry professionals may have a keen interest in leveraging the software for their work or applying the research findings to their domain. However, without clear links, it becomes harder to disseminate knowledge and ensure that the research has a broader impact.

To avoid these kinds of problems, we must find a solution to link both articles and software in an easy and automatically retrievable way, while maintaining the independence of software repositories and scientific publications. In this project, we propose the usage of linked data technologies as a method of linking scientific articles and the software they implement to promote a better visibility of software related research.

In this work, we address these issues by presenting SALTbot, a Software and Article Linker Toolbot[5], designed to find article and software entities in Wikibase instances in order to enrich and link them together. SALTbot takes as input one or multiple GitHub repositories and inspects them for references to existing articles in Wikidata. Then, if found, SALTbot will link software and their corresponding articles, creating a new software instance when necessary and enriching it with metadata.

[5] https://github.com/SoftwareUnderstanding/SALTbot

Our work includes two main contributions:

- A workflow designed to link software and articles with minimal user intervention, based on a manual analysis of dozens of software repositories with a link to a publication.
- SALTbot, an end to end implementation of our workflow.

We have validated SALTbot manually by assessing its performance in 597 GitHub repositories with citation files. As a result, we have added 40 new scholarly article entries, 95 new software instances, over 500 metadata

2 Background and State of the art

In this section, we will introduce some of the previous efforts at solving the problem separated research software and scholarly articles. Besides that, we will also introduce some of the technologies on which SALTbot is based upon.

The structure of this section is the following:

- **2.1 Previous works:** previous attempts and contributions at establishing relationships between papers and software in the scientific literature

- **2.2 Wikidata and Wikibase:** a brief introduction about the main knowledge graph we targeted as well as a description of its underlying technology

- **2.3 Wikibase bots:** an introduction about automated editors in Wikibase instances, as well as the category in which SALTbot falls as a software implementation

- **2.4 Software Citation Files:** a description about citation file formats and conventions

- **2.5 SOMEF:** an introduction to the software metadata extraction framework, our main way to extract metadata from repositories

- **2.6 CodeMeta:** a schema about characterizing software in knowledge graphs

- **2.7 OpenAlex:** a description of the scholarly article database we employ to extract metadata about scholarly articles

- **2.8 SALTbot:** an overview of SALTbot, a summary of our previous work with it and a description of the main weaknesses we attempt to correct in this project

2.1 Previous Works

Software and article disconnection is a problem that has been described in scientific literature before. An example of this is SoftCite [4], a dataset of scientific articles from the biomedical field with software mentions.

Another attempt at making the relationships between software and articles available and retrievable is SoMesCi [5]. SoMesCi is an open data knowledge graph whose purpose is to categorize and provide insights about software annotations in scientific articles

However, these datasets only tackle the relationship in a one directional way, describing the relationship between an article and the software it mentions, but not necessarily ensuring that said software is linked to the article through some other type of citation.

2.2 Wikidata and Wikibase

Hosted by the Wikimedia foundation, Wikidata [6] is one of the largest collaborative knowledge graphs in the world. In this graph, the information is represented via semantic triples in the form of "subject predicate object" statements, and currently stores more than 1.5 billion statements[6] about multiple domains. Its main purpose is to be one of the largest databases of open-source data, as well as to serve as the base knowledge repository for other Wikimedia projects such as Wikipedia.

Wikibase [7][7] is a suite of MediaWiki extensions designed to enable collaborative editing and management of structured data. Wikibase serves as the underlying technology that powers Wikidata

[6] https://grafana.wikimedia.org/d/000000167/wikidata-datamodel?orgId=1&refresh=30m
[7] https://wikiba.se/

2.3 Wikibase Bots

Bots in Wikibase are automated software applications capable of adding, modifying and removing statements from their corresponding Knowledge Graph. In Wikidata, these bots are developed by different communities to improve the completeness, accuracy and reliability of the information in the graph.

Wikidata currently receives millions of monthly bot contributions, even surpassing author contributions during certain months[8].

Bots are diverse, ranging from those which fetch data from external sources, adapt and integrate the data to the Wikidata model, those that add language tags, or those which improve qualifier descriptions of existing QNodes.

There are more than 350 Wikidata officially approved bots[9], and some of them enrich existing software tools in Wikidata. For example, Konstin's "Github to wikidata bot"[10], enriches entities with Github links with their software release metadata and project website.

However, to the best of our knowledge there are no bots that analyze the actual contents of a code repository, such as the README and citation files, to link code repositories with bibliographical entities in Wikibase instances.

[8] https://stats.wikimedia.org/#/wikidata.org/content/edited-pages/normal|line|1-year|editor_type~group-bot*name-bot*user|monthly
[9] https://hgztools.toolforge.org/botstatistics/?lang=www&project=wikidata&dir=desc&sort=ec
[10] https://github.com/konstin/github-wikidata-bot

2.4 Software Citation Files

The scientific community has developed the Software Citation Principles [2] which led to the proposal of the Citation File Format [3] as a machine-readable metadata file for citing software projects. Since GitHub implemented support for this representation[11], an increasing number of developers have started to add these files in their repositories to obtain their corresponding credit (more than 10.000 to date).

A CITATION.cff is a YAML file that usually contains the following information:

- Title: The title of the software project.
- Authors: The names of the software authors and contributors.
- Identifiers: A collection of identifiers (e.g., Digital Object Identifier) to uniquely identifythe software project or its releases.
- License: The software's license information (e.g., MIT, GPL, Apache, etc.).
- Repository: The URL of the software's source code repository.
- Preferred citation: If the software project has already been described in a publication, this field describes the paper to be used to credit the software project's authors.

Figure 1 shows an example of a CITATION.cff file for citing the software WIDOCO[12]

```yaml
title: "WIDOCO: A Wizard for documenting ontologies"
license: Apache-2.0
authors:
  - family-names: Garijo
    given-names: Daniel
    orcid: "http://orcid.org/0000-0003-0454-7145"
cff-version: 1.2.0
message: "If you use this software, please cite both the article from preferred-citation and the software itself."
preferred-citation:
  authors:
    - family-names: Garijo
      given-names: Daniel
  title: "WIDOCO: A wizard for documenting ontologies"
  type: article
  year: 2017
  doi: 10.1007/978-3-319-68204-4_9
identifiers:
  - description: "Collection of archived snapshots for WIDOCO"
    type: doi
    value: 10.5281/zenodo.591294
```

Figure 1: example of a CITATION.cff file for WIDOCO

[11] https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-citation-files
[12] https://github.com/dgarijo/Widoco

While the adoption for CFF files is growing, a wide number of researchers still credit articles describing their software contributions with plain BibTeX[13], a common format used to reference articles in LaTeX publications (e.g., by adding their preferred citation in a README file).

2.5 SOMEF

In order to parse the repositories to get information about the citation files, we use SOMEF [8] as a means to automatically extract metadata from GitHub repositories.

SOMEF allows for the collection of scientific software metadata scattered across various files and structures within the repository. It analyzes both files dedicated exclusively to the task of citation (such as the CITATION.cff) and the repository's README.md in search of citations.

Furthermore, SOMEF provides valuable insights about the software allocated in the repository, such as the main programming language and the licenses, which can be reused to characterize software with their own metadata in knowledge graphs

2.6 CodeMeta

To better characterize software, we used Codemeta[14] as a standard categorize metadata useful for the representation of software instances in knowledge graphs.

CodeMeta is a metadata schema developed to improve the discovery, reuse, and citation of software by providing a standardized way to describe various attributes of software projects. As such, CodeMeta provides a mapping between its different schema attributes and their corresponding properties in Wikidata.

[13] https://www.bibtex.org/About/
[14] https://github.com/codemeta/codemeta/blob/master/crosswalks/Wikidata.csv

2.7 OpenAlex

To determine the existance of a specific scholarly article outside the targeted knowledge graph, we opted to query OpenAlex [9][15] since it also provided us with valuable metadata to characterize articles in knowledge graphs.

OpenAlex is a comprehensive and open-source database that provides information about scholarly works, authors, institutions, and other related entities. It serves as a free and open alternative to proprietary academic databases, offering extensive data for researchers, librarians, developers, and other stakeholders in the academic community

[15] https://openalex.org/about

2.8 SALTbot

This master's thesis follows up our previous work where we describe the lack of unity between scientific software and its associated research, the problems it entails, and we presented a framework to find these entities in collaborative knowledge graphs in order to correctly link them[10].

The result of this previous work was the first iteration of SALTbot, a software tool capable of finding articles and their software in the test zone of Wikidata, analyze their existing relationships, and complete the links between these two instances. It was also able to create entities to simulate software when no research software was found.

However, this approach entailed the following issues:

- **Working in the test area of Wikidata is insufficient:** Since this subgraph of Wikidata gets constant wipes and resets to provide constant service, the data contained in this graph doesn't resemble the actual data on Wikidata. Besides that, SALTbot's changes to the graph aren't persistent due to the wipes, so they don't count as an actual contribution to the knowledge graph

- **SALTbot hasn't been tested over real data:** The validation of the behavior of SALTbot was done by providing only 10 repositories with different casuistic regarding the existence or absence of links between article and software entities in the graph. 10 handpicked case uses don't constitute a good baseline to assess the correct behavior of the program, and therefore more data is needed to test SALTbot.

- **Software creation is incomplete:** SALTbot creates entities whose label is the same as the GitHub repository in which it is contained. However, in order to be considered a software instance in Wikidata, entities must fulfill a series of statements, such as being an instance or a software subclass (such as a python library, a programming language, or free software) or having a repository URL with a version control.

- **SALTbot is unable to work when no articles have been found:** the lack of a mechanism to validate the existence or not of a scholarly article makes SALTbot unable to decide whether a scholarly article is missing from the graph because its entity hasn't been created yet, or whether if the entity doesn't exist because the article doesn't exist altogether.

An attempt to solve some of these problems with SALTbot has been made before, and we presented a paper describing our progress and results obtained [11]. This paper was published and accepted at the ISWC 2023 Wikidata Workshop, and in this project, we will continue to address the issues and weaknesses detected in that paper, provide further validation and we will implement some of the future lines of work proposed

3 SALTbot Development

This section describes all the improvements and technical aspects we implemented during the development of this project. The structure of this section is the following:

- **3.1 Workflow:** a description of the logic and reasoning behind our proposed framework

- **3.2 Architecture:** explanation of the main modules involved in SALTbots processing

- **3.3 Integrating SALTbot with other Wikibases:** a summary of the changes needed to make SALTbot compatible with other Wikibase instances

- **3.4 Creating Software in Wikibase with SALTbot:** description of how we create instances of software and how we characterize them in Wikibase instances

- **3.5 Creating Articles in Wikibase with SALTbot:** summarization on how we create scholarly article entities and enrich them with metadata on Wikibase.

- **3.6 Assumptions:** a set of characteristics that Wikibase instances must fulfill in other for SALTbot to work properly

3.1 Workflow

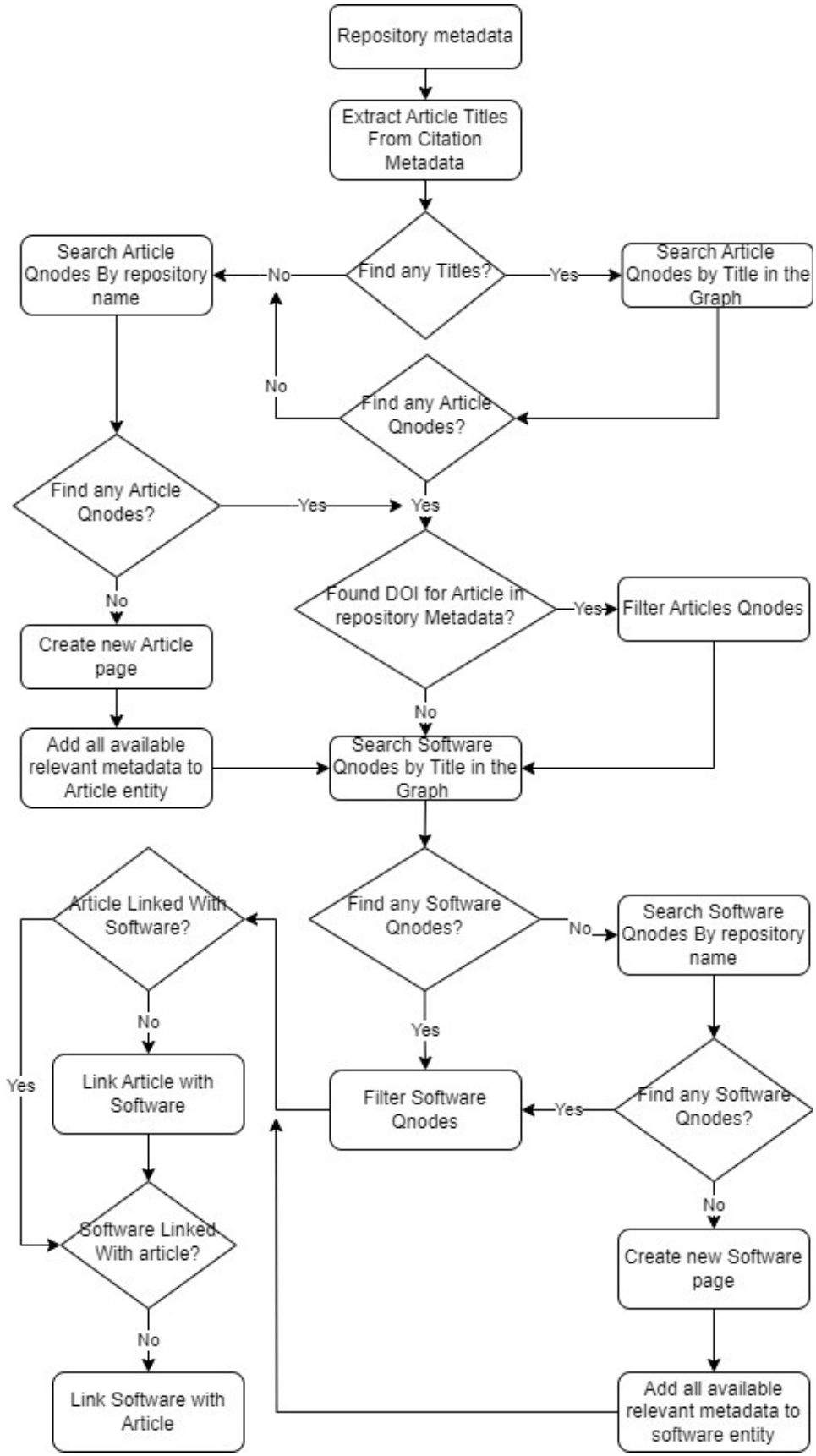

Figure 2: SALTbot's Workflow

Figure 2 shows an overview of the decision-making workflow followed by SALTbot. We start with a code repository URL. The first step is to get all the relevant metadata using SOMEF, which generates a JSON file with all the metadata in the code repository. SOMEF detects citation information with one or more preferred citations from the authors in both BiBtex and CFF (in YAML) formats, which are the ones we focus on.

SALTbot then calls the Searcher module to parse all the BiBtex and YAML citations in order to find titles for scholarly articles, as well as other information such as the Digital Object Identifier (DOI) of an article, if present. In this step, we also query OpenAlex to find more relevant information about the article

Once the candidate titles are extracted, the bot will query the target KG for entities which are instances of scholarly article and whose label is the title from the citation, filtering by the corresponding DOI. If no articles are found using the parsed citation, the Searcher module will attempt to find scholarly article entities using the GitHub repository name.

This strategy is less restrictive and consequently produces more vague results that need to be manually verified, but usually retrieves promising article candidates with a reference to the software project in their title.

The same process is also repeated to find software tools: we search for entities which inherit from the meta-class “software category” and whose label is similar to one of the parsed titles.

These entities are then filtered out by comparing their source code URL repository with the URL provided to SALTbot.

We use DOIs to filter articles. If no DOIs are found in either the parsed citation or the OpenAlex data, or if these DOIs do not match those found in the article entities, SALTbot will require manual validation from users in order to select one of the articles found to proceed with the execution.

Similarly, the repository URL allows identifying whether the software entities found correspond to the software component in the target repository. If no software candidates are found through their URL, SALTbot will ask to choose one of the found software components or to create a new one.

Next, the Analyzer module gathers all the previously existing relationships between the article and software in the graph. Using the Analyzer output, the Statement Definer will create a list with the necessary statements to completely link the article and software entities.

These statements are included in one of the following categories:

- If no software was found, SALTBot creates a new item which will be an instance of “software” and whose label will be the GitHub’s repository name. These software pages are further enriched by using the repository’s metadata such as the license, the source code repository URL, the programming languages in which the repository’s code is written and the fact that it uses Git as a version control system. Additionally, if the license detected is a open license and the “Free software” QNode was found in the graph, the new software item will be characterized as free software (i.e., “software distributed under terms that allow users to freely run, study, change and distribute it and modified versions”9).

- If no article entities were found, but an OpenAlex item regarding the scholarly article was found, SALTbot creates a new item which will be an instance of “scholarly article” and whose label will be the title found in OpenAlex. These instances will be further enriched using the metadata found in OpenAlex, adding to the items relevant information such as a DOI, its OpenAlex id, the author names… etc

- If the article is not linked to its corresponding software project, SALTBot adds a new statement to the article using the “main subject software” PNode.

- If the software project is not linked to the article, SALTBot adds a new statement using the “described by source article” PNode.

Once the number of statements in the list is higher than a batch size defined by users, all statements are loaded to the target KG using the Updater module. This process is repeated by SALTbot any number of times for each of the code repository URLs provided as input.

3.2 Architecture

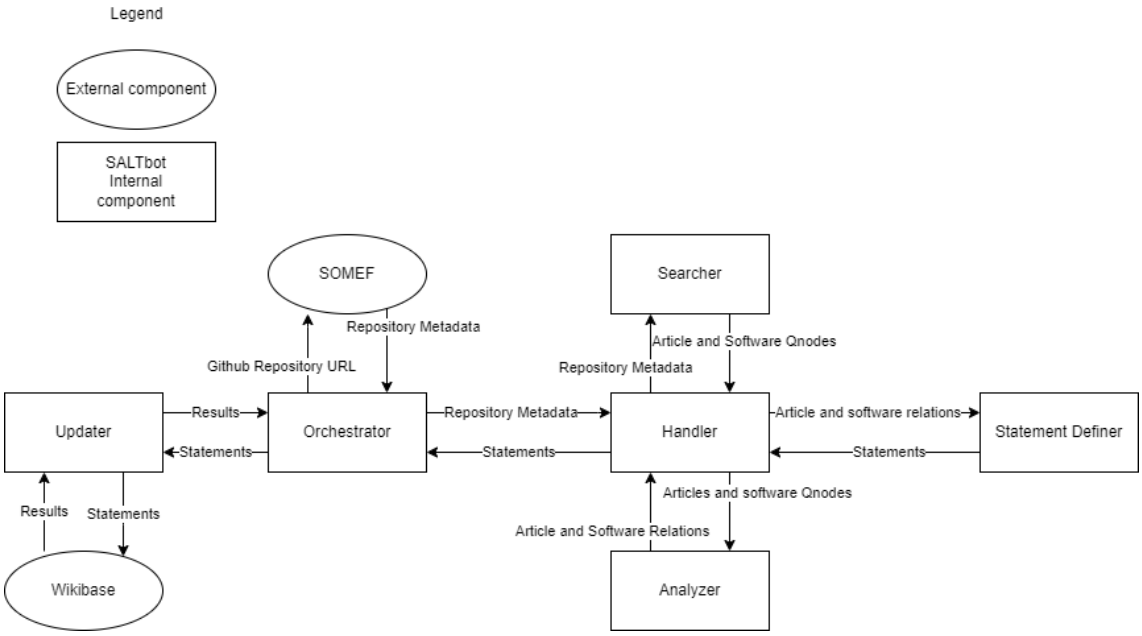

Figure 3: SALTbot's Workflow

Figure 3 shows an overview of the architecture of SALTbot. Given one or multiple GitHub repository URLs as input, SALTbot finds software and scholarly articles related to each of the repositories in a Wikibase instance, analyzes the existing relationships between these entities, and introduce new links between them to complete a bidirectional relationship between articles and software, creating and characterizing new software instances when they do not exist in the graph.

SALTbot is divided in the following modules:

- **Orchestrator:** The main module of SALTbot. It deals with the Wikibase configuration, proccesses the input, sends the parsed metadata to the handler module for each repository and calls the updater module to introduce data to the graph.

- **SOMEF:** We reuse the Software Metadata Extraction Framework, a tool that produces a JSON with relevant metadata from both the README and CITATION.cff files contained in code repositories when provided with a repository URL.

- **Handler:** This module is in charge of sending and receiving data from all of SALTbot's modules in order to figure the necessary statements to add to the graph for one repository

- **Searcher:** Finds possible article and software entity QNodes from the graph based on the metadata extracted by SOMEF.

- **Analyzer:** Assesses the existing relationships between all the articles and software found and prints them to the user

- **Statement Definer:** Creates a list of statements and entities to create in order to link an article and software, asking for user validation if needed.

- **Updater:** Uploads statements to a target Wikibase Knowledge Graph in bulk.

3.3 Integrating SALTbot with other Wikibases

Regarding the existing issues with SALTbot, the first one to be tackled was how to create a testing environment in which we could assess the correct behavior of SALTbot. Since the test area of Wikidata doesn't provide real world data about articles and their software, we need to create an environment which is able to mimic both the behavior and technology of Wikidata as well as its data.

Since the Wikidata graph is based on the Wikibase technology, we opted to create a local Wikibase and populate it with relevant entities from Wikidata related to software and articles. The gathering of this data, as well as the methods to import them to the local Wikibase will be further explained in 5.2.2

However, SALTbot targets Wikidata using a python library which provides tools to connect, query, edit and delete information on MediaWiki's collaborative graphs. Because our local Wikibase is not a graph maintained by the MediaWiki foundation, the first step in the new development of SALTbot was to refactorize it using another technology to access graph databases.

After considering other technologies and approaches we finally settled in using WikibaseIntegrator[16] due to its capacity to be easily integrated with both Wikidata and local Wikibases with minimum code changes and configuration. WikibaseIntegrator is an extension of the WikidataIntegrator[17] library, which was designed as way to query and edit Wikidata taking advantage of its SPARQL [12] [18] endpoint to ensure data consistency. The main upgrade from WikibaseIntegrator over WikidataIntegrator is that it also guarantees compatibility between different Wikibase instances.

[16] https://github.com/LeMyst/WikibaseIntegrator
[17] https://github.com/SuLab/WikidataIntegrator
[18] https://www.w3.org/TR/sparql11-query/

3.4 Creating Software in Wikibase with SALTbot

3.4.1 Software validation

After finishing the refactoring using WikibaseIntegrator, we addressed the issue of incomplete software entities. In order to tackle this issue, we must first review how SALTbot finds software entities in knowledge graphs.

Using the metadata extracted by somef, SALTbot finds software entities in the graph whose label is similar to either the repository name or the scholarly article titles found in the repository. Afterwards, the query to retrieve the entities checks that those entities are instances of a class which is itself an instance of software category

Although this approach may work for some cases, it is obvious to see that the instance checking is incomplete. ¿What if a specific software is an instance of a Python library, which is in itself an instance of a software library, which is then an instance of a software category? Using SPARQL recursion we can redefine the query to check for entities which eventually inherit from the metaclass "Software Category" in one way or another. Figure 4 shows a comparison between our old query (top), which just checks for one jump between the software instance and the software category, and the new query (bottom) which is esigned to recursively take any steps to reach the software category node.

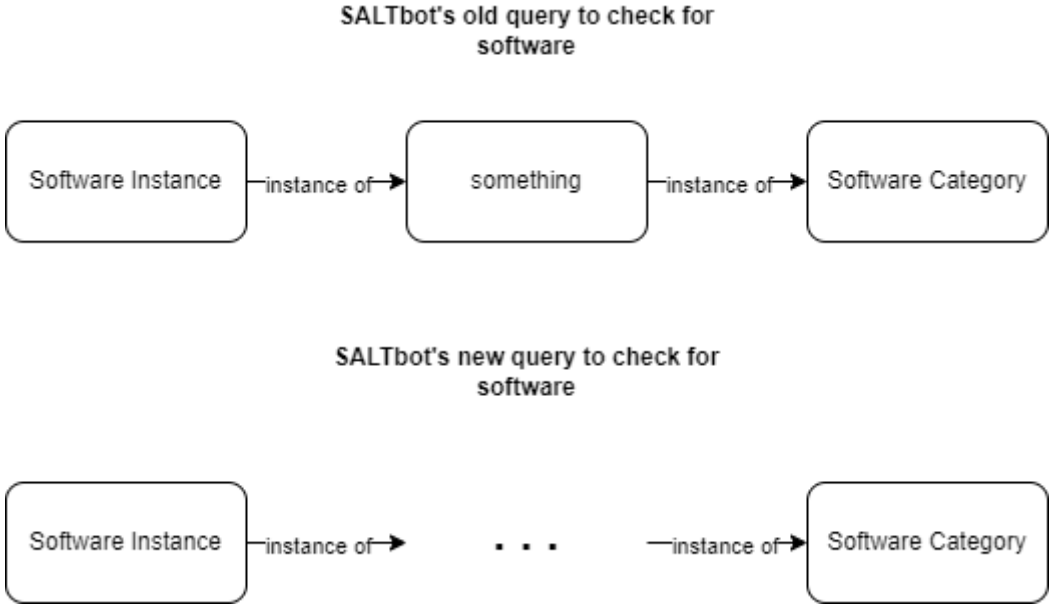

Figure 4: A comparison between SALTbot's query for fining software

Besides improving the software search, we must also find what are the necessary conditions for an entity to be considered as software in Wikibases in order to enrich the entities we intend to create. To make SALTbot fully integrated between Wikidata and Wikibase instances, we must refer to the definition of a software entity in Wikidata since they have a fixed format for software entities.

An entity is correctly characterized as a software in Wikidata if it satisfies the following:

- It is an instance of an entity which is recursively an entity of software category
- It has a repository URL with its corresponding version control system (such as a Github repository with Git as its version control system)

3.4.2 Codemeta

To further enrich our software entities with valuable metadata, we used CodeMeta to see what information is also relevant when characterizing software.

Using the mapping, we checked what data could be used from the repository metadata extracted by SOMEF to approach the CodeMeta schema when characterizing our entities. The results of this enhanced mapping can be seen in Table 1

Property	**Wikidata**	**Somef**
codeRepository	P1324	code_repository
programmingLanguage	P277	programming_languages
runtimePlatform	P400	
downloadUrl	P4945	dowload_url
fileSize	P3575	
operatingSystem	P306	
softwareRequirements	P1547	
softwareVersion	P348	
author	P50	citation
citation	P2860	
contributor	P767	

dateCreated	P571	date_created
dateModified	P5017	date_updated
datePublished	P577	
editor	P98	
fileFormat	P2701	
funder	P859	
keywords	P921	keywords
license	P275	license
producer	P162	
publisher	P123	
isPartOf	P361	
hasPart	P527	
sameAs	P2888	
url	P856	
givenName	P735	
familyName	P734	
email	P968	
IssueTracker		issue_tracker

Table 1: Mapping between CodeMeta attributes, Wikidata properties and SOMEF metadata

3.5 Creating articles in Wikibase with SALTbot

3.5.1 Article validation

Following the same reasoning behind defining what a software is in Wikidata, we must consider what is a scholarly article in Wikidata. Fortunately, the definition of a scholarly article in Wikidata is simpler than that of the software. In order to be a scholarly article in Wikidata, an entity must satisfy the following

- It's an instance of scholarly article
- It has at least one identifier that verifies its authenticity, such as a DOI

Although SOMEF is able to extract a DOI from the citation or markdown files, DOI are not always explicitly mentioned in these files. Therefore, we must refer to external data sources in order to assess the existence of the article identifiers

3.5.2 OpenAlex

To tackle the problem of univocally identifying articles using a DOI, we use OpenAlex's API to search for any relevant information about the articles.

Using the detected article titles in SOMEF's metadata, we query OpenAlex for scholarly articles whose title matches the ones found in the repository. Since OpenAlex orders the result by relevance using NLP techniques, we automatically select the first result if its relevance is significantly higher than the rest. If multiple results with a similar score are returned, we prompt the user to choose from the top 5 most relevant articles found

Besides using OpenAlex to obtain a unique identifier of an article in order to verify its existence, we also use OpenAlex as a source to obtain relevant metadata to characterize our article entities, since information such as the authors are store in OpenAlex.

3.6 Assumptions

We designed SALTbot to be compatible with any Wikibase instance holding software tools and articles. However, since every Wikibase instance may have different node identifiers, SALTbot assumes that the Wikibase modeling is similar to the Wikidata modeling in terms of the existing entities, albeit their respective identifiers (QNodes) may be different. Therefore, the first step to configure the bot is to query the graph to find the necessary QNodes and PNodes needed to operate.

The mandatory minimum items that SALTbot needs are the following:

- "instance of" property PNode: property used to check the existence of items of a specific type (both software and articles must be instances of something).

- "main subject" property PNode: property used to link an article with its specific software tool.

- "described by source" property PNode: property used to link a software with its specific article. This is the current practice by which existing articles and tools are currently linked in Wikidata, and hence we followed it.

- "Scholarly article" entity QNode: entity used to find scholarly articles in the graph. Every article must be an instance of this entity.

- "Software category" entity QNode: meta-class used to find software in the graph. Every software tool must be recursively an instance of a software category.

- "Software" entity QNode: entity used to add the mandatory "instance of something" statement to the software created by SALTbot.

If one or more of these items are missing from the target KG, SALTbot will not run

3.6.1 Software Assumptions

Additionally, SALTbot queries the graph for some optional information to better characterize the software entities. These additional elements are:

- "source code repository URL" PNode: property used to link a software entity with its code repository URL.

- "Free software" entity QNode: entity used to add the mandatory "instance of something" statement to the software node created by SALTbot (if the software tool has a free license in the GitHub repository). If a software project does not have a free license, we categorize it as "Software".

- "programmed in" PNode: property used to define the programming language in which a software entity is developed.

- "download link" PNode: property used to link a software entity with its specific article.

- "copyright license" Pnode: property used to specify the type of software license used by a software entity.

- "version control system" and "web interface software" PNodes: properties used as qualifiers when describing the source code repository of a software project.

- "Git" and "GitHub" QNodes: entities used with the two previous properties to add qualifiers to assign a source code repository URL to a software entity.

3.6.2 Article Assumptions

Following the same reasoning as for Software entities, SALTbot queries the graph for the following information regarding articles in order to add relevant metadata to the article entities

- "DOI" PNode: property used to add the DOI to a newly created article entity

- "OpenAlex ID" PNode: property used to add the ID found in openAlex to a newly created article entity

- "author name string": property used to a the author names found in OpenAlex metadata

4 Using SALTbot

This section elaborates on the processes to configure and use SALTbot. The structure is as follows:

- **4.1 Installation and Distribution:** required processes to download and configure the necessary packages to use SALTbot

- **4.2 How to use SALTbot to visibilize your work:** an explanation on how to use SALTbot and a use case

4.1 Installation and Distribution

One of the biggest inconveniences of SALTbot was its installation process. The previous installment of the bot required to install and configure multiple modules separately using a series of scripts and user input. With the migration to WikibaseIntegrator, this process is no longer needed, and the configuration process has been greatly simplified since.

However, SALTbot still lacks the necessary wrappers to configure somef, therefore, it must still be installed and configured separately. To install somef we execute in the command line:

```
>pip install somef
```

Afterwards, we can configure somef executing:

```
>somef configure
```

We strongly recommend that you configure somef with your own Github personal access token in order to enjoy SALTbot's multi-repository execution methos without hitting Github's API rate limit.

During the refactoring phase, all the code was reformatted and reordered following the necessary structure to create a pip package. SALTbot is now distriubted as a pip pckage and can easily be installed and configure in less steps

To install SALTbot, we can execute:

```
>pip install SALTbot.1.0.0
```

Once all the necessary software has been installed and configured, we need to run SALTbot's configuration using the configure command.

>python SALTbot.py configure

Figure 5 shows an example of running this command using the appropriate URLs to target our local Wikibase

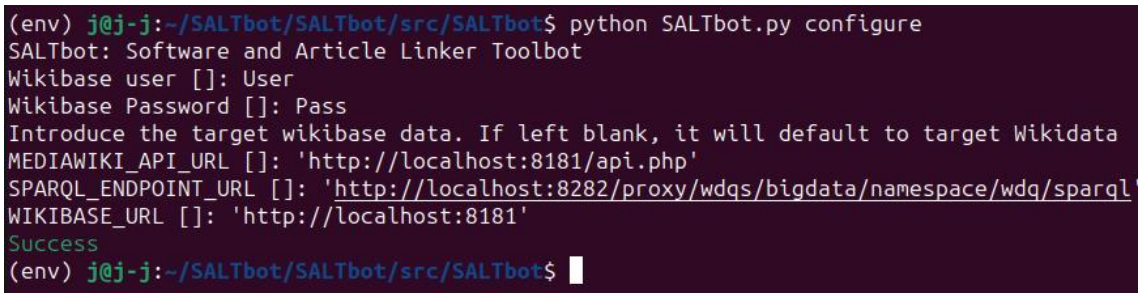

Figure 5: An example of running SALTbot configure

The information required to configure SALTbot is the following:

- **Wikibase user:** Since SALTbot uses OAuth1 to impersonate a user with whom it will create the new statements and entities, we must supply it with a valid username with editing permissions already created in the graph

- **Wikibase Password:** The password for the username provided before

- **MEDIAWIKI_API_URL:** the URL of the mediawiki api documentation of our target Wikibase. If left blank, this parameter will default to Wikidata's Mediawiki API URL

- **SPARQL_ENDPOINT_URL:** the URL of the SPARQL endpoint of the target graph. If left blank, this parameter defaults to Wikidata's SPARQL endpoint

- **WIKIBASE_URL:** The URL of our target Wikibase. It defaults to Wikidata's URL if left blank

4.2 How to use SALTbot to increase the visibility of your work

If we run

>SALTbot.py –help

We can see the usage and all the commands from SALTbot (Figure 6)

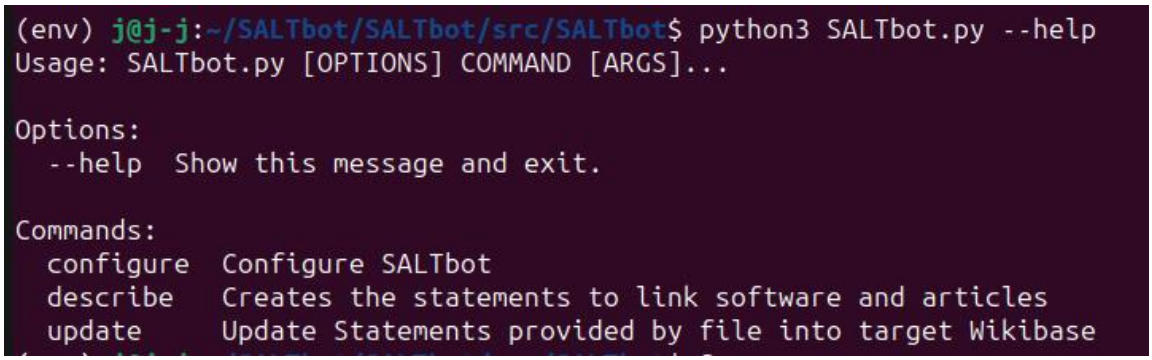

Figure 6: SALTbot usage and commands

Besides the already mentioned configure command, SALTbot uses 2 other commands:

- **SALTbot describe:** This is the main bulk of SALTbot's functionality. This command will process the entry provide to the bot, which can be either one or multiple repository URLs or one or multiple JSONs already produced with SOMEF and query the graph to find all the relevant information about the relationships between these entities, prompting the user for information when needed. After it finishes its execution, it will dump a file called "operation_list.txt" with all the necessary operations to be performed in the graph in order to successfully link the articles and software for the provided repositories. Figure 7 shows an example of said operation list.

- **SALTbot update:** given an operation_list.txt file, this command edits the target graph to create all the entities and introduce all the relevant statements contained in the file

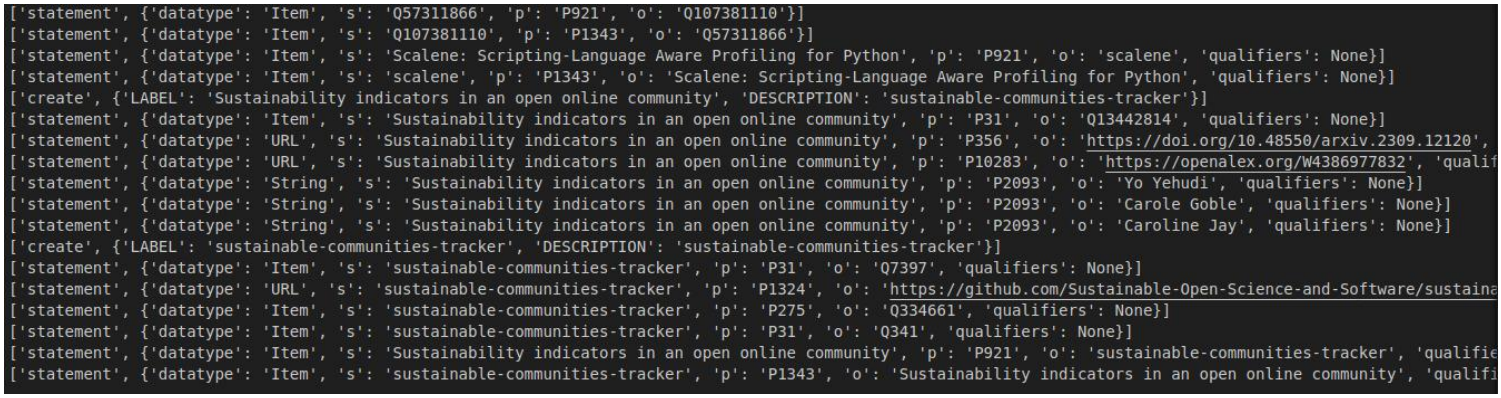

Figure 7: an example of an operation list created by SALTbot

4.2.1 Use case

We will now provide a use case to illustrate how a user would employ SALTbot to link a software and its article using one repository as its entry.

Let's assume we have a nearly empty Wikibase in which the SALTbot assumptions about the structure of the data stated in 3.4, as well as some of the assumptions for software and articles are satisfied (refer to figure 8 to see what assumptions are satisfied and what nodes correspond to the entities and properties). We use this case because it allows us to better illustrate the capabilities of SALTbot. Since the Wikibase is emptied besides the initial assumptions, both the software and article need to be created, enriched with metadata, and properly be linked to one another.

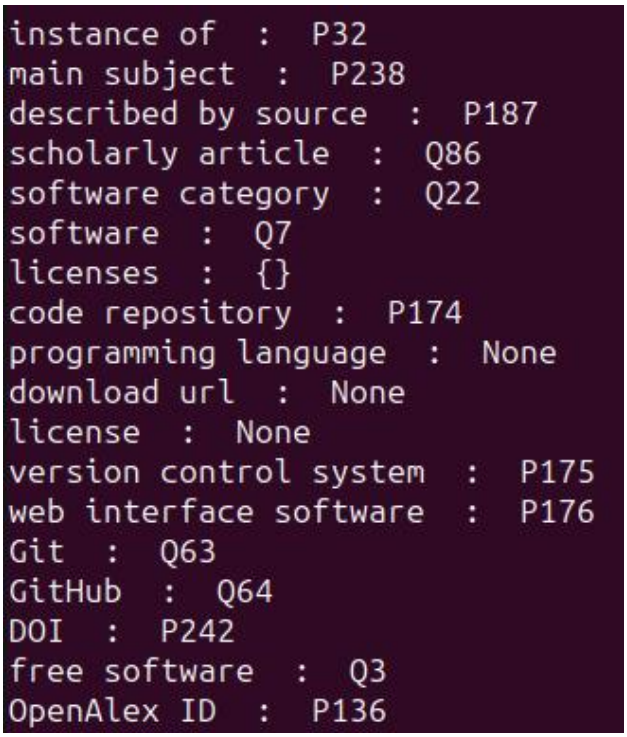

Figure 8: Local Wikibase Pnodes and Qnodes corresponding to SALTbot's assumptions

A user would execute SALTbot as seen in Figure 9 to create the operation list necessary for linking. We can also see that, during its search phase, SALTbot was able to extract an article title, but unsurprisingly, no entities were found in the Wikibase

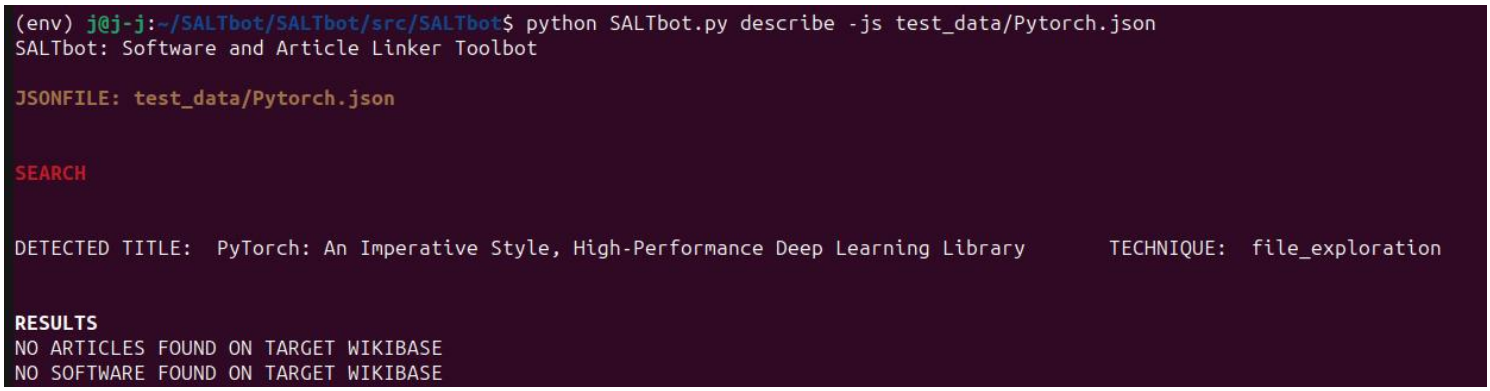

Figure 9: SALTbot's execution and search phase

During the Linking phase, SALTbot will detect the necessary statements to completely link the software and the article. Figure 10 provides the operations detected for our use case.

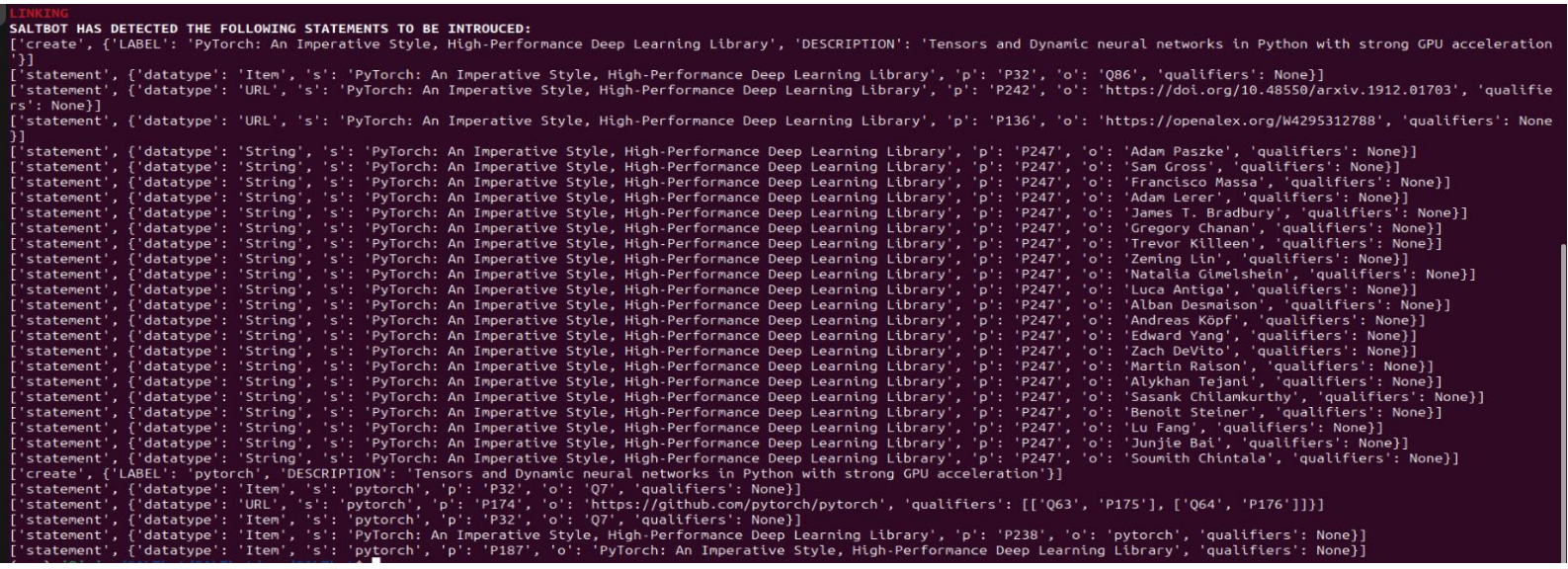

Figure 10: operations detected by SALTbot

As we can see on Figure 10, the first operation will be to create an entity whose label will be the detected title found in the search phase, and whose description will be extracted from somefs metadata. Afterwards, the bot introduces the statement "Pytorch:An Imperative… P32 Q86", If we refer back to Figure 8, we can infer that this statement introduces the "instance_of scholarly article" statement. The two following statements introduce some characterization data about the article, in this case a DOI and an OpenAlex ID found when querying OpenAlex.

After creating the article, SALTbot will follow a similar approach to create the software using “pytorch” (which is the repository name” as label, and the escription found in the repository metadata. Then, SALTbot will add an URL statement to represent the repository URL of the software, qualifie with its version control system and web interface software, in this case Git and GitHub respectively. Afterwards it adds the statement “instance_of software”

The last two statements refer to the linking of the two newly create entities. First we have the “Pytorch:An imperative… main_subject pytorch” statement to link the article with the software, an then the “pytorch described_by_source Pytorch:An imperative…” statement to link the software with the article.

After the operations are detected, they will be dumped into “operation_list.txt” file. Figure 11 illustrates the contents of this file

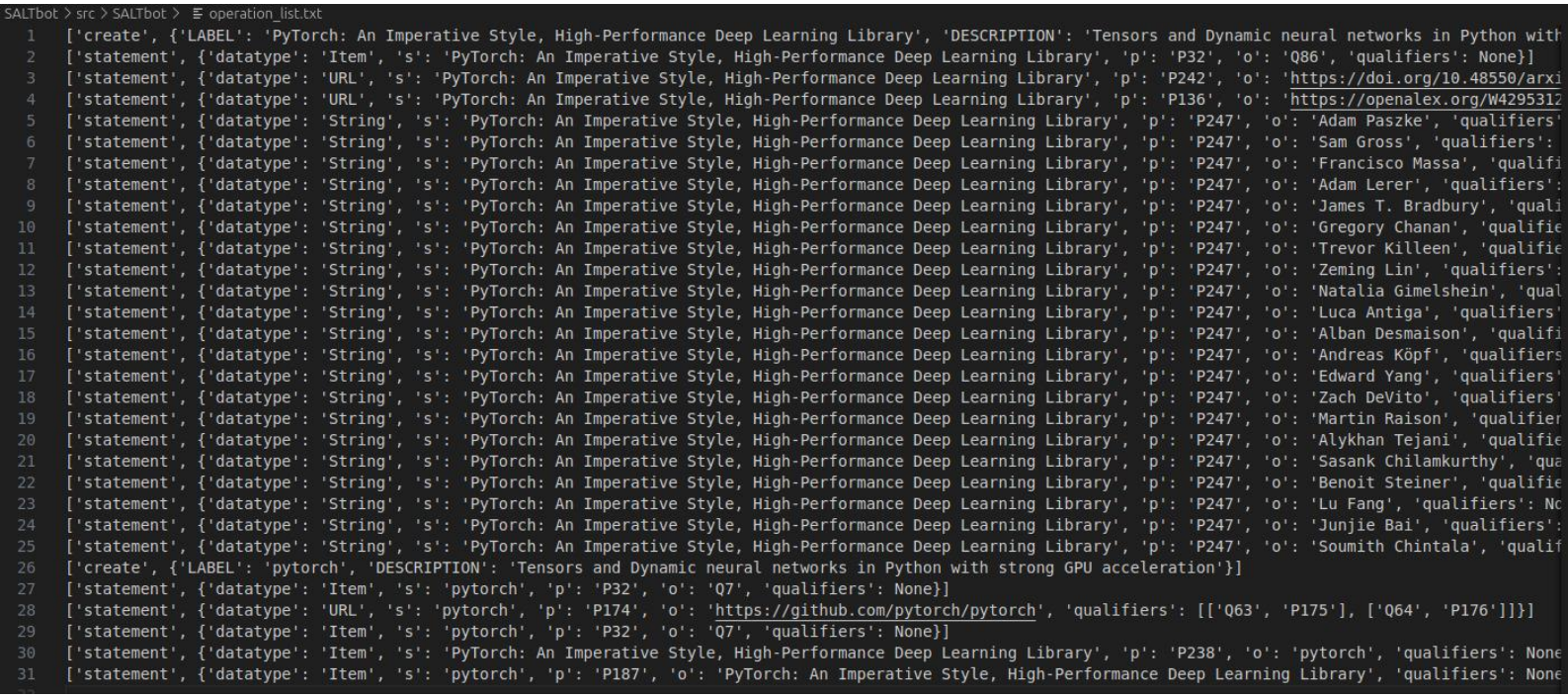

Figure 11: contents of the operation_list.txt file for our use case

Once the operation list has been created, we have to execute the SALTbot update command to upload the changes into the wikibase. This process can be observed in Figure 12.

```
(env) j@j-j:~/SALTbot/SALTbot/src/SALTbot$ python SALTbot.py update -f operation_list.txt
SALTbot: Software and Article Linker Toolbot
SALTbot WILL INTRODUCE THESE STATEMENTS IN WIKIDATA
CREATE ENTITY [ PyTorch: An Imperative Style, High-Performance Deep Learning Library ] WITH DESCRIPTION [ Tensors and Dynamic neural networks in Python with strong GPU acceleration ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P32   Q86 ] OF TYPE [ Item ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P242   https://doi.org/10.48550/arxiv.1912.01703 ] OF TYPE [ URL ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P136   https://openalex.org/W4295312788 ] OF TYPE [ URL ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Adam Paszke ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Sam Gross ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Francisco Massa ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Adam Lerer ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   James T. Bradbury ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Gregory Chanan ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Trevor Killeen ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Zeming Lin ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Natalia Gimelshein ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Luca Antiga ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Alban Desmaison ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Andreas Köpf ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Edward Yang ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Zach DeVito ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Martin Raison ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Alykhan Tejani ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Sasank Chilamkurthy ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Benoit Steiner ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Lu Fang ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Junjie Bai ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P247   Soumith Chintala ] OF TYPE [ String ] WITH QUALIFIERS [ None ]
CREATE ENTITY [ pytorch ] WITH DESCRIPTION [ Tensors and Dynamic neural networks in Python with strong GPU acceleration ]
CREATE STATEMENT [ pytorch   P32   Q7 ] OF TYPE [ Item ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ pytorch   P174   https://github.com/pytorch/pytorch ] OF TYPE [ URL ] WITH QUALIFIERS [ [['Q63', 'P175'], ['Q64', 'P176']] ]
CREATE STATEMENT [ pytorch   P32   Q7 ] OF TYPE [ Item ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ PyTorch: An Imperative Style, High-Performance Deep Learning Library   P238   pytorch ] OF TYPE [ Item ] WITH QUALIFIERS [ None ]
CREATE STATEMENT [ pytorch   P187   PyTorch: An Imperative Style, High-Performance Deep Learning Library ] OF TYPE [ Item ] WITH QUALIFIERS [ None ]
CONFIRM (Y/N):
```

Figure 12: SALTbot update execution

As seen in Figure 12, SALTbot will print a summary of the operations to be performed. Since the auto flag was not provided, it will also ask the user for a confirmation before uploading the changes. After confirming the changes, SALTbot will update the list, as can be seen in Figure 13

```
CONFIRM (Y/N): Y
Creating entity  PyTorch: An Imperative Style, High-Performance Deep Learning Library
Item created as  Q626
creating statement [ Q626   P32   Q86 ]
succesfully created [ Q626   P32   Q86 ]
creating statement [ Q626   P242   https://doi.org/10.48550/arxiv.1912.01703 ]
succesfully created [ Q626   P242   https://doi.org/10.48550/arxiv.1912.01703 ]
creating statement [ Q626   P136   https://openalex.org/W4295312788 ]
succesfully created [ Q626   P136   https://openalex.org/W4295312788 ]
creating statement [ Q626   P247   Adam Paszke ]
succesfully created [ Q626   P247   Adam Paszke ]
creating statement [ Q626   P247   Sam Gross ]
succesfully created [ Q626   P247   Sam Gross ]
creating statement [ Q626   P247   Francisco Massa ]
succesfully created [ Q626   P247   Francisco Massa ]
```

Figure 13: trace of the process of uploading statement

Once the upload has finished, the changes can be seen in the recent changes using the wikibase user interface. Notice in Figure 14 how SALTbot created both entities (denoted by the letter N to the left of each entry in the Recent Changes) but also introduced 4 changes to the software entity and 26 to the article entity, which correspond to the 30 statements described by SALTbot in the previous steps.

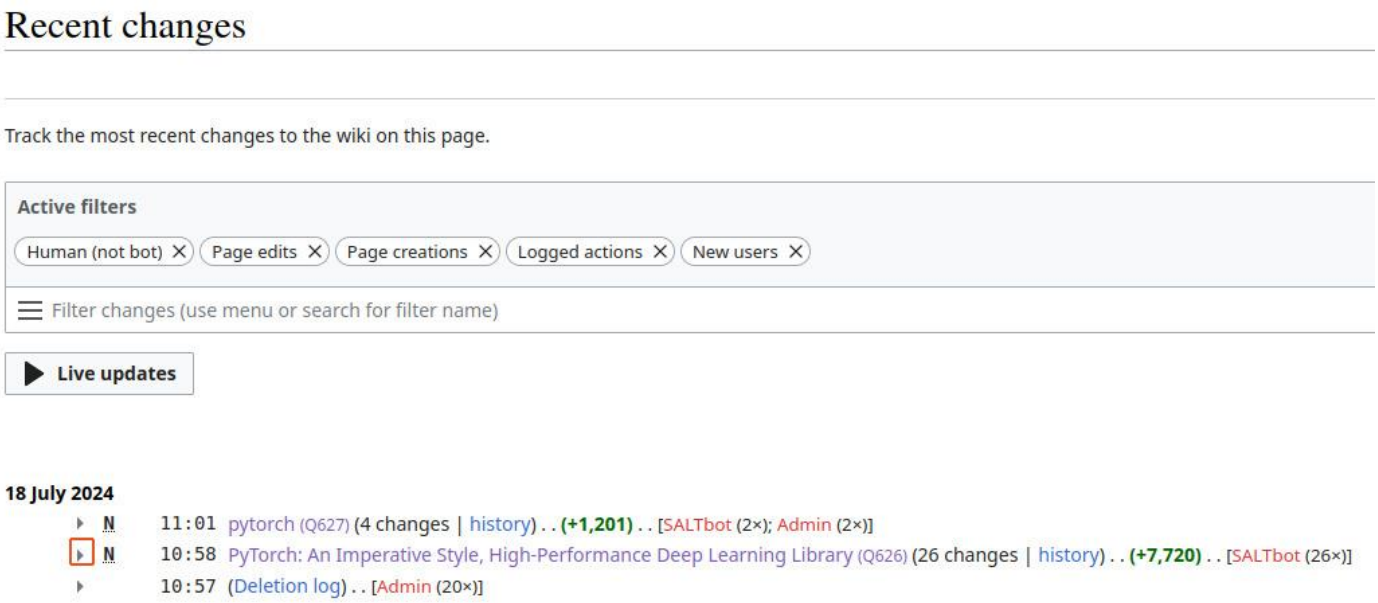

Figure 14: recent changes made to the graph

If we navigate using the user interface to the entities created, we can expect to see something like that of Figure 15

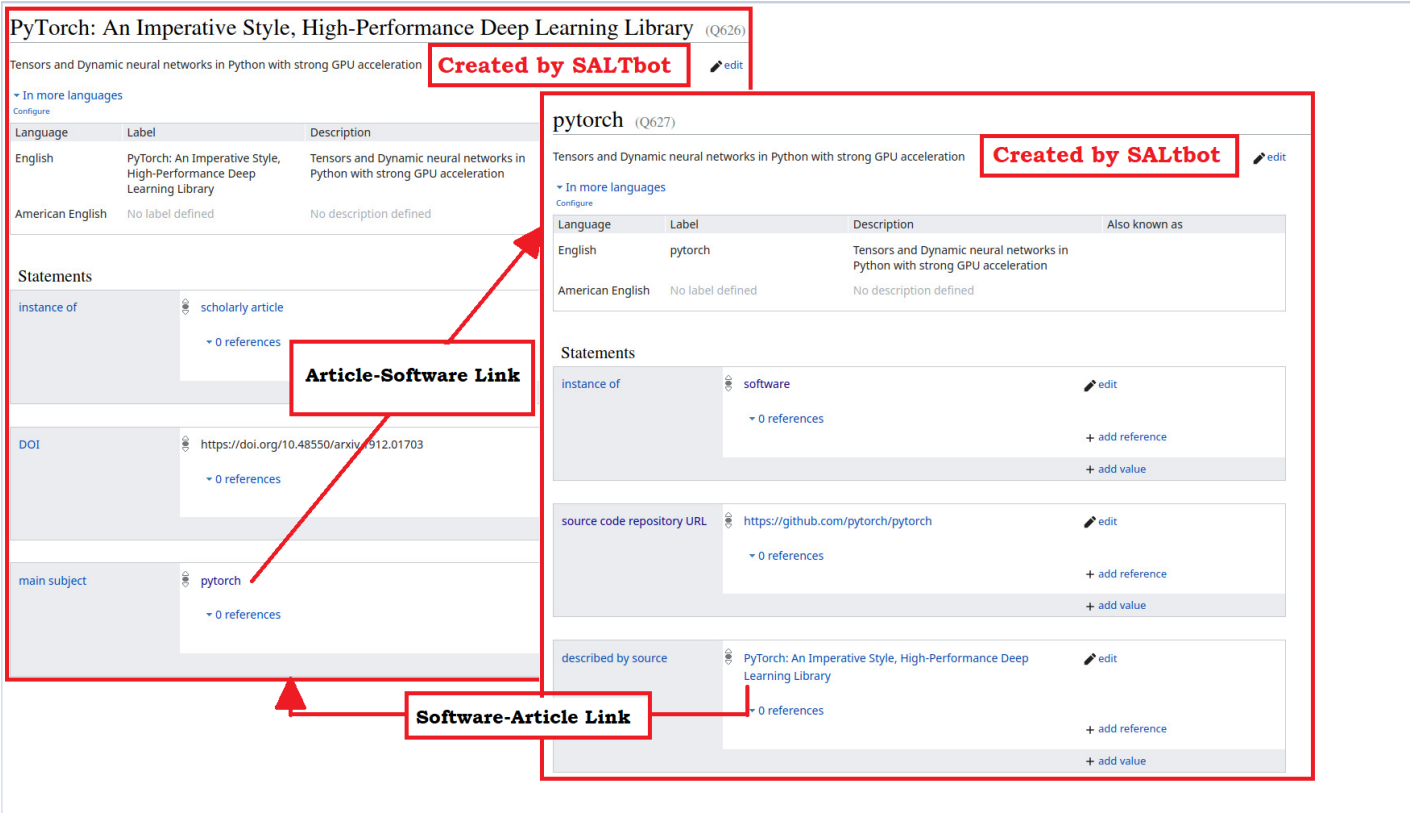

Figure 15: edits made by SALTbot in the Wikibase

5 Validation

This section explains the method use to assess the correct behaviour of SALTbot in both local Wikibases instances and Wikidata. It also summarizes the contributions made to Wikidata using SALTbot during the development of this project. The structure is the following:

- **5.1 Reproducing the environment of Wikidata:** a description of the environment we created and populated to test the behavior of SALTbot before introducing any data to Wikidata

- **5.2 Linking entities in Wikidata:** an explanation of the contributions done with SALTbot to Wikidata's knowledge graph

5.1 Reproducing the environment of Wikidata

Since we must avoid introducing erroneous data into Wikidata, we need to develop a test environment that is able to reproduce both Wikidata's behaviour and its data. This section describes how we created such environment and populated it with the same data that is currently stored in Wikidata

5.1.1 Docker Wikibase

To reproduce the environment of Wikidata, we deployed a local Wikibase using the docker images provided by MediaWiki[19]

This images allows to create an architecture (Figure 16) comprised of the following elements:

- A wikibase and its user interface
- A SPARQL endpoint to query the graph
- Additional services such as Quickstatements and ElasticSearch

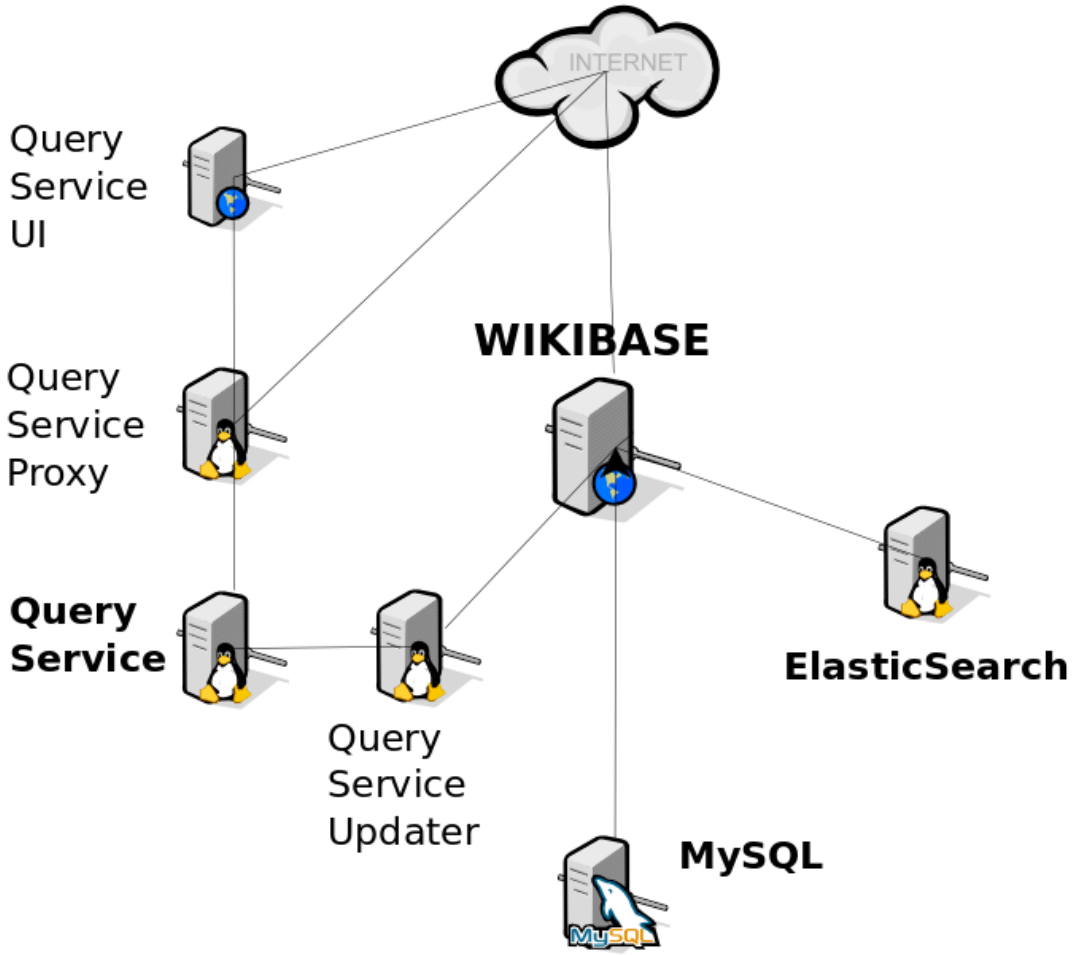

Figure XXX: architecture provided by MediaWiki docker images

[19] https://www.mediawiki.org/wiki/Wikibase/Docker/en

5.1.2 Importers

Once we deployed the local architecture to have our own Wikibase instance, we must address the problem of populating this instance with entities of software and articles, as well as to introduce the necessary assumptions into the graph

Different approaches were considered and discarded satisfy this requirement, such as

- WikibaseImport[20]: the php script contained by default in the Wikibase docker container. This script allows to import data from Wikidata into the Wikibase by connecting to the container and executing a series of commands. However, this script was deprecated a couple of years ago and it no longer works for newer versions of Wikibase. An attempt to update this script to work was made, but since it was a time consuming task in itself and we considered this project to be out of the scope of SALTbot's development, we moved on to other ways to import data

- WikibaseSync[21]: a python script to automatically import entities and properties from Wikidata into other Wikibase instances. When using this tool, we found out a bug which wouldn't allow us to configure the script to target our Wikibase. An attempt to contact the authors of the script was made but we obtained no answers and abandon this approach.

We finally settled on programming our own importer to recreate the data from Wikidata. Using WikibaseIntegrator, we created a python script which queries Wikidata for entities and properties, creates them in another Wikibase and introduces one by one all of its statements to mimic the behaviour of Wikidata.

The importer is stored in https://github.com/JB0linches/WikibaseImporter and its free of use and distribute.

[20] https://github.com/Wikidata/WikibaseImport
[21] https://github.com/the-qa-company/WikibaseSync

5.2 Linking entities in Wikidata

This section describes the processes involved in assessing the behaviour of SALTbot when uploading content to Wikidata, or what we call, validation.

5.2.1 Paper validation

A first approach of SALTbot's usage was presented and accepted at ISWC 2023 Wikidata Workshop[]. This bot however was capable of creating software entities, and could not operate when no articles where found in the graph In order to assess the correct behaviour of SALTbot, we tested the bot by gathering 500 repositories from GitHub with a "CITATION.cff" file using the GitHub API and validating the results manually. The rationale behind our approach is to ensure the selection of code repositories with at least a suggested pointer to a publication.

Our selected 500 repositories presented the following characteristics before our bot assessment was completed:

- 378 repositories had one or more mentions to scholarly articles (some refer to code deposits in archives like Zenodo[22]).
- 46 repositories had their corresponding scholarly article page in Wikidata.
- 35 repositories had their corresponding software page in Wikidata.
- 12 scholarly article entities were previously linked through the property "main subject" to their corresponding software entity.
- 5 software entities were previously linked through property "described by source" to their corresponding article entity.

In order to perform the validation of SALTbot, we created a bot page and a new username in Wikidata to keep a record of the contributions performed with the tool. These contributions can be seen in https://www.wikidata.org/wiki/Special:Contributions/SALTbotDev.

[22] https://about.zenodo.org/

After our manual validation, SALTbot enriched Wikidata with the following knowledge:

- 33 newly created software entities.
- 104 new software metadata statements.
- 34 scholarly articles linked with their corresponding software entity (this number includes articles whose software has been created in order to link them).
- 43 software entities linked with their corresponding scholarly articles (this number includes those software QNodes newly created by SALTbot in order to link them).

While validating SALTbot, we noticed how our approach blends in with the Wikidata ecosystem. Shortly after creating new software entities, other bots like Github-wiki-bot started improving existing page descriptions with their release contents (184 statements regarding software version identifiers and official pages were added to our newly created software entities).

5.2.2 Gathering Github repositories

To fully validate the correct behaviour of SALTbot, especially after adding the ability to create article entities, we needed to gather more repositories with CITATION.cff.

If we use the search option in Github, we can find that there currently exist more than ten thousand repositories with a file named “CITATION.cff”. Using GitHubs API[23], we were able to produce a script which was able to get all the repository URLs from github and check for the existence of a citation file in its contents. However, since the script needs at least one request per repository to get the contents, and there is no way to circumvent this, we were hitting the API rate limit very soon.

Our intention with this script was to scout all the repositories from github and get all the URLs from repositories with citation files. However, since Githubs API only allowed us to make 100000 requests per day, even when logged with an authorized user, we quickly discarded this approach due to the fact that there are more than 100 million repositories in GitHub[], making it impossible to complete this task in a reasonable time.

[23] https://docs.github.com/en/rest?apiVersion=2022-11-28

We limited our search scope to a known dataset which contains software related to its paper. The Bidirectional dataset is the result of an evaluation to find a bidirectional relationship between scientific papers and their software repository []. This dataset contains 3000 repository URLs which cite a paper, and provides arxiv ids and dois for scholarly articles which cite the software in the repository.

For preprocessing the dataset, we first selected only those urls which belong to github repositories, since the Bidirectional dataset also contains repositories from zenodo or gitlab. After selecting the 1205 urls from GitHub, executing our script to check for citations provided us with 97 repositories with a CITATION.cff file.

5.2.3 Validation in Wikidata

The 97 repositories gathered from the Bidirectional dataset presented the following characteristics before SALTbot's execution:

- 16 repositories had a previously created entity in Wikidata for their scholarly article mentioned in the CITATION.cff
- 74 repositories had an easily retrievable relationship between their scholarly article in the CITATION.cff and their corresponding article node in OpenAlex
- 14 repositories had a previously created entity in Wikidata
- 7 articles were already linked with their software in Wikidata
- 7 softwares were already linked with their software in Wikidata

Coincidentally, the 7 softwares and articles bidirectionally related in Wikidata that also were part of the bidirectional dataset were links introduced by SALTbot in a previous validation step.

After our execution, our novel contributions were:

- 40 newly created scholarly article entities
- 62 newly created software entities
- 48 article-software links created
- 49 software-article links created
- 251 article metadata statements created
- 230 software metadata statements created

These new contributions can be seen at
https://www.wikidata.org/wiki/Special:Contributions/SALTbot

6 Results and conclusions

Even though our second corpus of repositories was smaller than the one used on the first validation described in 5.2.1, our results executing over the bidirectional dataset are more promising. Table XXX shows a comparison between our first and second validation, expressing the results as a percentage over the total dataset repositories in which they were obtained.

	First validation results	**Bidirectional Dataset Validation results**
Size of dataset	500	97
Scholarly article entities created (%)	0 (0%)	40 (41.2%)
Software entities created (%)	33 (6.6%)	62 (63.9%)
Article-software links created (%)	34 (6.8%)	48 (49.5%)
Software-article links created (%)	43 (8.6%)	49 (50.5%)
Article metadata statements created	0	251
Software metadata statements created	104	230

Table XXX: a comparison between the results obtained in our first and second validations

As seen in table XXX, we obtained far better results over the bidirectional dataset. This is due to mainly two factors

- The bidirectional dataset is better curated than our own dataset, since it is gathered by experts with domain knowledge experts on the subject. Furthermore, the subset of the bidirectional dataset selected tends better to the strengths of SALTbot, since it attends better to the strengths of SALTbot, such as previous links between the data

- As denoted in the conclusions of SALTbot: Linking Articles and Software in Wikidata, we expected that creating scholarly article entities when none where found in the graph would significantly improve our results. Due to the fact that SALTbot is now able to create scholarly article entities, we are more likely to introduce more links between existing or created software and their missing articles

7 Conclusions

In this paper we introduced SALTbot, our effort towards enriching Wikibase/Wikidata with the software implementations of existing research articles. We have validated our approach with the processing of over 600 repositories from different sources, resulting in 40 new article entries, 95 new software entities, over 80 new links between articles and software and more than 500 statements of metadata for various entities in Wikidata.

In 2018, GitHub reached the staggering milestone of holding more than a hundred million code repositories. In comparison, ten thousand repositories with a CITATION.cff seems to be a small percentage. This is also pointed out by the fact that out of the 3000 repositories in the Bidirectional dataset, only 97 had a CITATION.cff file. These statistics also suggest that many research software projects in GitHub may be lacking a citation file to indicate the correct way of citing the software in a machine-readable way.

Our approach is orthogonal to the efforts of other platforms like Papers With Code[24] or Arxiv, which scan data/software availability statements or whole publications (manually or automatically) to find the corresponding associated code repositories. Instead, we analyze code repositories assessing the direct citation preference declared by authors.

SALTbot contributions are integrated within the Wikidata ecosystem, with other bots building and expanding on our work. We believe that continuously running SALTBot will increasingly enrich Wikidata with links between articles and software and, as developers continue adopting best software citation practices, SALTbot will become increasingly useful to the Wikidata and scientific communities.

[24] https://paperswithcode.com/

8 Bibliography

[1] N. P. Chue Hong, D. S. Katz, M. Barker, A.-L. Lamprecht, C. Martinez, F. E. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez. *FAIR Principles for Research Software (FAIR4RS Principles),* 2022.

doi: 10.15497/RDA00068

[2] A. M. Smith, D. S. Katz, K. E. Niemeyer, *Software citation principles*, PeerJ Computer Science 2 (2016) e86.

doi: 10.7287/PEERJ.PREPRINTS.2169V2

[3] S. Druskat, J. H. Spaaks, N. Chue Hong, R. Haines, J. Baker, S. Bliven, E. Willighagen, D. Pérez-Suárez, O. Konovalov, *Citation File Format*, 2021.

doi:10.5281/zenodo.5171937

[4] Du, C., Cohoon, J., Lopez, P., & Howison, J. *Softcite Dataset: A Dataset of Software Mentions in Biomedical and Economic Research Publications*. Journal of the Association for Information Science and Technology.

doi: 10.1002/asi.24454

[5] D. Schindler, F. Bensmann, S. Dietze, and F. Krüger, *SoMeSci-A 5 Star Open Data Gold Standard Knowledge Graph of Software Mentions in Scientific Articles*. CIKM '21: Proceedings of the 30th ACM International Conference on Information & Knowledge Management

doi: 10.1145/3459637.3482017

[6] D. Vrandečić, M. Krötzsch, *Wikidata: a free collaborative knowledgebase*, Communications of the ACM 57 (2014) 78–85

doi: 10.1145/2629489

[7] D. Diefenbach, M. D. Wilde, S. Alipio, *Wikibase as an infrastructure for knowledge graphs: The eu knowledge graph*, in: A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, H. Alani (Eds.), The Semantic Web – ISWC 2021, Springer International Publishing, Cham, 2021, pp. 631–647.

doi: 10.1007/978-3-030-88361-4_37

[8] A. Mao, D. Garijo, S. Fakhraei, *Somef: A framework for capturing scientific software metadata from its documentation*, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3032–3037.

doi: 10.1109/BigData47090.2019.9006447.

[9] J. Priem, H. Piwowar, R. Orr, *Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts*, 2022.

arXiv: 2205.01833.

[10] J. Bolinches, (2022). *SALTbot: creación de un bot para enlazar artículos y software en WikiData*. Proyecto Fin de Carrera / Trabajo Fin de Grado, E.T.S de Ingenieros Informáticos.

Available at: https://oa.upm.es/71007/

[11] J. Bolinches and D. Garijo, *SALTBot: Linking Software and Articles in Wikidata*, Wikidata 23: Wikidata workshop at ISWC 2023

doi: 10.5281/zenodo.8190001.

[12] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation (2008).

Retrieved from: http://www.w3.org/TR/rdf-sparql-query/

[13] Garijo, D., Arroyo, M., Gonzalez, E., Treude, C., & Tarocco, N. (2023). *Bidirectional dataset*.

Available at https://doi.org/10.5281/zenodo.10307603