Intelligent File System

Benjamin Hardin
(0030707528)
Purdue University
West Lafayette, Indiana, USA
hardin30@purdue.edu

Tejaswini Satish Kumar
(0033676893)
Purdue University
West Lafayette, Indiana, USA
tsatishk@purdue.edu

Mohamed Yilmaz Ibrahim
(0033795176)
Purdue University
West Lafayette, Indiana, USA
ibrahi35@purdue.edu

ABSTRACT

File searching can be an arduous task when the user does not remember the exact file details. With the current techniques, the user will have to search the file according to different parameters such as file type, name, date, or other metadata. This metadata can be difficult to remember and can require effort to share with the search engine. Often, metadata has to be filled into respective boxes rather than being parsed directly from the search request. We propose a tool that simplifies this task by allowing users to retrieve files with simple natural language queries.

The tool parses the query looking for phrases that relate to metadata so the user does not have to input them into specific fields. Additionally, the tool understands the context of words in order to convert words like "today" or "tomorrow" into the specific date that user means. The tool is accessible from Finder for quick access while already browsing for files. The tool presents all options that match the user's query like a common file search application, along with files that our algorithm consider to be potential matches. Our study involves a literature review to formulate the design requirements, an iterative design process followed by a user study. To evaluate the tool, we plan to conduct a user survey with the class.

CCS CONCEPTS

• **Human-centered computing** → **User interface management systems**; **User studies**; **Usability testing**; **Human computer interaction (HCI)**; • **Computing methodologies** → **Information extraction**; • **Natural Language Processing** → **Artificial intelligence**.

KEYWORDS

Natural Language Processing, File Search, Indexing, Intelligent Filesystem

ACM Reference Format:
Benjamin Hardin, (0030707528), Tejaswini Satish Kumar, (0033676893), and Mohamed Yilmaz Ibrahim, (0033795176). 2018. Intelligent File System. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Searching for files can be a time consuming and cumbersome process. Often, files are not properly named or the folder structure is completely disorganized. Even if file organization is ideal, it can be difficult to click through levels upon levels of folders or type in the folder path to get to the file. To reduce the clicking around in the file viewer, most operating systems have some form of file search that allows the user to search by file name or type. However, these file search methods require the user to add tags for date, file type, or other metadata to their search query in order to find files that match these criteria.

Our project aims to address this issue by applying natural language processing (NLP) to the user's file search to understand what file metadata pertains to the user's request. Natural language processing has shown promising results in translating how humans speak into keywords that computers can understand. Such systems are common for digital assistants like Alexa, Siri, and Google Assistant that allow the users to make common requests in the same manner they would ask a friend, such as, "Where did I park my car" or, "Is it hot outside" [9]. The use of NLP has been investigated for applications such as spelling auto-correct. In the context of writing, sentence context can often determine if a word is misspelled such as the difference between "to" and "too." Being able to perceive the natural language of the sentence can make this distinction and flag an error the user [12]. Additionally, Google's Neural Machine Translation (GNMT) helps Google translate sentences between languages in a way that is closer to how humans speak than just converting a sentence word-by-word [17].

We propose a tool to allow the user to search for files as if they were asking a secretary for a file. They can ask "can you show me presentations from a year ago" or "find files downloaded from brightspace it might be in a folder named fall21 or verification." This approach removes the need for an individual to remember specific details or the name of a file that can be difficult to remember and are required for many existing search methods. The tool is designed be invoked from an icon on the toolbar in Finder for quick access when looking for files, although this functionality was not implemented in our prototype. We will also consider easier methods for calling the file icon such as a digital assistant shortcut, menu bar icon, or keyboard shortcut. The file results are be presented to the user in a popup that highlights the top result with other possible results alongside it. It handles up to 8 files in case the search matches multiple files or the algorithm's primary prediction is not what the user is looking for. A link to the code can be found in Appendix A.

We conducted preliminary results with 5 participants. We asked users to interact with the tool, and we recorded how participants feel about interaction with the tool. We were not able to record the time and number of clicks it takes to find a file without using

the tool versus using the tool to determine if the tool improves productivity.

2 RESEARCH QUESTIONS

Our work is motivated by two research questions:

RQ1. How can simple natural language inputs improve file searching experience?
RQ2. How does a mixed-initiative approach allow a user to recover from badly parsed query?

Our hypotheses are as follows:

H1. Simple natural language inputs allows for quicker searching when you can't remember a file name.
H2. A mixed-initiative approach improves the user experience by allowing a user and tool to work together.

We consider an informal analysis of these hypotheses based on a few preliminary results from a small case study.

3 RELATED WORK

Natural Language Processing is being applied to an increasing number of domains. It shows promising results for converting user requests to shell commands, converting comments to source code, searching for papers, and searching for videos among other applications. Our tool builds on related work around approaches to search queries in various domains as well as mixed-initiative interface research.

To the best of our knowledge, no tool exists to use natural language queries to locate files on a user's system. Existing research has mainly focused on large databases and web searches where there are exponentially more files to search through. In those situations, query efficiency becomes paramount because database structure would be too complex to manually navigate through. File searching becomes an interesting case because a user with a well-organized folder structure might find that searching provides little benefit. We will expand on these results in our participant study section where we found that users approaches to file management and retrieval varied.

3.1 Paper Search

Natural language has been applied specifically for searching databases for papers. RusNLP is a semantic search engine and recommendation system that retrieves papers in Russian from 3 major conferences [14]. In conjunction with NLPub.ru, it can also understand the general focus of the research papers. RusNLP used 3 models - TF-IDF, LDA and Paragraph Vector to perform the semantic search. Out of the three models, TD-IDF performed the best. With the dataset being relatively small, solid conclusions could not be drawn and thus, the future work involves extending the corpus to more conferences and of different languages.

3.2 Video Search

Natural language has been applied to searching for videos. Jiang et al. presented a way to perform semantic search for videos on the internet. They propose E-Lamp, a state-of-the-art semantic video search engine, which matches textual queries for the user to the context of videos through a multimodal indexing search. Finally, they use pseudo-relevance feedback to refine the results of the search. They also provide a comparison of different retrieval models, and how they affect the final search results [11].

3.3 Software Code Search

Software search tools help developers find source code relevant to their maintenance tasks. One major challenge to successful search tools is to locate relevant code when the user's query contains words with multiple meanings or words that occur frequently throughout the program. Hill et al.[7] present a novel search technique to locate the source code that uses information such as the position of the query word and its semantic role to calculate the relevance. The accuracy of search technique used was limited and exploration of additional search techniques on more search tasks was proposed as a future work.

3.4 Shell Commands

Research has investigated the use of NLP to understand Linux commands that a user wishes to run. NL2Bash has shown promising advancements in this regard by developing a dataset of code snippets to natural language queries. NL2Bash demonstrated comparable performance to other tools, but found there was considerable room for improvement in accuracy and the number of commands supported [13]. As will be described in our approach, we are performing an abstracted version of this by using natural language to determine an intermediate logical form which then determines what mdfind terminal command to run.

3.5 Software Development

Natural Language Processing has been investigated for speeding up software development by translating natural descriptions of code segment into code. GitHub recently announced the introduction of Copilot, an "AI Pair Programmer" that writes code based on programmer natural language comments as well as features such as auto-filling repetitive code [6]. Copilot's effectiveness has already been investigated, to mixed results. It has been noted that the tool is quite helpful and saves time, but can introduce security concerns in code and should be monitored carefully [15].

Similarly, NLP has been used in the opposite direction for providing explanations for source code. The CODE-NN is a tool that learns from StackOverflow, a widely used site for programming-related questions, to understand what language correlates to code. It uses this knowledge to associate code sections with a question that could be asked about what the code section does [10].

3.6 File Retrieval

Previous studies have considered the way that graphical user interfaces can improve the process of hunting for files. It has been found that users prefer navigating file structure to find files rather than searching, however, search was used as a fallback method [2]. This was finding was echoed in participant statements in our preliminary results. Other studies have found that file access history can be important to helping a user quickly discover files that they have recently worked on [3]. Likewise, varying the amount of highlight on folders and files has been explored to help draw eyes

to recently used files that are likely to be relevant. It was found that this highlighting scheme helped reduce file retrieval time [4].

3.7 Natural Language Queries

Earlier studies have looked at natural language queries for document contents in a database by providing each document with a natural language description and comparing the description to the user's request [16]. This method was found to be somewhat effective and presents a different approach than we chose since we cannot manually associate every file on a user's systems with a way that they would search for the file.

3.8 Mixed-Initiative Interfaces

Mixed-initiative interfaces have been proposed to allow joint collaboration between humans and AI when completing a task. Most notably, design guidelines for these interfaces have been created by Eric Horvitz [8]. We aim to implement 5 of these principles:

- Develop significant value-added automation
- Consider uncertainty about a user's goals
- Allow efficient direct invocation and termination
- Provide mechanisms for efficient agent-user collaboration to refine results
- Employ socially appropriate behaviors for agent-user interaction

These five principles are most appropriate since this tool does not use a full-fledged conversational AI that might benefit from the other 7 principles outlined by Horvitz [8].

Mixed-initiative approaches have shown success for visualizing ambiguity in natural language queries [5]. Since natural language will almost always contain ambiguities that even humans have to clarify, a mixed-initiative approach seems quite promising. We apply simple query visualization techniques to our tool to help resolve ambiguities that may occur.

4 APPROACH

This section describes how to apply NLP to find files. We will parse a query into metadata about a file and search by that metadata. The metadata will include terms similar to the ones the user specified since language can often have multiple words for the same meaning. Figure 1 contains a flow diagram explaining the technical approach, and Figure 2 displays an example of this approach in action. We make use of the SEMPRE framework to get the list of the essential keywords from the users' natural language query. We could then use ConceptNet to get the list of the related synonyms for the keywords so that the intermediate logical form for each keyword remains the same. However, for the scope of this project, this step was left out. Once the intermediate logical forms are drawn, the file retrieval algorithm would fetch the list of files matching the representation. The resultant file list is displayed to the user.

4.1 Invoking the Tool

The tool is built for MacOS since it is is a widely-used platform, has an efficient file indexer enabled by default, and is the platform most available to the authors. We wanted to add an icon to the toolbar of Finder, the file manager of MacOS, that can invoke the tool while the user is already looking at files. Additionally, we wanted to add a keyboard shortcut or menu bar icon to invoke the tool so that the user does not have to open Finder in order to invoke the tool. However, because we faced issues with SEMPRE output parsing, we devoted our time there and these features were not implemented. These invocation methods should be included in future work to better conform with mixed initiative design principles [8].

4.2 Natural Language Parsing

The Stanford Natural Language Processing Group built a semantic parsing tool called SEMPRE, which stands for Semantic Parsing with Execution. It maps utterances in natural language to the appropriate response by first parsing them into logical forms. SEMPRE allows the usage of multiple logical forms like lambda calculus, lambda DCS, Java expressions, etc. Let's look at an example of the question, "When do the NBA league matches begin?". This should be parsed into a logical form such as (start_date NBA_league). After a database or a web search, the resultant solution (also called denotation) would be October 3rd.

Designers can specify the grammar for parsing as a set of rules. The rules are not limited to any particular logical form, but can be used to combine many logical forms to build a grammar with depth. By default, SEMPRE runs on a basic NLP processing model which is called Simple Language Analyser.

To leverage more sophisticated functionalities, support for Stanford CoreNLP is provided. Specifying the grammar is much like hard coding the way to derive a set of utterances. While grammars may be easy to parse for simple use cases, it becomes inefficient for larger and more complex problems. Thus, the developers equipped the tool with online learning algorithms that can choose the best derivation (based on features), given a set of rules. It calculates the conditional log-likelihood of the denotations as the score to determine the best fitting rule, and the optimization technique applied is the stochastic gradient descent. It also supports batch and online learning. Another advantageous feature of SEMPRE is it can query graph databases using lambda DCS.

In our smart search system, we aim to parse a question like "can you find slides that I made 2 or 3 years ago?", where the derivations focus on the words and phrases marked in italics. The grammar is written in a way such that the word "slides" can be interchanged with "presentation". The numerical values "2" and "3" are subtracted from the present year (2021) to derive the years the user is looking for. The word "years" can be interchanged with months, days, or other time periods, and affects the subtraction derivation mentioned previously. The word "ago" is used as an operator that triggers the subtraction. This is an example of how we will be leveraging SEMPRE to parse search queries.

4.3 ConceptNet

ConceptNet coudl be used to identify the synonyms for the keywords present in the Intermediate Logical Form. A user can specify a Powerpoint presentation as pptx in the input. The mapping of textual information present in ConceptNet along with our grammar rules would make sure we have a common representation across

different keywords in our intermediate logical form. The intermediate logical form can then be passed to mdfind to retrieve the files desired by the user. We have removed this from the scope of the project due to its complexity.

4.4 Indexing and Searching

Once the user's query has been parsed to understand what file features should be searched, the system will need to actually find these files. MacOS indexes files by default as part of its "Spotlight Search" so there is no need to build a file indexer that would take up more space on the user's computer and perhaps be slower or unreliable. We invoke a Spotlight search by running the terminal command *mdfind* that allows for searching by file name or file features. The mdfind command automatically sorts file results by date that was last accessed to prioritize results that are likely to be relevant. We can pass metadata tags to mdfind with the metadata that we parsed from the user's request, reducing the syntax requirements of the user's request. Some common metadata tags include kind, date, and the application that created the file. Likewise, the mdfind allows specifications of what directory to search in case the user wants to only search the documents or downloads folder. Another advantage of using mdfind is that it allows for complex boolean logic queries. The user can specify logical conjunctions to multiple metadata tags to narrow down the results to a more precise query. The output of mdfind is a list of file paths that match the search. The tool takes these file paths and uses them to parse the file name along with a file preview that can be presented to the user. The tool uses Apple's QuickLook framework to display a file preview that is able to be scrolled by the user for some interactivity. We list up to 8 files that match the query. If less than 8 files match, then we only list those files and do not generate blank icons.

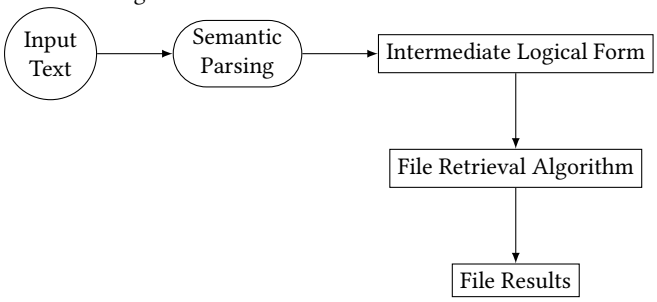

Figure 1: Flow chart of the Intelligent File System

Figure 2 displays an example of the user's query being parsed into intermediate logical form before being converted by our tool into an mdfind (Spotlight) search query that can run an indexed search on the user's system and return a file path. We take the file path and use Apple's QuickLook developer framework to show a preview of the file. If no preview can be generated (as is the case in Figure 2), then the file shows the app icon that would be used to open the file. This is the same behavior as in the MacOS Finder default file navigator application.

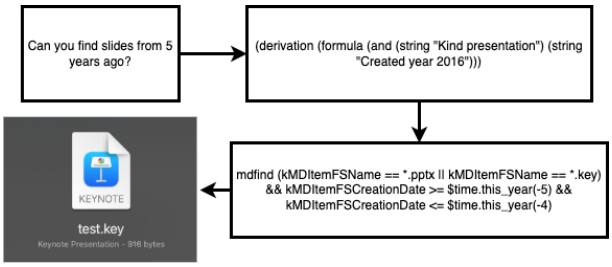

Figure 2: The user's query is parsed by SEMPRE to generate a logical form that is then converted by our application to mdfind query form. Finally, mdfind is run and a file is displayed to the user.

5 USAGE SCENARIO

The are four main usage scenarios that our tool's grammar currently supports:

(1) Can you find (slides/pdf/ppt/presentation/document/word doc) from (x) years ago?
(2) Can you find (slides/pdf/ppt/presentation/document/word doc) from (x) months ago?
(3) Find a file downloaded from (overleaf/brightspace/google docs) it might be in a folder named (fall21/spring21/verification/ receipts) or (fall21/spring21/verification/receipts)
(4) I'm looking for a (large/small) file that was edited in (january/february/.../december)

The words between parentheses are the possible options that can be inserted in the sentence. These four usages cover common queries for recently edited files based on our personal experiences.

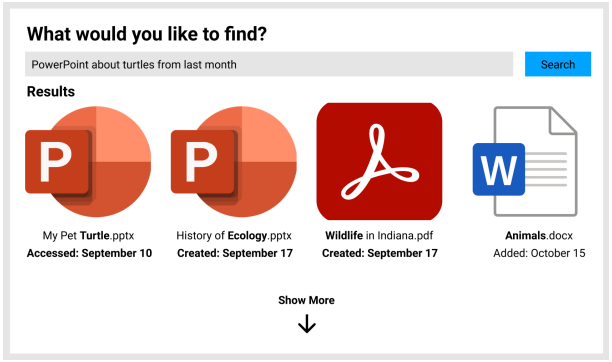

Figure 3: Early mock-up of user interface. The user can put a natural language query into the search box and a preview of a few files matching the query are displayed below.

Suppose Alice is a MacOS user that wants to find a PowerPoint about that she was working on last week. Because it has been a week since she last saved it, she cannot remember the name of the file, and many more files have piled up in her documents folder where it was saved. Without our tool, Alice would have to search

for terms that she thinks might be in the title or content of the file in order to find it. This might take several searches before she realizes what terms were used. If she wants to search by date, she would have to remember the exact date that she accessed the file and type "date:" followed by a formatted date. If she knew it was a PowerPoint she would have to also type "kind:pptx". The file might not be found if it was actually a PDF or Keynote file that she had accessed.

With our tool, Alice would only need to click an icon next to the Finder search bar and type in "can you find slides from a month ago?". Our tool automatically determines that she is looking for a file that is a presentation even if it is not directly a .pptx file. Our file might present PDFs or Keynote files that match other metadata. Additionally, our tool would recognize that "last month" is a date range, and it would search for all files in this date range. Likewise, Alice could have said "yesterday", and our tool would parse the date for yesterday based on the current date. Up to 8 results would be displayed for Alice to choose from.

As shown in the UI mock-up in Figure 4, the user sees a preview of files that the tool determines relate to the user's query. The user can click the Show/Hide Options button to display the metadata tags that the system has parsed from the query that were used in the search. The user can also edit these tags and click "Run search with these options" so as to recover from a failed search. This aligns with a mixed-initiative design principle from Horvitz that the tool should "Provide mechanisms for efficient agent-user collaboration to refine results" [8]. This may help the user understand times when the query was incorrectly parsed versus times when the query simply did not match the correct file. Ambiguities can also be resolved from this method if there are any words used that could have multiple meanings. Likewise, this improves algorithm explainability by helping the user understand what terms correspond to what metadata. Editing the query also allows for recovery from a misunderstanding and would prevent the user from needing to switch to another search tool to complete the query.

We follow the principle that the user may not always want to give feedback to the model [1]. To this end, we automatically hide the descriptions that allow to see the way their query was parsed. The user may view and modify these values if desired but it is not intrusive.

The file previews will are clickable so the user can directly open the files from our tool. The user can double click the file to quickly open the tool. If the file has scrollable content like a document or presentation, the user can scroll the preview to see some of the document's contents. The user can right click on the file to select "Open" as an alternative to a double click on the icon. The user can right click and select "Open in Finder" to open Finder with the current file highlighted. From here, the user can take more complex actions such as file renaming, viewing more file properties, or moving the file to another directory.

As shown in Figure 5, there is a help button at the bottom right of the screen that tells the user the 4 main use cases of the SEMPRE grammar that are included in above the Usage Scenario section. This button can be invoked at any time so the user recalls what queries are possible or if the user does not understand why their query was not parsed.

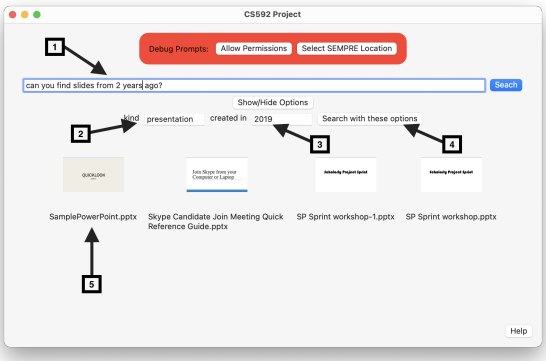

Figure 4: Usage Scenario 1/2. The user's query is placed into the search box (1). After hitting search the user can select the Show/Hide Options button to see the metadata that was parsed (2), (3). The user can edit this metadata if it was parsed incorrectly and select "Search with these options" (4) to run the tool with the updated metadata. Up to 8 files are displayed below (5) that match the user's query. A scrollable preview of the file is shown and the user can open the file or the file's location directly from the tool.

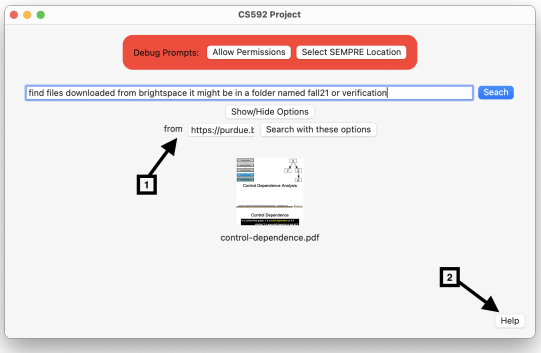

Figure 5: Usage Scenario 3. The user can tell that the "from brightspace" command has been parsed to equal "https://purdue.brightspace.com" (1). The text box is editable so the user can replace the URL and run the command again if this is not the correct URL. If the user wonders what websites are supported, they can select the help button (2) to display the possible options.

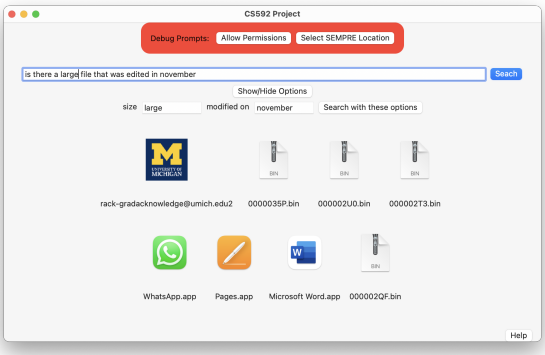

Figure 6: Usage Scenario 4. The user can search for large or small files that were edited in a particular month of the current year.

As shown in Figure 6, the tool allows for searching by "large" or "small" files. We determined a small file to be less than 5KB and a large file to be anything greater than 5KB.

6 PRELIMINARY RESULTS

We conducted a case study with 5 participants (5 male, 0 female), ages 21 to 24, with backgrounds in computer science (2 undergraduate, 3 graduate). One user had experience with NLP, and 4 users had used Spotlight before. We asked the users several questions before using the tool and after using the tool to get an understanding of how they felt about the tool.

Before using the tool, we asked the participants:

(1) How do you search for files normally when you can't remember a file name?
(2) Have you used Spotlight search before?

After using the tool, we asked the participants:

(1) How was your experience?
(2) How did your experience compare to the old way you searched way for files?
(3) Rate your experience from 1 to 5 with options of very unsatisfied, unsatisfied, neutral, satisfied, very satisfied, with 1 being very unsatisfied and 5 being very satisfied.

On a 5 point Likert scale, our participants gave their experience an average 3.8 out of 5. Our results can be summarized in several categories. Because the tool requires the user to also download and install SEMPRE, we had the users run the tool on our computers and interact with the results. This presents a limitation to the realism, as the user would be better at evaluating the tool on their own computer where they are more accustomed with their own file structure. Users were shown the graphic that appears when they press the help button so they understood the possible queries. We verbally explained how the Show/Hide Options buttons works to allow users to edit a parsed query. We verbally explained how the Show/Hide Options buttons works to allow users to edit a parsed query. Our results can be summarized in several categories.

6.1 File Retrieval Methods Prior to our Tool

One participant stated that they commonly use Spotlight or terminal command to search files when only vague details are remembered. If these did methods do not work then they result to manually searching through the folders. Another user stated that they never use the Windows search feature but instead rely on their strong file organization structure. One user stated that they had thousands of documents in their downloads folder and that they rarely organize files.

6.2 Time Reduction

Based on much of the research around file searching from the related works section, time reduction is a big motivation for improving file retrieval. On this note, one participant commented that the tool speeds up the file retrieving process and helps minimize effort. Another participant felt that this tool is quick for retrieving downloaded files when the user only remembers a given timeframe.

6.3 Mixed-Initiative Success

Overall, participants found that the ability to modify the way a query was parsed to be quite helpful. One participant stated "I really appreciated the feature of correcting the query when it's parsed wrong. It saves a lot of time that could get wasted in trying to guess the correct natural language phrase." One user commented that the user interface was very interactive and simple to use.

6.4 Limited Grammar

Concerning the the number of grammar rules, one user felt that the number of query options was not enough or not all common files types were represented. This finding highlights the need for implementing SEMPRE's active learning that can broaden the number of rules automatically.

6.5 Broad Queries

One participant suggested the the provided queries of year and month are too broad and that the "number of files you're searching through could be hundreds." The participant would have liked smaller queries based on week or day. Likewise, the participant thought that this limited is amplified by the fact that a person who needs this tool is likely to have many files, not just a few and only showing 8 files is not enough to be relevant. This is in contrast to what other participants had said that the tool is helpful when given a timeframe.

6.6 Usefulness

One participant stated that they do not usually search by metadata because if they do not remember a file's name, they just go to the folder where they know it should most likely be. They felt that the tool would not be very useful to them because they keep a very organized folder structure. This could perhaps contradict one of our initial assumptions that users commonly have to resort to metadata to search for files. Another participant felt that the use cases were very limited and felt that the tool could be improved by parsing document contents as well as metadata in order to provide more relevant results.

6.7 Natural Language as a way to Search

One participant stated that "it was a neat prototype" and they "could see the tool being useful compared to Spotlight" but that they were not convinced that natural language is an effective search method. They found that Spotlight allowed them to type metadata properties freeform into the query and it did a good job of doing a logical AND conjunction of the properties to parse a result. They felt that Spotlight was more natural and quicker to respond compared to our tool.

6.8 Results Summary

Overall, participant results were positive but feedback about the tool's usefulness was mixed. These mixed results can be seen the varying usefulness of this tool based on the way the user structures their files and if they already rely on traditional search methods. For users who have very unorganized files, this tool was seen as helpful. However, for users with very organized files, a preference was seen for navigating directly to the folder. This aligns with what related work noted about file retrieval [2]. For users who already use Spotlight, this tool was seen as a slight improvement to certain functionality and the users felt that being able to modify the way the query was parsed would be a helpful feature for Spotlight to integrate.

7 DISCUSSIONS AND FUTURE WORK

In general, our tool was seen favorably as being helpful and providing a clean user interface. The ability to modify the metadata that was parsed from the query is a highlight of the tool and was well-received by the participants. The main limitations with the tool lie in its limited amount of queries.

Considering our hypotheses, both H1 and H2 appear to be supported by preliminary result question responses. However, we did not conduct a large enough study with enough numerical data to create statistically significant results. This could be an important contribution of a future work.

The natural language based file search relies heavily on SEMPRE. As part of the course project, the basic features of SEMPRE have been explored through the SimpleLanguageAnalyzer module. Since the grammar has been hard coded into the tool, it tends to overgenerate, and the tool does not scale feasibly. We do not recommend using a hard-coded rule-based grammar for future work based on this scaling issue. The tool can be made more intelligent by utilizing the coreNLP module which would allow the integration of a learning component. The parser can be trained on datasets to make it learn how to choose the most appropriate derivations. Additionally, it would be nice if this learning component could be incorporated with a UI actions. The user could right click a file and have an option to select "Not relevant" to provide feedback that the file did not match the description. This would follow the principle that the user may not always want to give feedback to the model by allowing the feedback to be optional. The feedback would not be intrusive because it is accessed through a right click option rather than a button always cluttering the view [1].

Another very interesting component of SEMPRE is the ContextEntity feature that can be used as a "short-term" memory to remember the context from previous queries. ContextEntity helps the tool remember what words like "it","he","hers" mean in the given context. Making the interactions conversational could be very helpful in incrementally refining search results.

We found a large drawback of SEMPRE is its inability to run as a background process. SEMPRE is designed to be run as command line interface tool and thus we had to develop our own tool to run SEMPRE in the background and redirect the standard out file descriptor to a file that we could then scrape. This process works well for our scenarios but as grammar becomes more complex, its robustness could degrade. This could be avoided by the ability to pass SEMPRE a single command to run and quit and an output folder for the result.

In a real-world premise, scaling becomes a hurdle as more generative rules will be needed. To alleviate some of the load off of SEMPRE, ConceptNet or WordNet can be incorporated. With these technologies in the pipeline, multiple synonyms of a word can be mapped to the keyword required to conduct the search.

Concerning UI improvements, we would liked to have integrated the application into a menu bar application, created a tab bar icon in Finder for the application, or created a keyboard shortcut to invoke the tool. We believe this would have satisfied one of Horvitz's design principles that a tool should allow easy invocation and dismissal [8]. Another helpful UI feature could include bolding the relevant words in the file name that the user sees under the file preview. This would help the user understand which aspects of the file the tool thought were pertinent. Sometimes a search query can be somewhat slow since the tool waits until all of the search is done to display results. However, it could be more helpful to start displaying values as soon as Spotlight returns them. This could improve file search speed, reducing the time it takes for the user to start seeing relevant files. It will be a nice addition if the user can press the space bar to open Quick Look like in Finder and view file contents in more detail than just the small preview that is given. This could help the user determine if the file is really what they are looking for without having to take the time to open the file. Likewise, it will be nice if the user can right click the files and perform normal Finder right click actions such as copying and renaming from our file viewer. However, this may be left to future work since it adds little to the user experience of our system. We think another UI improvement could be displaying the file's location at the bottom of the screen when you click on a file. Finder offers similar functionality and it can help the user understand which file is being shown when there are multiple files with the same name on the user's system.

Future work could additionally consider using SiriKit to make the app voice activated and usable completely through Siri. This not only could be more convenient than typing out the commands, but it could improve accessibility of the tool for visually impaired users. Likewise, since the functionality would live inside Siri, the user could invoke Siri with their voice or the existing Siri button rather than having to create another button on their menu bar specifically for our app.

8 CONCLUSION

Natural language processing has been applied to many scenarios and is a promising research area. Natural language has been applied to large database queries but not files on a personal computer. While

file navigation has been extensively studied to decrease the time it takes for users to perform everyday computer tasks, natural language methods for file retrieval have not.

In this paper we prototyped a natural language file searcher tool that runs on a user's MacOS system. The system uses the SEMPRE framework to parse the user's query and creates a query that is sent to MacOS Spotlight indexed search using the mdfind command. The tool implements mixed-initiative methods to help the user understand how the query was parsed and recover from a poorly parsed query.

Preliminary results with 5 users indicate that the tool is novel but suffers from a small set of grammar rules. Future work could expand these grammar rules and benefit from using the learning component of SEMPRE to build rules and the system learns from the user. Future work could benefit from deploying this tool in the wild and running studies on how long it takes on average for a user to find the file using the tool verses navigating folders. Likewise, it could be important to study how many times the tool fails to correctly parse a query.

9 ACKNOWLEDGMENTS

We would like to acknowledge David Kim, Pallav Agarwal, Rohan Simha, Vignesh Somasundaram, and Rahul Senguttuvan for their help in evaluating this tool and providing preliminary results.

REFERENCES

[1] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine* 35, 4 (Dec. 2014), 105–120. https://doi.org/10.1609/aimag.v35i4.2513

[2] Ofer Bergman, Ruth Beyth-Marom, Rafi Nachmias, Noa Gradovitch, and Steve Whittaker. 2008. Improved Search Engines and Navigation Preference in Personal Information Management. *ACM Trans. Inf. Syst.* 26, 4, Article 20 (oct 2008), 24 pages. https://doi.org/10.1145/1402256.1402259

[3] Stephen Fitchett, Andy Cockburn, and Carl Gutwin. 2013. Improving Navigation-Based File Retrieval. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2329–2338. https://doi.org/10.1145/2470654.2481323

[4] Stephen Fitchett, Andy Cockburn, and Carl Gutwin. 2014. Finder Highlights: Field Evaluation and Design of an Augmented File Browser. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 3685–3694. https://doi.org/10.1145/2556288.2557014

[5] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 489–500. https://doi.org/10.1145/2807442.2807478

[6] GitHub. [n. d.]. *GitHub Copilot: Your AI Pair Programmer*. https://copilot.github.com

[7] Emily Hill, Lori Pollock, and K. Vijay-Shanker. 2011. Improving source code search with natural language phrasal representations of method signatures. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 524–527. https://doi.org/10.1109/ASE.2011.6100115

[8] Eric Horvitz. 1999. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) *(CHI '99)*. Association for Computing Machinery, New York, NY, USA, 159–166. https://doi.org/10.1145/302979.303030

[9] Matthew B Hoy. 2018. Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants. *Medical reference services quarterly* 37, 1 (2018), 81–88.

[10] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 2073–2083. https://doi.org/10.18653/v1/P16-1195

[11] Lu Jiang, Shoou-I Yu, Deyu Meng, Teruko Mitamura, and Alexander G. Hauptmann. 2015. Bridging the Ultimate Semantic Gap. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ACM. https://doi.org/10.1145/2671188.2749399

[12] Jung-Hun Lee, Minho Kim, and Hyuk-Chul Kwon. 2020. Deep Learning-Based Context-Sensitive Spelling Typing Error Correction. *IEEE access* 8 (2020), 152565–152578.

[13] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. 2018. NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System. arXiv:1802.08979 [cs.CL]

[14] Irina Nikishina, Amir Bakarov, and Andrey Kutuzov. 2018. RusNLP: Semantic Search Engine for Russian NLP Conference Papers. In *Analysis of Images, Social Networks and Texts*, Wil M. P. van der Aalst, Vladimir Batagelj, Goran Glavaš, Dmitry I. Ignatov, Michael Khachay, Sergei O. Kuznetsov, Olessia Koltsova, Irina A. Lomazova, Natalia Loukachevitch, Amedeo Napoli, Alexander Panchenko, Panos M. Pardalos, Marcello Pelillo, and Andrey V. Savchenko (Eds.). Springer International Publishing, Cham, 111–120.

[15] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2021. An Empirical Cybersecurity Evaluation of GitHub Copilot's Code Contributions. arXiv:2108.09293 [cs.CR]

[16] Tomek Strzalkowski, Fang Lin, Jose Perez-Carballo, and Jin Wang. 1997. Building Effective Queries in Natural Language Information Retrieval. In *Proceedings of the Fifth Conference on Applied Natural Language Processing* (Washington, DC) *(ANLC '97)*. Association for Computational Linguistics, USA, 299–306. https://doi.org/10.3115/974557.974601

[17] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144

A GITHUB LINK

https://github.com/bbhardin/cs592-project

B GRAMMAR FOR SEMPRE

(rule *Intermediate*1(*you*)(*ConstantFnnull*))
(*rule*Intermediate1 (can) (ConstantFn null))
(rule *Intermediate*1(*canyou*)(*ConstantFnnull*))
(*rule*Intermediate2 (find) (ConstantFn null))
(rule *Intermediate*2(Intermediate1) (ConstantFn null))
(rule *Intermediate*2(Intermediate1 find) (ConstantFn null))
(rule *Intermediate*3(*a*)(*ConstantFnnull*))
(*rule*Intermediate3 (*Intermediate*2)(*ConstantFnnull*))
(*rule*Intermediate3 (*Intermediate2a*)(*ConstantFnnull*))
(*rule*Intermediate4 (*Entity*1)(*SelectFn*0))
(*rule*Intermediate4 (*Intermediate*3Entity1) (SelectFn 1))
(rule *Intermediate*5(Intermediate4) (SelectFn 0))
(rule *Intermediate*5(Intermediate4 ago) (SelectFn 0))
(rule *Intermediate*6(Intermediate5) (SelectFn 0))
(rule *Intermediate*6(Intermediate5 ?) (SelectFn 0))
(rule *ROOT*(Intermediate6) (IdentityFn))
(rule *Intermediate*7(*find*)(*ConstantFnnull*))
(*rule*Intermediate7 (please) (ConstantFn null))
(rule *Intermediate*7(*pleasefind*)(*ConstantFnnull*))
(*rule*Intermediate8 (a) (ConstantFn null))
(rule *Intermediate*8(Intermediate7) (ConstantFn null))
(rule *Intermediate*8(Intermediate7 a) (ConstantFn null))
(rule *Intermediate*9(Entity1) (SelectFn 0))
(rule *Intermediate*9(Intermediate8 *Entity*1)(*SelectFn*1))
(*rule*Intermediate10 (*Intermediate*9)(*SelectFn*0))
(*rule*Intermediate10 (*Intermediate9ago*)(*SelectFn*0))
(*rule*ROOT (*Intermediate*10)(*IdentityFn*))
(*rule*Intermediate11 (a) (ConstantFn null))
(rule *Intermediate*11(*find*)(*ConstantFnnull*))
(*rule*Intermediate11 (find a) (ConstantFn null))
(rule *Intermediate*12(Entity1) (SelectFn 0))
(rule *Intermediate*12(Intermediate11 *Entity*1)(*SelectFn*1))
(*rule*Intermediate13 (*Intermediate*12)(*SelectFn*0))
(*rule*Intermediate13 (*Intermediate*12*ago*)(*SelectFn*0))
(*rule*ROOT (*Intermediate*13)(*IdentityFn*))
(*rule*Intermediate14 (you) (ConstantFn null))
(rule *Intermediate*14(*can*)(*ConstantFnnull*))
(*rule*Intermediate14 (can you) (ConstantFn null))
(rule *Intermediate*15(*look*)(*ConstantFnnull*))
(*rule*Intermediate15 (*Intermediate*14)(*ConstantFnnull*))
(*rule*Intermediate15 (*Intermediate*14*look*)(*ConstantFnnull*))
(*rule*Intermediate16 (for) (ConstantFn null))
(rule *Intermediate*16(Intermediate15) (ConstantFn null))
(rule *Intermediate*16(Intermediate15 for) (ConstantFn null))
(rule *Intermediate*17(*a*)(*ConstantFnnull*))
(*rule*Intermediate17 (*Intermediate*16)(*ConstantFnnull*))
(*rule*Intermediate17 (*Intermediate*16*a*)(*ConstantFnnull*))
(*rule*Intermediate18 (*Entity*1)(*SelectFn*0))
(*rule*Intermediate18 (*Intermediate*17Entity1) (SelectFn 1))
(rule *Intermediate*19(Intermediate18) (SelectFn 0))
(rule *Intermediate*19(Intermediate18 ago) (SelectFn 0))
(rule *Intermediate*20(Intermediate19) (SelectFn 0))
(rule *Intermediate*20(Intermediate19 ?) (SelectFn 0))
(rule *ROOT*(Intermediate20) (IdentityFn))
(rule *Intermediate*21(*for*)(*ConstantFnnull*))
(*rule*Intermediate21 (look) (ConstantFn null))
(rule *Intermediate*21(*lookfor*)(*ConstantFnnull*))
(*rule*Intermediate22 (a) (ConstantFn null))
(rule *Intermediate*22(Intermediate21) (ConstantFn null))
(rule *Intermediate*22(Intermediate21 a) (ConstantFn null))
(rule *Intermediate*23(Entity1) (SelectFn 0))
(rule *Intermediate*23(Intermediate22 *Entity*1)(*SelectFn*1))
(*rule*Intermediate24 (*Intermediate*23)(*SelectFn*0))
(*rule*Intermediate24 (*Intermediate*23*ago*)(*SelectFn*0))
(*rule*ROOT (*Intermediate*24)(*IdentityFn*))
(*rule*Entity1 (*Set*1Set2) (MergeFn and))
(rule *Set*1(*slides*)(*ConstantFn*(*string*"*Kindpresentation*")))
(*rule*Set1 (presentation) (ConstantFn (string "Kind presentation")))
(rule *Set*1(*ppt*)(*ConstantFn*(*string*"*Kindpresentation*")))
(*rule*Set1 (pdf) (ConstantFn (string "Kind PDF document")))
(rule *Intermediate*25(*pdffile*)(*ConstantFnnull*))
(*rule*Set1 (*Intermediate*25)(*ConstantFn*(*string*"*KindPDFdocument*")))
(*rule*Set1 (document) (ConstantFn (string "Kind Microsoft Word Document")))
(rule *Intermediate*26(*worddoc*)(*ConstantFnnull*))
(*rule*Set1 (*Intermediate*26)(*ConstantFn*(*string*"*Kind MicrosoftWordDocument*")))
(*rule*Intermediate27 (from a) (ConstantFn null))
(rule *Intermediate*28(Intermediate27 year) (ConstantFn null))
(rule *Set*2(Intermediate28) (ConstantFn (string "Created Year 2020")))
(rule *Intermediate*29(*from*2)(*ConstantFnnull*))
(*rule*Intermediate30 (*Intermediate*29*years*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*30)(*ConstantFn*(*string*"*CreatedYear*2019")))
(*rule*Intermediate31 (from 3) (ConstantFn null))
(rule *Intermediate*32(Intermediate31 years) (ConstantFn null))
(rule *Set*2(Intermediate32) (ConstantFn (string "Created Year 2018")))
(rule *Intermediate*33(*from*4)(*ConstantFnnull*))
(*rule*Intermediate34 (*Intermediate*33*years*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*34)(*ConstantFn*(*string*"*CreatedYear*2017")))
(*rule*Intermediate35 (from 5) (ConstantFn null))
(rule *Intermediate*36(Intermediate35 years) (ConstantFn null))
(rule *Set*2(Intermediate36) (ConstantFn (string "Created Year 2016")))
(rule *Intermediate*37(*from*6)(*ConstantFnnull*))
(*rule*Intermediate38 (*Intermediate*37*years*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*38)(*ConstantFn*(*string*"*CreatedYear*2015")))
(*rule*Intermediate39 (from 7) (ConstantFn null))
(rule *Intermediate*40(Intermediate39 years) (ConstantFn null))
(rule *Set*2(Intermediate40) (ConstantFn (string "Created Year 2014")))
(rule *Intermediate*41(*from*8)(*ConstantFnnull*))
(*rule*Intermediate42 (*Intermediate*41*years*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*42)(*ConstantFn*(*string*"*CreatedYear*2013")))
(*rule*Intermediate43 (from 9) (ConstantFn null))
(rule *Intermediate*44(Intermediate43 years) (ConstantFn null))
(rule *Set*2(Intermediate44) (ConstantFn (string "Created Year 2012")))
(rule *Intermediate*45(*from*10)(*ConstantFnnull*))
(*rule*Intermediate46 (*Intermediate*45*years*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*46)(*ConstantFn*(*string*"*CreatedYear*2011")))
(*rule*Intermediate47 (from a) (ConstantFn null))
(rule *Intermediate*48(Intermediate47 month) (ConstantFn null))

(rule *Set*2(Intermediate48) (ConstantFn (string "Created Month November")))
(rule *Intermediate*49(*from*2)(*ConstantFnnull*))
(*rule*Intermediate50 (*Intermediate*49*months*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*50)(*ConstantFn*(*string*"*CreatedMonthOctober*")))
(*rule*Intermediate51 (from 3) (ConstantFn null))
(rule *Intermediate*52(Intermediate51 months) (ConstantFn null))
(rule *Set*2(Intermediate52) (ConstantFn (string "Created Month September")))
(rule *Intermediate*53(*from*4)(*ConstantFnnull*))
(*rule*Intermediate54 (*Intermediate*53*months*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*54)(*ConstantFn*(*string*"*CreatedMonthAugust*")))
(*rule*Intermediate55 (from 5) (ConstantFn null))
(rule *Intermediate*56(Intermediate55 months) (ConstantFn null))
(rule *Set*2(Intermediate56) (ConstantFn (string "Created Month July")))
(rule *Intermediate*57(*from*6)(*ConstantFnnull*))
(*rule*Intermediate58 (*Intermediate*57*months*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*58)(*ConstantFn*(*string*"*CreatedMonthJune*")))
(*rule*Intermediate59 (from 7) (ConstantFn null))
(rule *Intermediate*60(Intermediate59 months) (ConstantFn null))
(rule *Set*2(Intermediate60) (ConstantFn (string "Created Month May")))
(rule *Intermediate*61(*from*8)(*ConstantFnnull*))
(*rule*Intermediate62 (*Intermediate*61*months*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*62)(*ConstantFn*(*string*"*CreatedMonthApril*")))
(*rule*Intermediate63 (from 9) (ConstantFn null))
(rule *Intermediate*64(Intermediate63 months) (ConstantFn null))
(rule *Set*2(Intermediate64) (ConstantFn (string "Created Month March")))
(rule *Intermediate*65(*from*10)(*ConstantFnnull*))
(*rule*Intermediate66 (*Intermediate*65*months*)(*ConstantFnnull*))
(*rule*Set2 (*Intermediate*66)(*ConstantFn*(*string*"*CreatedMonthFebruary*")))
(*rule*Intermediate67 (files) (ConstantFn null))
(rule *Intermediate*67(*find*)(*ConstantFnnull*))
(*rule*Intermediate67 (find files) (ConstantFn null))
(rule *Intermediate*68(*downloaded*)(*ConstantFnnull*))
(*rule*Intermediate68 (*Intermediate*67)(*ConstantFnnull*))
(*rule*Intermediate68 (*Intermediate*67*downloaded*)(*ConstantFnnull*))
(*rule*Intermediate69 (from) (ConstantFn null))
(rule *Intermediate*69(Intermediate68) (ConstantFn null))
(rule *Intermediate*69(Intermediate68 from) (ConstantFn null))
(rule *Intermediate*70(Entity2) (SelectFn 0))
(rule *Intermediate*70(Intermediate69 *Entity*2)(*SelectFn*1))
(*rule*Intermediate71 (*Intermediate*70)(*SelectFn*0))
(*rule*Intermediate71 (*Intermediate*70*please*)(*SelectFn*0))
(*rule*Intermediate72 (*Intermediate*71)(*SelectFn*0))
(*rule*Intermediate72 (*Intermediate*71?)(*SelectFn*0))
(*rule*ROOT (*Intermediate*72)(*IdentityFn*))
(*rule*Intermediate73 (for) (ConstantFn null))
(rule *Intermediate*73(*search*)(*ConstantFnnull*))
(*rule*Intermediate73 (search for) (ConstantFn null))
(rule *Intermediate*74(*a*)(*ConstantFnnull*))
(*rule*Intermediate74 (*Intermediate*73)(*ConstantFnnull*))
(*rule*Intermediate74 (*Intermediate*73*a*)(*ConstantFnnull*))
(*rule*Intermediate75 (file) (ConstantFn null))
(rule *Intermediate*75(Intermediate74) (ConstantFn null))
(rule *Intermediate*75(Intermediate74 file) (ConstantFn null))
(rule *Intermediate*76(*downloaded*)(*ConstantFnnull*))
(*rule*Intermediate76 (*Intermediate*75)(*ConstantFnnull*))
(*rule*Intermediate76 (*Intermediate*75*downloaded*)(*ConstantFnnull*))
(*rule*Intermediate77 (from) (ConstantFn null))
(rule *Intermediate*77(Intermediate76) (ConstantFn null))
(rule *Intermediate*77(Intermediate76 from) (ConstantFn null))
(rule *Intermediate*78(Entity2) (SelectFn 0))
(rule *Intermediate*78(Intermediate77 *Entity*2)(*SelectFn*1))
(*rule*ROOT (*Intermediate*78)(*IdentityFn*))
(*rule*Intermediate79 (*Set*3*which*)(*SelectFn*0))
(*rule*Intermediate80 (*Intermediate*79*might*)(*SelectFn*0))
(*rule*Intermediate81 (*Intermediate*80*be*)(*SelectFn*0))
(*rule*Intermediate82 (*Intermediate*81*in*)(*SelectFn*0))
(*rule*Intermediate83 (*Intermediate*82*a*)(*SelectFn*0))
(*rule*Intermediate84 (*Intermediate*83*folder*)(*SelectFn*0))
(*rule*Intermediate85 (*Intermediate*84*named*)(*SelectFn*0))
(*rule*Entity2 (*Intermediate*85Entity3) (MergeFn and))
(rule *Intermediate*86(Set3 it) (SelectFn 0))
(rule *Intermediate*87(Intermediate86 might) (SelectFn 0))
(rule *Intermediate*88(Intermediate87 be) (SelectFn 0))
(rule *Intermediate*89(Intermediate88 in) (SelectFn 0))
(rule *Intermediate*90(Intermediate89 a) (SelectFn 0))
(rule *Intermediate*91(Intermediate90 folder) (SelectFn 0))
(rule *Intermediate*92(Intermediate91 named) (SelectFn 0))
(rule *Entity*2(Intermediate92 *Entity*3)(*MergeFnand*))
(*rule*Intermediate93 (google docs) (ConstantFn null))
(rule *Set*3(Intermediate93) (ConstantFn (string "Where from https://docs.google.com/document/")))
(rule *Set*3(*brightspace*)(*ConstantFn*(*string*"*Wherefrom https* : //*purdue.brightspace.com*/")))
(*rule*Set3 (overleaf) (ConstantFn (string "Where from https://www.overleaf.com/")))
(rule *Entity*3(Set4) (IdentityFn))
(rule *Intermediate*94(Set4 or) (SelectFn 0))
(rule *Entity*3(Intermediate94 *Set*4)(*MergeFnor*))
(*rule*Set4 (verification) (ConstantFn (string "Where verification")))
(rule *Set*4(*fall*21)(*ConstantFn*(*string*"*Wherefall*21")))
(*rule*Set4 (receipts) (ConstantFn (string "Where receipts")))
(rule *Set*4(*spring*21)(*ConstantFn*(*string*"*Wherespring*21")))
(*rule*Intermediate95 (am) (ConstantFn null))
(rule *Intermediate*95(*i*)(*ConstantFnnull*))
(*rule*Intermediate95 (i am) (ConstantFn null))
(rule *Intermediate*96(*looking*)(*ConstantFnnull*))
(*rule*Intermediate96 (*Intermediate*95)(*ConstantFnnull*))
(*rule*Intermediate96 (*Intermediate*95*looking*)(*ConstantFnnull*))
(*rule*Intermediate97 (for) (ConstantFn null))
(rule *Intermediate*97(Intermediate96) (ConstantFn null))
(rule *Intermediate*97(Intermediate96 for) (ConstantFn null))
(rule *Intermediate*98(*a*)(*ConstantFnnull*))
(*rule*Intermediate98 (*Intermediate*97)(*ConstantFnnull*))
(*rule*Intermediate98 (*Intermediate*97*a*)(*ConstantFnnull*))
(*rule*Intermediate99 (*Entity*4)(*SelectFn*0))
(*rule*Intermediate99 (*Intermediate*98Entity4) (SelectFn 1))
(rule *ROOT*(Intermediate99) (IdentityFn))
(rule *Intermediate*100(*there*)(*ConstantFnnull*))

(*rule*Intermediate100 (is) (ConstantFn null))
(rule *Intermediate*100(*isthere*)(*ConstantFnnull*))
(*rule*Intermediate101 (a) (ConstantFn null))
(rule *Intermediate*101(Intermediate100) (ConstantFn null))
(rule *Intermediate*101(Intermediate100 a) (ConstantFn null))
(rule *Intermediate*102(Entity4) (SelectFn 0))
(rule *Intermediate*102(Intermediate101 *Entity*4)(*SelectFn*1))
(*rule*ROOT (*Intermediate*102)(*IdentityFn*))
(*rule*Intermediate103 (*Set*5*that*)(*SelectFn*0))
(*rule*Intermediate104 (*Intermediate*103*was*)(*SelectFn*0))
(*rule*Intermediate105 (*Intermediate*104*edited*)(*SelectFn*0))
(*rule*Intermediate106 (*Intermediate*105*in*)(*SelectFn*0))
(*rule*Entity4 (*Intermediate*106Set6) (MergeFn and))
(rule *Intermediate*107(*largefile*)(*ConstantFnnull*))
(*rule*Set5 (*Intermediate*107)(*ConstantFn*(*string*"*Size* > 5*mb*")))
(*rule*Intermediate108 (small file) (ConstantFn null))
(rule *Set*5(Intermediate108) (ConstantFn (string "Size <5mb")))

(rule *Set*6(*january*)(*ConstantFn*(*string*"*ModifiedMonthJanuary*")))
(*rule*Set6 (february) (ConstantFn (string "Modified Month February")))
(rule *Set*6(*march*)(*ConstantFn*(*string*"*ModifiedMonthMarch*")))
(*rule*Set6 (april) (ConstantFn (string "Modified Month April")))
(rule *Set*6(*may*)(*ConstantFn*(*string*"*ModifiedMonthMay*")))
(*rule*Set6 (june) (ConstantFn (string "Modified Month June")))
(rule *Set*6(*july*)(*ConstantFn*(*string*"*ModifiedMonthJuly*")))
(*rule*Set6 (august) (ConstantFn (string "Modified Month August")))
(rule *Set*6(*september*)(*ConstantFn*(*string*"*ModifiedMonthSeptember*")))
(*rule*Set6 (october) (ConstantFn (string "Modified Month October")))
(rule *Set*6(*november*)(*ConstantFn*(*string*"*ModifiedMonthNovember*")))
(*rule*Set6 (december) (ConstantFn (string "Modified Month December")))