CURSUS

Științele limbajului

Roxana Patraș
Loredana Cuzmici
(editors)

Tools, Methods, and Solutions
for the Exploration of Romanian Corpora

INSTITUTUL EUROPEAN

Roxana Patraş, Loredana Cuzmici (editors), *Tools, Methods, and Solutions for the Exploration of Romanian Corpora*

INSTITUTUL EUROPEAN
Iaşi, Str. Băluşescu nr. 2, et. I, O P 6, CP 1309
euroedit@hotmail.com.; www.euroinst.ro

Descrierea CIP a Bibliotecii Naţionale a României

Tools, methods, and solutions for the exploration of Romanian corpora /
ed.: Roxana Patraş, Loredana Cuzmici. - Iaşi : Institutul European, 2024
Conţine bibliografie
ISBN 978-606-24-0395-9

I. Patraş, Roxana (ed.)
II. Cuzmici, Loredana (ed.)

811.135.1

PRINTED IN ROMANIA

Roxana Patraș, Loredana Cuzmici
(editors)

TOOLS, METHODS, AND SOLUTIONS FOR THE EXPLORATION OF ROMANIAN CORPORA

INSTITUTUL EUROPEAN
2024

This work was supported by a grant of the Romanian Ministry of Education and Research, CNCS–UEFISCDI, project number PN-III-P1-1.1-TE-2019-0127, within PNCDI III.

The revision and editing of this collection was performed by BolderWords translation & revision and was supported by the Altissia Chair in Digital Cultures and Ethics held by Chris Tanasescu at Université catholique de Louvain.

Table of Contents

The Many Faces of *Virtual*: charting the family resemblance network for conceptual understanding

Matei Alexandru STOENESCU

Semantic fuzziness and a full spectrum of meanings for *virtual*

The term *virtual* and its related concepts have become ubiquitous across modern scientific discourse and cultural narratives. This family of concepts, dynamic and expansive, has the potential to reshape our cognitive and behavioral frameworks across multiple fields. *Virtual* carries a broad array of interpretations—some sharply delineated, others more fluidly connected. This study sets out on an epistemological quest to delineate and elucidate this cluster of meanings through the lens of family resemblances, as originally conceptualized by Ludwig Wittgenstein in his seminal work, *Philosophical Investigations*.

This article builds upon our prior research, concluding with the Ph.D. thesis *From Fictional Worlds to Virtual Worlds. Drivers of Change Between Perspectives on Fictional, Virtual, and Real.* In this previous work, we explored the essence of fictional and virtual worlds through the lens of the possible-worlds theory, principally as delineated by David Lewis (Lewis 2). We showed that fictional and virtual worlds constitute a dynamic framework intimately connected to the real world, suggesting that each entity regarded as *fictional*, *virtual*, or *real* can transition into one of the other states. Furthermore, we identified and discussed what we believe to be the fundamental drivers of these transformations. As part of the research process, we endeavored to chart the array of meanings attributed to the concept of the virtual, aiming to reveal the complex interconnections that form its family resemblance network. To that end, we gathered all the semantic nuances of "virtual" found in the

referenced literature, along with the term's diverse uses. We then manually plotted these findings on a dynamic Kumu chart, carefully delineating the relationships between uses and features. This endeavor was marked by three essential qualities: a high degree of accuracy in aligning uses with features, a considerably rich set of *semantic features*, and a relatively small initial set of uses, which was a direct consequence of the meticulous manual mapping process we undertook.

Despite the precision of our manual mapping and identifying the sets of uses and semantic traits, we must address the concern that this process does not entirely reflect the natural use of the term "virtual" in natural language. Hence, this article presents an alternative method aiming to capture a broader and more objective set of "virtuality" uses. Furthermore, in contrast to the previous dynamic chart—which primarily highlighted the intricate relationships formed between use and semantic traits—our recent visualization experiment underscores the vast array of meanings and the profound complexity of this concept.

The term "virtual" encompasses a full spectrum of meanings generated through use, some more clearly defined than others, some strongly and some loosely related. Drawing inspiration from Ludwig Wittgenstein's statements in paragraphs 66–69 of *Philosophical Investigations*, our analysis reveals that these semantic facets do not converge around a common core but instead form a complex network of relationships, described as family resemblances. We consider the construction of a panoramic view of the family resemblances for *virtual(ity)* a unique approach that serves to make a distinctive contribution to the field of Digital Humanities, enhancing our collective understanding of "virtuality" and expanding the existing body of scholarly knowledge. Our research methodology involved an extensive search process utilizing various tools, including integrations with Google Search API and scraping Google Scholar with third-party software. This approach yielded a substantial dataset comprising over 2000 results pertaining to the term "virtual." We subsequently employed this dataset to map and visually represent the intricate network of family resemblances, utilizing the Kumu platform.

Ludwig Wittgenstein's theory of "family resemblances," as elaborated in *Philosophical Investigations* (see also Ryan and Thon), offers a complex approach to

the human beings' conceptual understanding. The philosopher exemplifies his theory by looking at "game" and shows that when we investigate the different uses of the word game, we discover that while many correspondences can be found between uses, there are many features that drop out and others that appear. When looking at *board-games*, *card-games*, *chess*, *noughts and crosses*, *ball-games*, *a child's game of throwing a ball*, we should investigate *similarities* and *relations* in terms of *competition*, *amusement*, *winning and losing*, different types of *skill*, such as *skill in chess* and *skill in tennis*, *luck* etc. The similarities, correspondences and relationships between the features that appear or, in other cases, drop out form *a complicated network of similarities overlapping and criss-crossing: sometimes overall similarities, sometimes similarities of detail*. Built on this complex web of relationships, the term *family* is used by Wittgenstein to underscore his rejection of the idea that all members of a category should share a common core of similarities. In a family, for any given feature, there is at least one other member that shares it, but no trait is shared by all.[1]

Adopting a similar approach, we have delineated the network for "virtual(ity)," illustrating that each application of the concept unveils a distinct array of semantic nuances. These nuances then form a pattern of similarities or contrasts with other distinct applications. We note that certain features consistently emerge across uses while others recede. In addition to mapping similarities, we have introduced a contrasting relationship, particularly in relation to the concept *real*. The size of each node corresponds to the number of connections it has within the network, highlighting uses with extensive connections, such as *virtuality in fiction*, *virtuality of the real world*, *virtuality in storytelling*, and *virtualization*, as well as highly connected features like *real*, *digital*, *world*, *not actual*, *technology*, *possible worlds*, and *choice*. Through this method of visualization and prioritizing by connectivity, we show that

[1] The Family Resemblance Network for "game" following L. Wittgenstein and its analysis can be found here Stoenescu, Stoenescu, M. A. Family Resemblances: "game" Network Analysis. Zenodo Repository, 10 Dec. 2023, https://doi.org/10.5281/zeno-do.10340085.

nodes with high connectivity[2] represent uses that are more common and features that are generally considered more probable. Conversely, nodes with fewer connections can be deduced to be less frequent in usage. We can also observe the levels of relationships within this family by isolating relations for each node. We look at *virtuality in fiction* and identify on the map semantic traits such as *intentio lectoris*, *variation*, *time*, *choice*, *truth*, as well as the contrasting relation to *real*. Next, we see that each semantic feature is shared by other *uses* such as *text virtualization*, *VR, virtually possible.*[3]

Similar to the "game" analysis, our diagram illustrates the "family" concept which encapsulates the rejection of a universal semantic trait. For example, the term "virtual image" encompasses concepts such as *non-actuality* (A); *an image that appears to exist but does not* (B); *a specific phenomenon in optics* where an image is formed behind a mirror (C); *what is perceived visually* (D); and presents a contrast to what is *real* (E) and to *the actual state of affairs* (F), thereby constituting a feature set of {A, B, C, D, E, F}. In contrast, another usage such as *we are virtually there* conveys notions of *nearness to completion* (G), a *figurative sense* (H), *non-actuality* (A), *proximity* (I), resulting in the features {G, H, A, I}. Meanwhile, *virtual bank* implies a *digital* aspect (J) and involves *technology* (K), while still maintaining a contrast with the *real* (E), leading to the set {J, K, E}. The first and second sets share the concept of *non-actuality* (A), while the second and third sets share a contrast with the *real* (E). Although no single semantic trait is common across all three sets, we can detect one or more common features in some pairs. A key feature of the map is that it is dynamic and interactive; it allows for responsive exploration as hovering over a node highlights its immediate connections and dims the rest, thus facilitating an intuitive and informative navigation of the network.

[2] This frequency of use also influences the functioning of translation algorithms in machine translation tools.

[3] Snapshots of the different levels of the digram are available here: Stoenescu, M. A. Family Resemblance Network: Virtual - Levels 1 to 4. Zenodo, 14 Apr. 2024, https://doi.org/10.5281/zenodo.10970004.

As noted earlier, the dataset used to construct the Kumu map is derived from data obtained through meticulous examination and critical analysis of scientific literature on the subject. Specifically, the semantic traits (highlighted in blue) were manually gathered based on findings and descriptions from the research. While this method can produce a detailed map of the concept using selectively curated data, there's an argument to be made that such data may not genuinely reflect the authentic usage of the word. Consequently, we attempted to source a more objective dataset, aiming to capture a more organic linguistic application and a genuine representation of the concept. We believe that by examining the uses found in the results of the most dominant search engine, we can contend that acquiring a broader dataset in this way more accurately reflects the authentic application of the concept.

Tools for data retrieval (scraping), preprocessing, data analysis, and data visualization

Google API Integration: The optimization of search functionalities has become a pivotal concern in the contemporary digital landscape for researchers, scholars, developers, and engineers alike. As such, integrations with Google Cloud resources present an exceptional opportunity for scientific exploration and developing precise tools for scientific inquiry. Leveraging Google Cloud as an Infrastructure as a Service (IaaS) provides convenience and enhances precision, scalability, serverless computing capabilities, reliability, and access to comprehensive documentation for scientific projects. Without such a platform, scholars and developers would be constrained to construct and manage the required infrastructure themselves, a task that cannot match the services leading cloud providers offer.

The Google Cloud platform is accessible through the Google Cloud Console, where users can opt for more than 200 services and integrations covering a wide range of categories, including AI and ML (computing or conversational AI), containers, data analytics, databases, developer tools, networking, storage, and API management. For this project, we used the Google Search API to build a custom tool that uses the

Google search engine capabilities to retrieve results for specific queries. API stands for Application Programming Interface and consists of a set of protocols defining how different software applications can communicate with each other, e.g., data exchange, formats, and security. Generally, APIs provide a level of abstraction, hiding the underlying building blocks of a system. For that reason, they play a crucial role in software development by enabling developers and scholars to leverage existing services and functionalities to build new applications without programming everything from scratch. In our case, we used the Google Search API to integrate Google Search as a service in our script and tailor it to our requirements.

Setting up the access to the Google Search API requires following these steps. First, we create a new project in the Google Cloud Console. In the APIs & Services menu, we enable APIs & Services. We search for Custom Search API and enable it. Accessing the API through a command shell (or any script) requires a method of authentication, such as a personal API key. For that, we choose Credentials, Create Credentials, and API key from the menu. This unique key establishes the authenticated access to the API, ensuring a secure and authorized interaction with the service. For that reason, keeping it safe and secret is crucial. We stored it in a .txt file inside the project folder for later access.

With this, we have established the access to the Google Search service. The next step requires creating a Custom Search Engine (CSE)[4] that would perform the query task. For that, we follow the simple setup on programmablesearchengine.google.com. When creating a new CSE, a search engine ID is assigned. Like the API key, this ID is unique and should be kept safe and secret. We stored it in a new .txt file inside the project folder.

Building a Custom Scraper for search queries: When developing a software application, we typically rely on Jupyter Notebook. This widely recognized web application allows developers to create and share live code, equations, visualizations,

[4] "Custom Search JSON API." *Google Developers*, developers.google.com/custom-search/v1/introduction. Accessed 28 Sept. 2023.

and text. What sets it apart from other computing environments is its utilization of *cells*: distinct blocks of data that can be compiled independently. For example, developers can isolate a print function inside a separate cell, allowing them to fine-tune the desired output without having to recompile the entire script with every test. For that reason, it is particularly well-suited for data science and scientific computing projects, as well as for a wide range of Python projects. Its cell-based structure is not the only benefit of the Jupyter environment. Jupyter Notebook encourages users to include text explanations and documentation in their scripts. This is especially useful for data analysis and reporting, as well as for writing academic articles that include scripts.

For this project, we found the notebook's cell structure especially valuable due to the Google Search API limitation, which imposes a daily limit of 100 queries for the free plan. During the testing phase, without the ability to organize the query task in a separate cell, we would have quickly depleted our daily query limit each time we ran the script to test other functions. By isolating the query function, we ensured it was only executed when necessary.

Building a scraper for search queries Google Search API (libraries, key components, logic): The script (see Script 1 in Script Repository) extracts expressions containing "virtual," "virtually," "virtuality," or "virtualization" using the Google CSE through Google Search API[5], and saves the results, along with their context and links, into an Excel file for future process. The following is a detailed run through the script that justifies the choices and provides additional clarification. The main components are:

- API and CSE access
- search query setup
- search query task
- defining the regex pattern to extract results

[5] "Google Search API Overview." *Google Developers*, developers.google.com/custom-search/v1/overview. Accessed 25 Sept. 2023.

- listing the results
- output as .xlsx

We referenced three Python libraries: *requests* for the query function, *re* for the regex pattern, and *pandas* for data/file manipulation and storage. Next, we read the API key and the CSE ID from external files (API_KEY.txt and WittgensteinEngine_ID.txt) and stored them in two variables. This makes the approach more flexible (other search engines can be used by changing the ID) and more secure since sensitive data is not hardcoded into the script. Next, we set up the search query and the address to send the request. We performed many tests to optimize the search query considering the Google Advanced Search options (e.g., using operators like OR) as well as the limitations of the API, such as *pagination*[6] and the *number of results per page.* An example is the following search query: search_query = "virtual OR virtualization OR virtually OR virtuality." The script would send a query for the four words and generate a maximum of ten pages, with ten results per page, counting a total of 100 results per called function. This means that when using the OR operator, the 100 results are split for each of the four words. If we instead send a separate query for each of the four words, we get 100 results for each. For that reason, we chose to input the value of the query variable from the keyboard to get a more fine-tuned query setup. It could have also been done by using a *for* function and calling the query function for each of the words. Next, the query parameters are required for any query function.

[6] A note on pagination: Custom Google Engines return paginated results, which can be limiting when extracting larger datasets, not just the first results. To handle this limitation, the script sends multiple requests to the engine and aggregates results for each page. Unfortunately, this generates another issue regarding the consumption of the 100 daily queries credit that the free plan provides because when 'num' is set at 10 (pages), the script sends ten queries for each call. To better control the consumption for each run of the entire script, we chose to input the number of pages from the keyboard so that for simple tests, we would choose only one page, and for final runs, we would set it to the maximum value (10).

```
params = {
'q': search_query,          # The search_query variable[7]
'key': API_KEY, # Our private API_KEY
'cx': SEARCH_ENGINE_ID,
'start': 1,              # Start with the first result
'num': 10  # Number of results per page (max 10)
}
```

Regex patterns for raw data cleaning and preprocessing (challenges in the preprocessing phase): Next is the definition of the *regex pattern* we used to extract the desired expressions from the results. *Regex* stands for *regular expression*, a tool for pattern matching and text manipulation. Regular expressions are formed of a sequence of characters that define a search pattern that can be used for different operations such as searching, matching, extracting, validation, and replacement. The regex pattern we used is complex, but with the proper documentation[8] and enough testing, this is the pattern we finally created:

expression_pattern = re.compile(r'\b(\w+\s+virtual(?:ly|ity|ization)?)\s+(\w+(?:\s+\w+)?)\b', re.IGNORECASE)

This pattern captures expressions of the form: virtual + noun, virtually+word, virtuality+noun virtuality+in+noun, noun+virtualization, word+virtualization+noun, adj+virtualization, virtualization+of+noun. This regular expression is designed to match expressions of the form "virtual [word1] [word2]," where [word1] can be followed by optional suffixes "-ly," "-ity," or "-ization," and [word2] is another word. The capturing groups are used to extract [word1] and [word2] separately when a match is found.

[7] Here, the inline comments (marked with #) clarify the meaning of each variable. Moreover, the entire script includes comments that explain the purpose and functionality of different parts of the code, making it easier for others to understand and maintain the code.

[8] Tools such as https://regexr.com/, https://regex101.com/.

Octoparse integration for scraping Google Scholar (deployment and data retrieval): We used third-party software like Octoparse for our scraping tasks for two primary reasons. Firstly, third-party software solutions are often more advanced and robust compared to basic API calls for data extraction. In many scenarios, the challenge is not the API integration but processing the acquired data. Using third-party scrapers can often yield a richer dataset, which introduces intricate data-processing challenges worth exploring in a scientific paper. Secondly, upon examination, we found that Octoparse effectively addresses and overcomes certain limitations of the Google Search API and the Custom Search Engine, especially the 10-page limit and the restriction of 10 results per page. Interestingly, the Custom Search Engine interface mirrors Google's webpage design from a decade ago, featuring a top search bar, two tabs (web and image), an advertisement section, ten results, and a *pages* toolbar. In contrast, the current Google webpage has evolved significantly, now segmented into various sections such as dictionary, related search, "people also ask" section, and image section. Most notably, the traditional page toolbar has replaced the "More results" button. This evolution asks for a change in how scrapers interact with the webpage. We will look at Octoparse and highlight its pivotal features that enabled us to optimize data collection and bypass the restrictions associated with the API.

What follows is the documentation of the Octoparse experiment for scraping Google Scholar. There is an additional reason for introducing Octoparse here, which is the fact that there is no official Google Scholar API.

Octoparse is a web scraping tool that allows users to extract information from websites without writing code and building applications (scapers) from scratch. It is mainly designed for non-programmers and offers a visual interface where users can design their scraping tasks using predefined templates and setting up custom workflows. It is primarily used for data extraction of any kind (research, business intelligence, market analysis, news and content aggregation, etc.). Using a third-party tool like Octoparse helped us mitigate some of the limitations we have encountered with the Google API scraper. The advantages are many: no coding required, it has a visual workflow (users can see the data they are scraping in real-time so they can

fine-tune the tasks), no limitations for paginations or queries, the option to export raw data in xlsx, etc.

We also considered potential roadblocks and limitations when working with Octoparse (handling CAPTCHAs, limits, exports). Octoparse can be used in one of two ways: selecting a template or designing a custom task or workflow. The software does not allow you to download the data directly to your computer when using a template. Instead, it only offers the option to store it in its cloud storage, a feature that comes with a paid subscription. On the other hand, setting up a custom workflow is straightforward. It is similar to crafting a macro in MS Office. Essentially, you need to sequence the tasks a user would carry out during a Google search: launching the Google site in the browser, entering "virtual" in the search bar, hitting the *search* button or pressing enter, scrolling down, selecting the *More results* button, and so forth. Once established, this workflow acts as an "algorithm" that Octoparse runs automatically, capturing all the search results.

One particularly important aspect is that Octoparse does not use APIs, which raises the problem of handling CAPTCHAs. The reason for that is that many websites have security mechanisms that identify bots or invasive third-party software that access their data, such as web scrapers. CAPTCHA stands for "Completely Automated Public Turing Test to Tell Computers and Humans Apart." It is a system designed to determine whether the user is a human or a robot, and it is a prevalent method many websites use to combat scraping. While Octoparse does not handle it directly, there are a few ways to overcome this challenge.[9] The only method that proved effective for our project was using the *Browser mode*. This allows real-time observation of the scraper's interaction with the website, providing the option to pause the workflow and introduce user input directly into the site. Google is highly sensitive to third-party access and can swiftly detect when a scraper attempts to connect. Upon detection, Google triggers the reCAPTCHA system, often starting with the checkbox "I'm not a robot." Frequently, it follows this with another

[9] "How to Solve CAPTCHA in Octoparse." *Octoparse Help Center*, Octoparse, 2023, https://helpcenter.octoparse.com/en/articles/6471138-how-to-solve-captcha-in-octo-parse. Accessed 30 Sept. 2023.

CAPTCHA challenge: the Image CAPTCHA test. In this test, users are typically shown a prompt to select all images in a grid corresponding to a specific description, such as "Select all images with traffic lights." By temporarily pausing the scraper and addressing the CAPTCHA challenges, we successfully navigated past Google's protective measures.

After data capture, we implemented a series of filtering processes, applying the same regex pattern to both the title (the search result) and the snippet, using a subsequent Python script (refer to Script 2 below). Given its straightforward nature and similarity to the initial script, we will not focus on its specifics but will move to the project's next phase.

Matching *semantic features* with a *spaCy* model

The project's next phase involves linking the processed values to their corresponding semantic features. This step is likely the most challenging due to the size of the dataset and the complexity of matching uses to features. We have used various tools and experimented with several methods to automate this process.

ChatGPT 4 and Advanced Data Analysis Feature (Beta): GPT-4, the September 25 Version, incorporates four advanced tools that extend its functionality. The *Browse with Bing* feature enables real-time web searches for information retrieval; the *Advanced Data Analysis* feature enables complex data manipulation and analysis through Python code generation and execution; the *Plugins* feature enables different third-party plugins to enhance GPT-4, and the DALL·E tool, to generate images from textual descriptions (prompts) using a version of OpenAI's DALL·E image synthesis model.

The Advanced Data Analysis tool is one of the initial methods we have explored. The aim was to leverage the NLP and LLM capabilities of GPT-4 to analyze the set of semantic features and match them with the uses. This tool allows users to upload datasets (such as *xlsx* or *csv* formats) and specify the operations they want the AI to execute, or even describe the end result they aim for and let the AI handle the process. GPT-4 generates Python code in the background, which users can observe

in real-time. It executes this code within its own interpreter, debugging and modifying as needed to achieve the desired result. Once satisfied, users can instruct GPT-4 to save the refined dataset to a new file, which they can then download. The capability to upload, process, and download datasets circumvents the token limitations inherent in the prompting system, which is capped at 4,096 tokens for GPT-3.5 and 8,192 tokens for GPT-4. While these limits might appear very generous (approximately 32,000 characters per context window), processing large datasets directly within prompts leads GPT-4 to conserve space and computational resources. Consequently, it does not read or output entire datasets in full. To address the challenges posed by complex data processing and analysis tasks, the Advanced Data Analysis tool serves as an essential workaround. Nonetheless, this tool has a notable limitation. It performs the required operations by generating Python code and executing it, thus utilizing the LLM's capabilities solely for code writing (including debugging and optimization) but not within the code itself. For instance, if GPT-4 (in its default mode) is prompted to generate 20 semantic features for the word "virtual," it will use its LLM capabilities to do so effectively. However, when employing the Advanced Data Analysis tool for the same task, it will start generating the Python code and then execute it for this purpose. However, this code will rely on basic methods, such as generating sets of synonyms using basic models and libraries. If asked to perform a more complex linguistic analysis, it will commit to doing its best, yet it will still depend on basic modules and libraries. Admittedly, this tool can perform machine-learning tasks, which are inherently complex. But for our project, which aims to process a dataset that could later serve as a training model for a machine-learning algorithm, the lack of a pre-trained model to enhance our dataset means this is not a viable option. Having established that, we had to choose the best approach available, even if it meant leaving behind GPT's LLM capabilities, which would have been a powerful solution for our task.

WordNet similarity test: Our second approach involved writing an algorithm using the NLTK module, focusing on similarity calculations through WordNet's synsets, which are sets of synonyms sharing a common meaning. For each "use" of

"virtual" in the dataset, it calculates and compares its similarity to the *synset* of the semantic features (we predefined the semantic features set), recording the highest similarity score for each use-feature pair in the results. However, the script relies on WordNet's *synset* match for each word, which may not always represent the most contextually appropriate meaning, leading to potential inaccuracies in similarity scores. The reason is that semantic features are not synonyms but often incredibly nuanced meanings, specifically determined through use in a particular context. These sensible nuances may not even have a clear lexical expression, so trying to capture the semantic features with *synsets* is not the best approach. We used a similarity threshold of 0.25.[10] The total execution time was is around 15 minutes for our dataset (2000 rows and 150 columns approx.).

Similarity test with SpaCy word vectors: Shifting to spaCy, we utilized its word vector capabilities for semantic similarity computations. SpaCy's reliance on word vectors, which are multidimensional representations of words, allowed for capturing a broader array of semantic nuances. This approach provided a more nuanced understanding of context and word relationships than WordNet's path similarity. SpaCy also offered performance optimizations and faster text processing capabilities. Advanced features like part-of-speech tagging and entity recognition helped refine the process. We optimized the process by filtering out semantically less significant parts of speech, which reduced the computation time from about 90 seconds per iteration to 2-5 seconds. SpaCy's flexibility in optimizing functions for specific purposes proved beneficial in managing computation time. We used a strategy similar to the NLTK approach for measuring similarities, establishing a threshold to translate results into a binary format. The uniform distribution of values obtained with spaCy suggested a more precise and consistent evaluation of semantic relationships. After thorough testing, we set the optimal threshold at 0.5. Generally, spaCy is considered more accurate than WordNet when compared on summarization, and lexical richness calculation (Sharma, Aggarwal, and Alawadhi) and (Spring and Johnson). However,

[10] A higher threshold would result in fewer, more confident associations, while a lower threshold would include more associations but with less confidence.

in the case of semantical features as described by Wittgenstein, we note that "out-of-the-box" NLP libraries are still lower in accuracy compared to deep learning models using transformer architecture such as BERT or GPT.

Data visualization: final phase

Kumu[11] was our initial choice due to its user-friendly interface. Despite its advantages, like easy project sharing and customization, Kumu struggled with our project's scale, crashing even with reduced data (60% of 2000 nodes and 20,000 connections), compromising our goal to illustrate complex relationships in our study. Tableau,[12] known for its interactive dashboards and data analysis, was tested next. However, its limitation in plotting many-to-many node connections, crucial for network graphs, led us to rule it out. Finally, Gephi[13] proved to be the optimal choice for our project. As a leading open-source tool for graph and network visualization, it supports various file formats and offers powerful network analysis tools. Its high-performance rendering engine efficiently handles large networks, aligning with our needs for up to 50,000 nodes and 1,000,000 edges.

During the import step, Gephi requires (as any platform) a specific way of structuring the dataset: two dataframes — one for nodes and one for edges. Each node must have a unique ID, with optional tags for additional classification. We preprocessed our data, assigning an ID for each node and a type-tag (Use and Feature) to filter and format the uses and semantic features in the graph. The edges dataframe must list the connections between nodes on separate lines, with a 'Source' and 'Target' column containing the corresponding node IDs. Given that our plot comprises approximately 20,000 many-to-many connections, we had to represent

[11] "Documentation." *Kumu Help*, help.kumu.io/. Accessed 27 Sept. 2023.
[12] "Tableau Help." *Tableau Software*, help.tableau.com/current/pro/desktop/en-us/ default.htm. Accessed 25 Sept. 2023.
[13] "User Documentation." *Gephi - The Open Graph Viz Platform.* gephi.org/users/. Accessed 30 Sept. 2023.

each connection on a separate line in the edges dataframe. This resulted in multiple entries for individual nodes, totaling 19,586 lines.

After importing both dataframes into Gephi, we customized the graph for readability. Gephi is primarily built for network visualization and analysis; however, its default visualization features are not tailored to arranging nodes and edges in specific forms. Networks are typically intricate structures and often appear as free-floating forms, not regular shapes. Users can apply algorithms called *layouts*, such as Force Atlas, Fruchterman Reingold, or Yifan Hu to organize the elements or dictate their behavior. Visualizing the graph as two concentric circles (one for uses, one for features) seemed optimal to capture its essence. Since Gephi does not offer this type of layout by default, we integrated a plugin called Circular Layout, which provides three distinct layout algorithms: Circular Layout, Dual Circle Layout, and Radial Axis Layout. We primarily used the Dual Circle Layout to separate uses from features into two distinct concentric circles. Next, we customized the appearance of the nodes by filtering by the 'Type' tag (previously integrated into the dataset containing two values: use and feature) and selected two colors: orange for features and blue for uses. After enabling the visibility of the labels (the names of the nodes), we had to apply another set of algorithms because the nodes for uses were overlapping. We filtered out the features to display only the uses and applied the Expansion algorithm a few times, followed by the Noverlap algorithm and the Label Adjust algorithm. Finally, we plotted the graph at different resolutions and formats, yielding these results.

Figure 2 is the rendering of the entire network on a large scale.[14] The size of the feature nodes is proportional to their degree. The degree of a node is defined by its

[14] Due to this scale, individual nodes appear very small. Detailed images that offer a more zoomed-in view for better clarity are available here: Stoenescu, M. A. Total Family Resemblance Network: Virtual - 20000 Connections in Gephi. Zenodo, 14 Apr. 2024, https://doi.org/10.5281/zenodo.10970150. These perspectives enhance the readability of specific sections of the network and add depth to the representation, thereby illustrating the complexity and intricacies of the relationships within it. In particular, The view of the center closely resembles a weave or fabric due to the sheer volume of connections, creating an effect where the edges themselves seem indistinguishable, collectively forming a fabric-like structure. This representation invites multiple interpretations: it illustrates the dynamic and continually evolving nature of the concept, where new connections and features are perpetually integrated, further enriching the network; it signifies a high level of integration, suggesting that understanding one semantic feature likely leads to insights

number of connections, thus, nodes with a greater degree are represented as larger in size. This variation in size effectively embeds an additional dimension of information into the network's structure.

Conclusions

In our investigation into the concept of *virtual(ity)*, we have unearthed a complex network of semantic traits interconnected in a manner reminiscent of family resemblances. The nuanced variations of *virtual(ity)* across different contexts reveal a rich tapestry of meanings that go beyond a singular, definable essence.

Our study effectively harnesses the methodological framework of Wittgenstein's family resemblances concept to elucidate the semantic traits network surrounding *virtual(ity)*. We have confirmed Wittgenstein's argument that there is no universal common semantic trait across the entire set of meanings; however, many features such as *real*, *digital*, and *possible worlds* appear with notable frequency. The vast array of features underscores the multifaceted nature of the concept. Through a meticulous combination of scientific literature and organic data, we have constructed a more authentic linguistic representation of *virtual(ity)*. Integrating Google API, custom scraping techniques, and advanced data analysis using word vectors have been pivotal in this endeavor.

The research also thoroughly explored the application of various data visualization tools, with Gephi emerging as the preferred choice due to its robust handling of large datasets and complex many-to-many relationships. This visualization provides an aesthetic representation of the data and serves as an analytical tool to decipher the intricate semantic connections.

The broader implications of this research significantly extend beyond the semantic exploration of *virtual(ity)*. The methodologies and tools developed can be applied to other semantic studies, offering profound insights into the construction of

into others; and it implies that a segmented approach might not suffice for fully comprehending and analyzing this concept. Instead, a holistic view, considering the synergy and cumulative impact of various features, might be more effective in grasping the concept.

meaning in our increasingly digital world. Additionally, this research substantially contributes to computational linguistics and machine learning, providing a framework for semantic network analysis and the development of advanced translation algorithms.

Several open questions remain for future research. The dynamic nature of language and the evolution of *virtual(ity)* suggest that this network will continue to expand and shift. Further investigation into the impact of these shifts on translation tools and the exploration of additional semantic features as they emerge will be necessary. Moreover, the potential application of these findings in the development of artificial intelligence and machine learning models holds immense promise for future inquiry.

In conclusion, our comprehensive journey through the semantic landscape of *virtual(ity)* has demonstrated a complex web of meanings, decisively challenging the notion of a singular definition. The application of the *family resemblances* framework has not only been exceptionally effective in mapping this intricate terrain but also has revealed the dynamic and continually evolving nature of the concept advocating for a holistic view, one that embraces the synergy and cumulative impact of various features, to effectively grasp the full concept.

Moreover, the family resemblances framework has not only mapped the existing semantic landscape but also provided a robust methodological foundation for future explorations in computational linguistics and machine learning. The insights garnered from this study have major implications for these fields, offering a new lens through which the construction and evolution of complex concepts can be understood and applied.

As we progress in our research, we are committed to continuously refining our methodologies and analytical tools to represent the complexity of *virtual(ity)* more accurately. This research highlights the potential for significant advancements in understanding digital phenomena. The future presents numerous opportunities for further scholarly inquiry and methodological innovation in this field.

Accordingly, we posit that the exploration of *virtual(ity)* extends beyond mere academic inquiry; it is an investigation into a concept central to our increasingly digitalized world, with substantial implications for comprehending the foundational

concepts that shape our contemporary experience. The depth of our findings encourages further examination, and we invite the academic community to join us in this pursuit, contributing to a richer, more nuanced understanding of the virtual and its myriad dimensions.

Annex 1: Figures

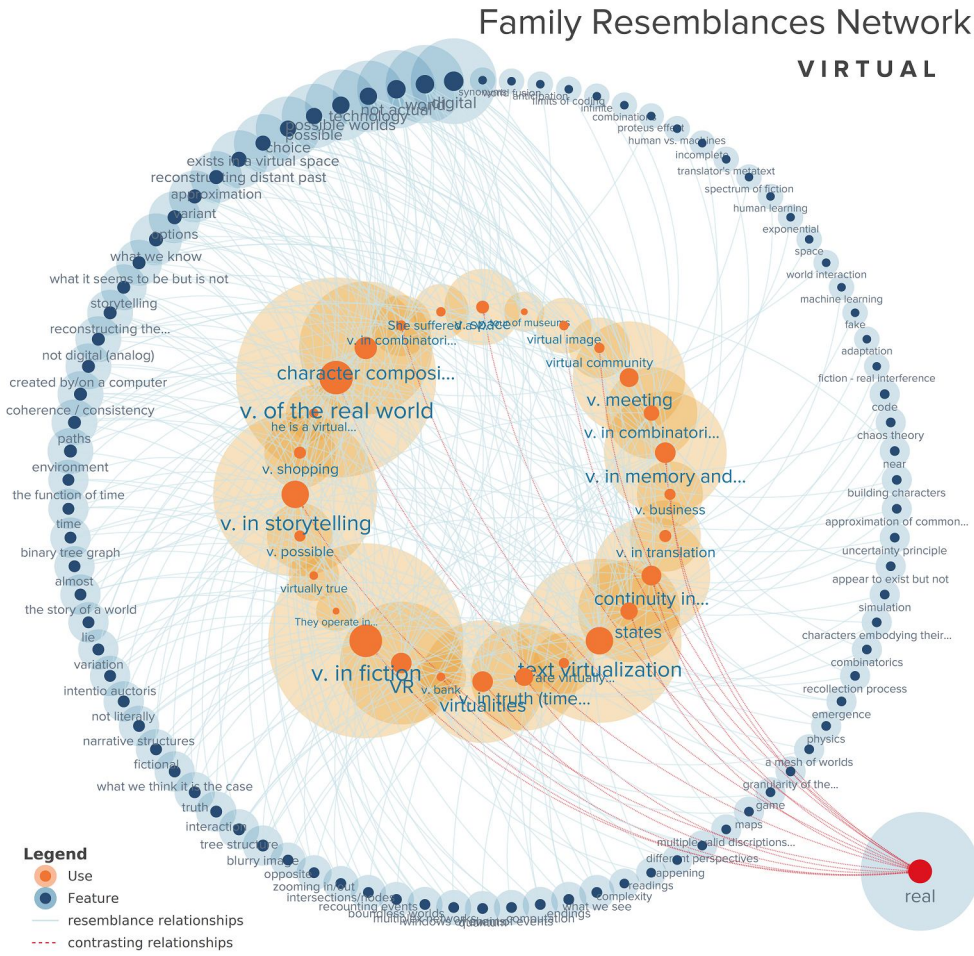

Figure 1: Family Resemblance Network: virtual.

The Dynamic model is available at https://embed.kumu.io/63c4468fe768675f5fc1960ffb6b87cf.

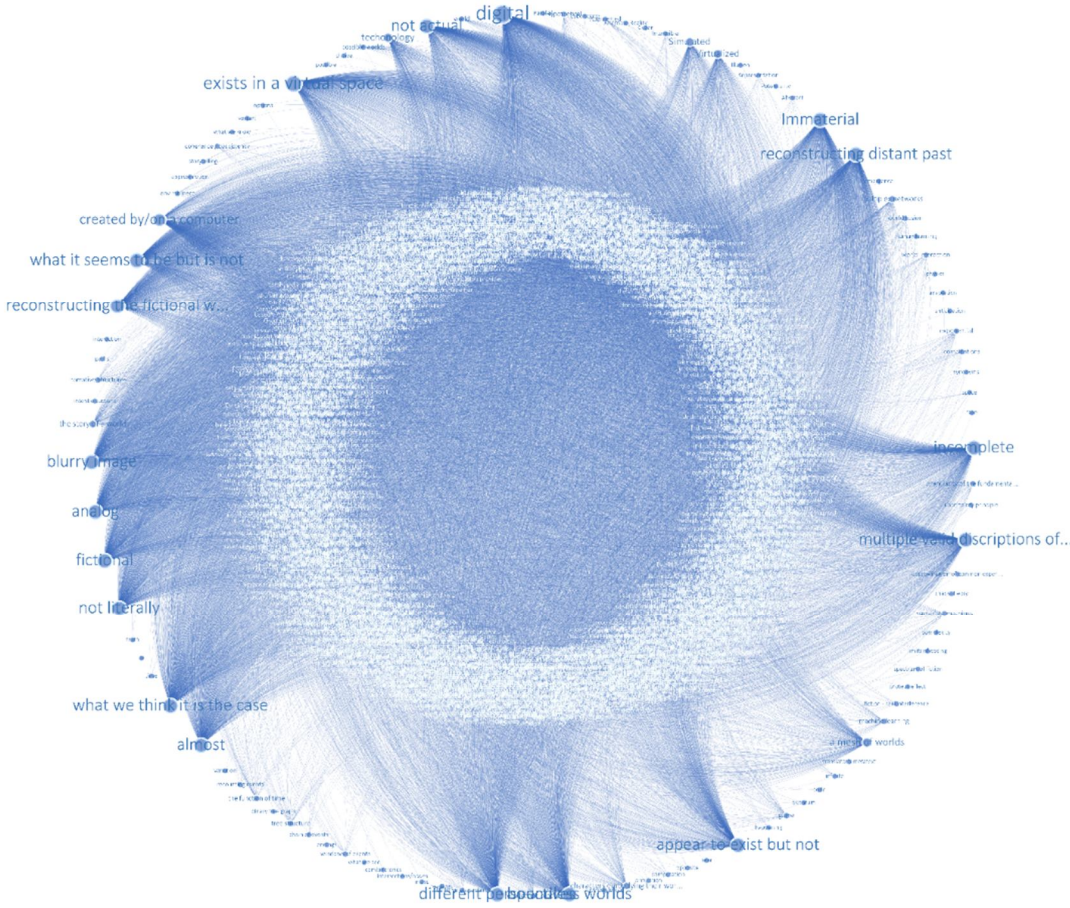

Figure 2: Total Family Resemblance Network – virtual – Gephi

Annex 2: Script Repository

Script 1

```
#This is the main script.
#It uses Google Search API integration through Custom Search Engine to retrieve data from Google Search.
#It writes one time (then overwrites) to an excel file the Expression (regex extracted) + Context + links
#for query = 'virtual OR virtually OR virtuality OR virtualization'

import requests
import re
import pandas as pd

#Read from file the API key and the Search Engine ID. Both are private keys and will not be disclose here.
API_KEY = open('API_KEY.txt').read()
SEARCH_ENGINE_ID = open('WittgensteinEngine_ID.txt').read()

#Set up a search query and the url to send the request to (our custom Google search engine)
# Take user input for the search query
search_query = input("Enter your search query: ")
base_url = "https://www.googleapis.com/customsearch/v1"

#Set up the parameters of the request as a dictionary of parameters' key' as  and 'cx' is the ID of our Search Engine
params = {
```

```
    'q': search_query, # The search_query variable
    'key': API_KEY, # Our private API_KEY
    'cx': SEARCH_ENGINE_ID,
    'start': 1, # Start with the first result
    'num': 10 # Number of results per page (max 10)
}

# Initialize lists to store extracted expressions and their corresponding links
all_results = []
extracted_expressions = []
extracted_links = []

# Take user input for the number of pages to retrieve
num_pages = int(input("Enter the number of pages to retrieve: "))

# Send the requests to the url with the parameters that we defined for multiple pages
for page in range(num_pages):
    response = requests.get(base_url, params=params)
    results = response.json().get('items', [])
    # Add results from this page to the list of all results
    all_results.extend(results)

    # Increment the 'start' parameter to get the next page of results
    params['start'] += 10

# Define a regex pattern to extract expressions of the form "virtual [noun]"
expression_pattern =
re.compile(r'\b(\w+\s+virtual(?:ly|ity|ization)?)\s+(\w+(?:\s+\w+)?)\b', re.IGNORECASE)

# Initialize lists to store extracted expressions, their corresponding contexts
(snippets), and links
extracted_expressions = []
extracted_contexts = []
```

```
    extracted_links = []

    # Extract expressions from titles, snippets, and links
    for item in all_results:
        title = item['title']
        snippet = item.get('snippet', '')
        link = item.get('link', '')

        # Search for the expressions in the title and snippet
        title_matches = expression_pattern.findall(title)
        snippet_matches = expression_pattern.findall(snippet)

        # Add the matches to the list of extracted expressions, contexts, and their
corresponding links
        for expression in title_matches:
            extracted_expressions.append(' '.join(expression))
            extracted_contexts.append(snippet)
            extracted_links.append(link)
        for expression in snippet_matches:
            extracted_expressions.append(' '.join(expression))
            extracted_contexts.append(snippet)
            extracted_links.append(link)

    # Create a DataFrame with columns for expressions, contexts, and links
    df = pd.DataFrame({'Expressions': extracted_expressions, 'Context':
extracted_contexts, 'Links': extracted_links})

    # Save the DataFrame to an Excel file
    df.to_excel('virtualization_extraction1.xlsx', index=False)
```

Script 2

#This Script reads the excel file containing the raw data extracted from Google Scholar, extracts with Regex the desired data from the Title and the Snippet and stores them in a new excel file.

```
import pandas as pd
import re

# Load the original Excel file into a DataFrame
original_file_path = 'GoogleScholarExtraction-raw_work.xlsx'
original_df = pd.read_excel(original_file_path)

# Extract the first and second columns into separate lists
column1_values = original_df.iloc[:, 0].tolist()
column2_values = original_df.iloc[:, 1].tolist()

# Initialize lists to store the extracted data and snippets
extracted_data = []
snippets = []

# Define the regex pattern
pattern = r'\b(\w+\s+virtual(?:ly|ity|ization)?)\s+(\w+(?:\s+\w+)?)\b'

# Iterate through the lists and extract data using regex
for value1, value2, snippet in zip(column1_values, column2_values,
original_df['snippet']):
    match1 = re.search(pattern, str(value1))
    match2 = re.search(pattern, str(value2))

    if match1:
        extracted_data.append(match1.group(0))
        snippets.append(snippet)
```

```
        if match2:
            extracted_data.append(match2.group(0))
            snippets.append(snippet)

    # Create a new DataFrame with the extracted data and snippets
    new_df = pd.DataFrame({'Extracted_Data': extracted_data, 'Snippet': snippets})

    # Save the new DataFrame to a new Excel file
    new_file_path = 'ExtractedDataWithSnippet.xlsx'
    new_df.to_excel(new_file_path, index=False)

    print(f"Extracted data with 'snippet' column added saved to '{new_file_path}'")
```

Extracted data with 'snippet' column added saved to 'ExtractedDataWithSnippet.xlsx'

Script 3

```
# This script uses spaCy module to match uses to semantic features.
# Step 1: Setup
import spacy
import pandas as pd
import numpy as np

# Load the pre-trained spaCy model
nlp = spacy.load("en_core_web_md")

# Load the dataset
df = pd.read_excel("ExtractedDataWithSnippet_work.xlsx")

# Adjust the tokenize function to return SpaCy tokens instead of strings
def tokenize(text):
```

```
        return nlp(text)

    # Compute similarity only if tokens have vectors and are not stop words
    def compute_similarity(token1, token2):
        if token1.has_vector and token2.has_vector and not (token1.is_stop or
token2.is_stop):
            return token1.similarity(token2)
        return 0

    # Function to process each row and calculate similarity, excluding prepositions and
conjunctions
    def process_row(row, features):
        row_data = []
        tokens = [token for token in tokenize(row['Label']) if token.pos_ not in ('ADP',
'CONJ', 'CCONJ')]
        for feature in features:
            feature_tokens = [token for token in tokenize(feature) if token.pos_ not in
('ADP', 'CONJ', 'CCONJ')]
            max_similarity = 0
            for token1 in tokens:
                for token2 in feature_tokens:
                    max_similarity = max(max_similarity, compute_similarity(token1,
token2))
            row_data.append(max_similarity)
        return row_data

    # Define the features to compute similarity
    features = df.columns[2:-1]  # Excluding the last column

    # Starting similarity computation
    print("Starting similarity computation for each use-feature pair...")
    results = []
    for index, row in df.iterrows():
        row_data = process_row(row, features)
```

```
            results.append(row_data)
            # Print progress
            print(f"Processed {index + 1} of {len(df)} rows...", end='\r')

    print("\nFinished similarity computation for the loaded rows.")

    # Step 4: Thresholding
    threshold = 0.5
    true_matrix = (np.array(results) > threshold).astype(int)

    # Create a new DataFrame from the true_matrix to avoid fragmentation
    true_df = pd.DataFrame(true_matrix, columns=features)

    # Concatenate the new DataFrame with the original one, this will create non-
fragmented DataFrame
    df = pd.concat([df.iloc[:, :2], true_df], axis=1)

    # Calculate the total associations and add as a new column
    df['Total_Associations'] = true_df.sum(axis=1)

    # Save/display results
    df.to_excel("updated_data_spacy_threshold_0_5_first_allows.xlsx", index=False)
    print(df)
```

References

Aarseth, Espen. 2001. "Virtual Worlds, Real Knowledge: Towards a Hermeneutics of Virtuality." *European Review* 9 (2): 227–232.

Bartle, Richard Allan. 2004. *Designing Virtual Worlds*. Indianapolis: New Riders Publishing.

Belanger, Wayne D. 2009. *A Semiotic Analysis of Virtual Reality*. UMI Microform, PhD dissertation.

Chan, Melanie. 2014. *Virtual Reality: Representations in Contemporary Media*. London-New York: Bloomsbury.

Heim, Michael. 1993. *The Metaphysics of Virtual Reality*. Oxford: Oxford University Press.

Heim, Michael. 1998. *Virtual Realism*. Oxford: Oxford University Press.

Lewis, David. 1986. *On the Plurality of Worlds*. Oxford: Blackwell.

Ryan, Marie-Laure, and Jan-Noël Thon. 2014. *Storyworlds Across Media: Toward a Media-Conscious Narratology*. Lincoln-Nebraska: University of Nebraska Press.

Ryan, Marie-Laure, and Alice Bell (eds). 2019. "From Possible Worlds to Storyworlds: On the Worldness of Narrative Representation." In *Possible Worlds Theory and Contemporary Narratology*, edited by Alice Bell and Marie-Laure Ryan. Lincoln-Nebraska: University of Nebraska Press, 62–87. Accessed January 14, 2021. JSTOR, www.jstor.org/stable/j.ctv8xng0c.7.

Wittgenstein, Ludwig. 1953. *Philosophical Investigations*. Translated by G.E.M. Anscombe. New York-London: Macmillan.

Google Developers. 2023. "Custom Search JSON API." Accessed September 28. developers.google.com/custom-search/v1/introduction.

Google Developers. 2023. "Google Search API Overview." Accessed September 25. developers.google.com/custom-search/v1/overview.

Octoparse. 2023. "How to Solve CAPTCHA in Octoparse." *Octoparse Help Center*. Accessed November 1. helpcenter.octoparse.com/en/articles/6471138-how-to-solve-captcha-in-octoparse.

Sharma, Abhilasha, Raghav Aggarwal, and Raghav Alawadhi. 2023. "A Comparative Study of Text Summarization using Gensim, NLTK, Spacy, and Sumy Libraries." *Journal of Xi'an Shiyou University, Natural Science Edition* 19 (4).

spaCy Documentation. 2023. "spaCy 101: Everything you need to know." Accessed September 30. spacy.io/usage/spacy-101.

Spring, R., and M. Johnson. 2022. "The Possibility of Improving Automated Calculation of Measures of Lexical Richness for EFL Writing: A Comparison of the LCA, NLTK and SpaCy Tools." *System* 106. https://doi.org/10.1016/J.SYSTEM.2022.102770.

Tableau Software. 2023. "Tableau Help." Accessed September 25. help.tableau.com/current/pro/desktop/en-us/default.htm.

Gephi. 2023. “User Documentation.” *Gephi - The Open Graph Viz Platform*. Accessed September 30. gephi.org/users/.
Kumu Help. 2023. “Documentation.” *Kumu Help*. Accessed September 27. help.kumu.io/.