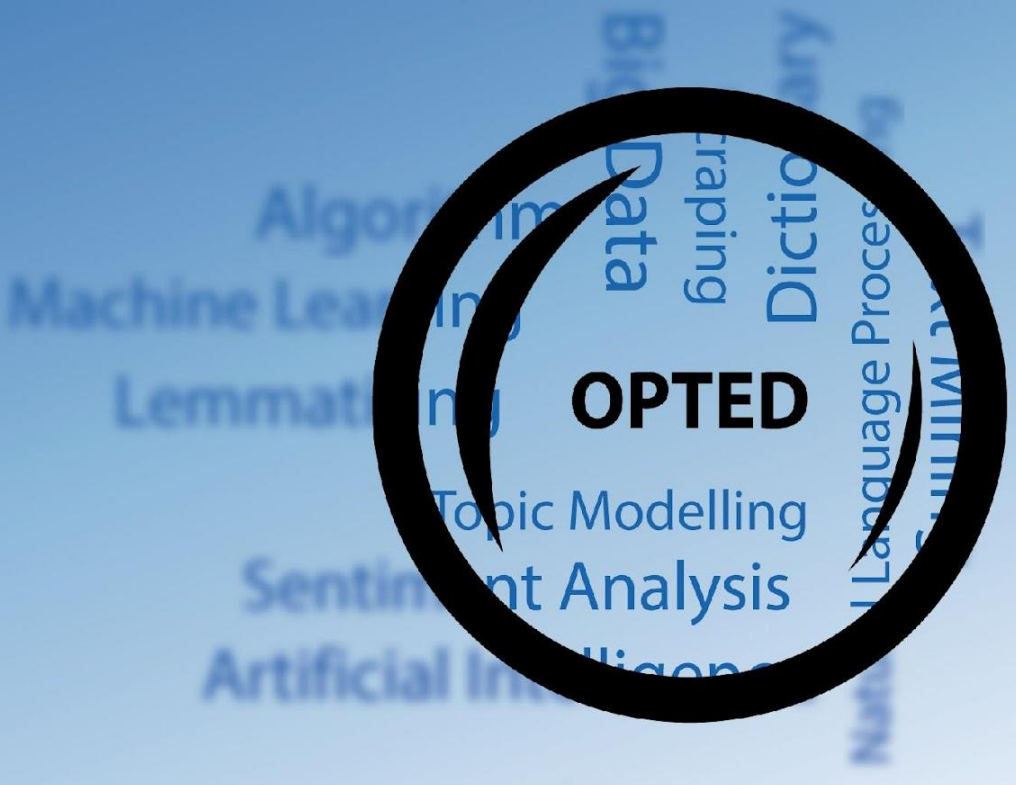
OPTED

Deliverable D5.6

Christian Rauh, Péter Gelányi, Lukas Hetzer, Sven-Oliver Proksch, Jan Schwalbach, Miklós Sebők



Disclaimer

This project has received funding from the European Union's Horizon 2020 research & innovation programme under grant agreement No 951832. The document reflects only the authors' views. The European Union is not liable for any use that may be made of the information contained herein.

Dissemination level

PU

Type

DEC/R

OPTED
Observatory for Political Texts in European Democracies:
A European research infrastructure

ParlLawSpeech Public Access Website
Deliverable D5.6

OPTED WP5 team

Christian Rauh[1/2], Péter Gelányi[3], Lukas Hetzer[4], Sven-Oliver Proksch[4], Jan Schwalbach[4/5], Miklós Sebők[3]

[1] WZB Berlin Social Science Center
[2] University of Potsdam
[3] Institute for Political Science, Centre for Social Sciences, Budapest
[4] University of Cologne
[5] GESIS Leibniz Institute for the Social Sciences

Due date: September 2023

Table of Contents

Purpose

The project **OPTED: Observatory for Political Texts in European Democracies** (Horizon 2020 Grant agreement 951832) outlines a European Research Infrastructure facilitating the large-scale computational analysis of political texts in Europe. **Work Package 5** focuses on texts produced in the decision-making processes of national and supranational parliaments.

Objective 6 of the OPTED initiative highlights that the wealth of systematic information in large collections of political text should not only be available to experienced researchers and analysts. Rather, a key purpose of the future platform is to also provide an *entry point for laymen* – having especially *journalists*, *political practitioners*, and not the least *interested European citizens* in mind. One may think, for example, of a citizen who is interested in which party spoke about a certain topic and how. One may think about experts trying to figure out whether certain policies are adequately represented in political discourse. Or one may also think about a journalist who looks for structured information on how politicians have positioned themselves on specific policies over time to contextualize a contemporary debate. The key point is that these target audiences outside of academia could gain enhanced insights on the functioning of contemporary European democracies if they get access to the initially unstructured information hidden in large text corpora.

However, for extracting this information laymen audiences face notable *barriers of entry*. While various OPTED work packages have generated relevant text data collections (as we did for parliamentary speeches, bills, and laws in Deliverable 5.2 [**link**]) and provide analysis tutorials based on open-source software (as we did for legislative debates in Deliverable D5.3 [**link**]), information extraction still requires handling very large datasets, access to relevant software environments, and advanced programming skills. This can quickly set off laymen audiences who potentially just want to extract basic information (e.g. the prominence of certain keywords in political speeches), who require only specific subsets of larger collections (e.g. legal texts mentioning certain keywords for qualitative analysis), or who want to quickly assess a working hypothesis for deciding on whether to invest in advanced analysis. Seen from this angle, researchers collecting large-scale political text corpora should ideally also provide low-key access tools for non-technical audiences.

Yet, *also the researchers face notable barriers of entry* in this regard. Investing time in public access tools is not highly rewarded in academic promotion logics which mostly revolve around scientific publications. Moreover, the scarce resources in research projects can often not be allocated to buying additional proprietary software or to hiring dedicated application developers. And while modern

political scientists are usually well trained in analytical programming, they rarely control advanced software or web development skills. For these reasons, the added societal value of low-key public access to scientific collections of political texts often remains unearthed.

With the purpose to lower such entry barriers for both data users and data providers, this deliverable thus develops proof-of-concept prototypes for public access websites that (a) allow non-technical users to extract systematic information from large text data and that (b) can be built with a reasonable investment of time and resources for academic researchers.

As an *exemplary case study* we use our own *ParlLawSpeech* text collection (Deliverable 5.2 [**link**]) and build such prototypes for providing *low-barrier access to the full texts of 3,172,026 plenary speeches* from eight European parliaments contained therein.

In the following pages we outline the key design principles that guide our approach (Section 2), briefly sketch the development and functionality of three consecutive prototypes with increasing levels of technical sophistication (Sections 3-5) and provide links to the resulting online apps as well as to their source code (Section 6).

In conclusion our tools and experiences suggest that building low-key public access tools can be achieved by researchers at reasonable levels of investment. But we also note that *in-depth exchange with technical experts inside and outside the OPTED initiative* proved highly beneficial, while a *standing server infrastructure is needed to provide such access tools to the wider public*. This highlights *two key services that a future platform for political text analysis in Europe could provide*.

Key design principles

To address the dual purpose of lowering entry barriers for both potential data users and data providers, we initially set up a list of basic design and functionality principles that informed our subsequent development steps.

With a view to laymen audiences, we consider the following principles particularly important:

- **No prerequisites regarding computer skills or software:** Potential users should be readily able to interact with the tools without having to acquire additional skills, to read long explanations, or

to install additional software. Specifically, we aimed at *tools that work in any conventional web browser*.

- **Interactivity:** To engage laymen users, the tools should allow them to 'play' with the data along whatever is of interest to them. And to maintain their interest, they should receive quick and quickly comprehensible results. This suggests that the tools should be *interactive*, *responsive*, and provide *visual information*.

- **Optional user flexibility:** Beyond quick results, users wishing to dig deeper should also have additional degrees of freedom – either in terms of accessing the raw texts behind aggregated numbers or in terms of alternative presentations of the extracted information. This suggests that the tools should *offer download options for raw and aggregated data*.

- **Low-key text analyses first:** This principle is derived from the three preceding ones. While advanced methods such as topic models or semantic scaling, for example, can and do extract highly relevant information from political text collections, they often build on additional assumptions or knowledge about text classification algorithms that may set off laymen audiences or provide a wrong sense of certainty. We thus rather first focus on *simple keyword or phrase extraction tools that are aggregated along well-known categories* (in our case of parliamentary speeches time, parties, and speakers). At the same time, we aim to *provide links to expanded data descriptions and/or tutorials* for advanced analyses and *allow raw data downloads* for those users wishing to go further.[1]

Implementing these principles quickly amounts to notable challenges in terms of technical skills and costs for academics generating relevant text collections. With a view to these data providers, we thus consider particularly the following two principles important for developing public access tools:

- **Open-source software:** For academics open-source software has three key advantages. First, it is usually available at no or low costs thus creating fewer or no additional financial burdens for scarcely funded research projects. Second, open-source software is usually accompanied by helpful online communities that offer help for specific technical challenges and customization of

[1] Such download options are, of course, desirable from a user perspective and for replicable research but can be limited by copyright restrictions of the original text data, especially if they come from proprietary sources or non-public institutions and organizations. For further insights on intermediate solutions under the heading of "non-consumptive research" in such instances, the reader may refer to the work in OPTED work package 7.

respective solutions. Third, open-source software allows for sharing the source code of particular applications (as we do with this Deliverable, see below) thus providing examples and lowering the entry barriers for others. Thus, we think that using and developing open-source tools *maximizes the proliferation of public access tools* among the academic community.

- **Close to conventional analytic programming environments:** Academics collecting large scale text data are usually well versed in analytic programming environments. The closer the development of public access tools stays to such environments (in terms of syntax for handling text data as well as extracting, aggregating, and visualizing information), the lower are their barriers of entry. As social science scholars employing text analysis tend to use *R* (a software for statistical programming) or *Python* (a programming language), the development of public access tools should also focus on these programming environments.

With these key considerations in mind, we started to develop prototypes around our own text data – thereby perfectly emulating academics with little or no prior expertise in web development.

First set of prototypes: Self-contained Shiny apps

Why Shiny?

When looking for software that meets the above specified principles for potential data users and academic data providers alike, our options quickly converged on Shiny.[2] This framework ticks virtually all boxes with regard to our initial considerations. Shiny is a web application framework that has developed in the R ecosystem but, since its most recent installment, also handles Python.[3]

At its core, Shiny bridges the gap between data analysis in R or Python on the one hand, and *web-based interactivity* on the other. It allows data providers to swiftly transform elements of the analytic scripts that they write in their daily scientific work into visually engaging web applications without the necessity of a particularly deep expertise in web development languages such as HTML, CSS, or JavaScript (even though such knowledge does not hurt). Beyond low entry barriers for academics and

[2] https://shiny.posit.co/ (last accessed: August 30, 2023)

[3] As our own ParlLawSpeech data collection has been conducted in R we stuck with this programming environment also for the prototype development summarized here.

high user interactivity, Shiny is furthermore geared to generating dashboards with *textual, tabular or visual data presentation*, all powered in the background by any of the respective packages and libraries that the open-source R or Python communities have to offer.

Shiny code functions through a dichotomy of a user interface (UI) and a server logic. The UI outlines the visual layout of the application, detailing elements like input controls, plot areas, or text. The Server logic, on the other side, defines and controls the computational reactions to user inputs, harnessing *the power of R to dynamically process data and relay results*. As laymen users interact with the application, Shiny – if set up appropriately – handles the communication between the UI and server, providing *real-time updates* and rather fluid *responsiveness*.

By using Shiny, *academics can largely stay within the syntax that they are used* to from data analysis projects, especially with regard to the server logic of any given app. However, it has to be noted that especially the reactivity of the UI and the server logic adds a notable layer of complexity while more advanced UIs require some engagement with basic elements of web technologies. It helps also in this regard, that Shiny is also *open source* meaning that a large community of developers,[4] helpful online fora,[5] and numerous example applications[6] or style sheets[7] can be relatively easily accessed.

For these reasons, Shiny appears optimal for our purposes and by sharing the source code we hope to contribute and encourage others to improve, expand, and build on our initial ideas for using the Shiny framework in granting public access to large text data collections.

However, one structural barrier remains. Beyond development, presenting a Shiny app to a broader audience necessitates dealing with the issue of *hosting*. Shiny apps can be locally built (and potentially also included in broader data releases for easing the access of users on their own computers) but to reach an expansive audience, they require deployment on platforms equipped to handle external web traffic while providing an R or Python environment on the backend. Useful paid

[4] See, for example, https://github.com/topics/shiny-app (last accessed August 31, 2023)

[5] See, e.g., https://stackoverflow.com/questions/tagged/shiny (last accessed August 31, 2023)

[6] See, e.g., https://shiny.posit.co/r/gallery/ (last accessed August 31, 2023)

[7] Cascading Style Sheets (CSS) provide a language used for defining the look and formatting of HTML documents. For our applications we manually adapted the Cerulean Sheme (https://bootswatch.com/cerulean/) extracted from the shinythemes package (https://rstudio.github.io/shinythemes/) to include OPTED colors and logos, for example. The theme is provided in our source code as well.

services (with helpful trial options) exist,[8] but with the additionally accruing cost they are often not a good solution for providing long-term data access by academic researchers who usually work in fixed-duration projects. Importantly, Posit also offers a *free and open source Shiny Server environment*[9] that can be used in establishing an academically driven hosting platform. However, setting up such a platform still requires public-facing server hardware and technically skilled staff to set up and to maintain the server and its analytic backends – resources that not all universities or research institutes can easily muster by themselves.[10] ***Especially in this regard, we think, a future European text analysis platform could provide much added value and benefits of scale for the academic community and the laymen audiences we want to reach.***

Key functionality of the two prototype apps

Having decided on a software and server framework, we then started to translate the above specified design principles into first operational Shiny applications. As these first development steps had to be taken in parallel to the ongoing ParlLawSpeech (PLS) data collection of WP5, we initially worked with two corpora from ParlSpeech, our prior collection of parliamentary speeches.[11] Considering the actual functionality with a view to the above principles, we initially opted to build *two separate public access applications*.

The first one – ***PLS-words*** – allows laymen users to (rather) quickly analyze the prominence of specific keywords and phrases across very large numbers of parliamentary speeches. We present detail on the user interface in section 4 below, but once a user has chosen the parliament and keyword

[8] One prominent option is https://www.shinyapps.io/ (last accessed: August 31, 2023). Shinyapps.io also offers a free tier which is very useful and convenient for deploying and testing apps but it is limited in the number of applications and, importantly, their run time. Other paid services include cloud platforms like AWS or Google Cloud which also offer flexible hosting opportunities, often leveraging virtual machines or Docker containers.

[9] https://posit.co/download/shiny-server/ (last accessed: August 31, 2023)

[10] In this regard our work here profited from the generous support of the project partner WZB Berlin Social Science Center (in particular by Sabrina Milewsky) in setting up a respective environment.

[11] Rauh, Christian; Schwalbach, Jan, 2020, "The ParlSpeech V2 data set: Full-text corpora of 6.3 million parliamentary speeches in the key legislative chambers of nine representative democracies", https://doi.org/10.7910/DVN/L4OAKN , Harvard Dataverse, V1

(combinations) of interest to her or him, the R server backend of the app loads the respective corpus, filters the respective speeches along keyword/phrase presence, and aggregates the resulting data across time (months), political parties (as organized in the respective parliament), and individuals speakers (members of parliaments and other persons with plenary speaking time).[12] The user interface then returns partially customizable and downloadable visualizations while also offering download options for the aggregated data (for maximum flexibility in comma-separated ASCII files). In sum, *PLS-words allows non-technical users to quickly analyze the prominence of self-chosen keywords in self-chosen parliaments over time, political parties, and individual speakers*.

For users wishing to look behind these aggregated numbers (e.g., by reading respective speeches or subjecting them to more advanced text analyses), a second app – ***PLS-extract*** – offers customizable *full-text downloads based on user choices* on the parliament of interest and optional filters regarding keyword presence in speech, date of speech, as well as party or name of speaker (for see user interface see section 4 below). Based on these choices, the server backend of the app filters the larger corpora and offers the resulting data in one of three user-chosen formats. Specifically, we offer *tab-separated ASCII files* to be used with any kind of spreadsheet software, the binary *.rds format to be used with R*, as well as *.feather files*, a fast and highly efficient data format that can be used with Python, amongst others. In sum, *PLS-extract facilitates data access by freeing users interested in specific speech collections from downloading very large data dumps and filtering them locally with advanced tools.*

Notably, we designed both apps initially to be **self-contained** in the sense that they *solely rely on the R environment* for frontend and backend while the apps also *include the raw text corpora* themselves. This has advantages especially for academic data providers as it allows them to stay within an environment that they use for their own analysis anyway while retaining full control over data management without the need to rely on external technical support in this regard (apart from the need to set up a hosting server, see above).

[12] Our R backend relied particularly on the tidyverse libraries (https://www.tidyverse.org/) for data management, and on quanteda (http://quanteda.io/) for the text analysis steps.

Downsides of the self-contained solution

However, we also quickly learned that such a self-contained solution has limits especially when it comes to very large text collections. On the one hand, including the data itself requires lots of memory on the public-facing Shiny server which creates *additional burden with regard to hardware requirements*. On the other hand, it may run against the principles of user interactivity and responsiveness of the application as *data loading and especially text filtering solely through an R backend becomes comparatively slow*.[13] While the respective R tools maximize researcher degrees of freedom and while waiting times of up to a couple of minutes are reasonable in a local data analysis project, such dead times quickly alienate users from the wider public which is nowadays used to a fast and responsive online environment.

Yet and still, we think that our initial self-contained proof-of-concept prototypes may be useful starting points for the community and are useful for customizing them to smaller text collections. Links to the operational web applications and their source code is thus provided in Section 6 below.

Prototypes 2: Shiny apps with AMCAT backend

In pushing our apps further to better user experiences with regard to interactivity and responsiveness, especially the *second annual meeting of the OPTED initiative* during September 2022 in Amsterdam proved extremely helpful. Here we learned in particular about the work of WP7 [**link**] on pre-processing, storage and data sharing tools and the corresponding AMCAT infrastructure that was further developed to that end.[14]

Why AMCAT?

While the AMCAT environment offers various useful tools for researchers with regard to data annotation and also provides specific approaches to generate dashboards and basic visualizations,

[13] We have tested different file formats and reading procedures as well as different text indexing approaches to optimize speed, but could not push user waiting time below 10 seconds when also allowing for phrase search in a corpus of around 1 million parliamentary speeches. Some of these benchmarks are included in the public code repository of the apps.

[14] https://amcat.nl/ (last accessed August 31, 2023)

especially the *server backend* attracted our attention. Four features stand out with a view to the purpose of this deliverable.

First, the storage of text databases in AMCAT builds on Elasticsearch,[15] a search engine that indexes texts very efficiently and offers a reasonably well understandable query language. To us, AMCAT thus offers *improvements with regard to generating quick search results across large text corpora and enhanced user functionality*. Second, AMCAT servers can be addressed through an automated programming interface (API). For us this means that we *do not have to store the text data locally* with the applications themselves and can rather send requests to and receive results from an AMCAT server elsewhere. Third, Johannes Gruber form Work Package 7 has developed an R package that binds our preferred programming environment to the AMCAT API.[16] This allows us to *address AMCAT servers from within our extant R backend* of the apps. Fourth and finally, AMCAT is open-source. To us, this renders reliance on the respective servers *sustainable* as the servers can be at least setup elsewhere free of monetary cost if infrastructural conditions in individual universities or research institutes change.

We thus decided to use AMCAT as a backend for the PLS-words and PLS-extract apps. Johannes Gruber (WP7) and Paul Baluff (WP3) kindly set up respective AMCAT servers in Amsterdam and Vienna and allowed us to upload our large *ParlLawSpeech* data that we had collected in the meantime. We then re-programmed our apps on the basis of these servers, gaining marked improvements in terms of speed, responsiveness, and usability for non-technical audiences.

As a side product we could also offer useful feedback for software development by the AMCAT and Vienna teams while generating examples, such as the upload script in the source code below, that in the future may assist other data providers interested in linking AMCAT and R projects. **This points to another key added-value that a future platform of political text analysis in Europe could offer: regular exchange among the relevant community offers highly productive mutual learning that not only promotes academic cooperation but also facilitates public access to scholarly data collections.**

[15] https://en.wikipedia.org/wiki/Elasticsearch (last accessed August 31, 2023)

[16] https://github.com/ccs-amsterdam/amcat4r (last accessed August 31, 2023)

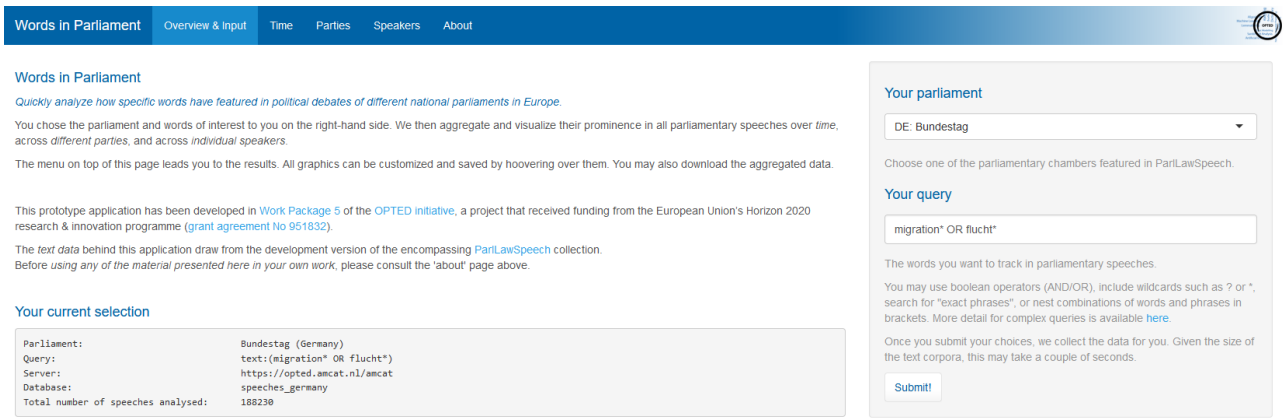
Key functionality – PLS-words and PLS-extract in action

We now turn to the user interface of the thus created applications. The home screen of the PLS-words app welcomes users with a very brief introduction on what is on offer (while providing direct web links to further background information). It furthermore contains two entry boxes in which users can enter their choices regarding the parliament and keywords of interest to them.

Currently the app features the development version of ParlLawSpeech (which is currently validated and prepared for public release) and thus includes 3,172,026 speeches from the major parliamentary chambers in seven European states (Austria, Croatia, Czech Republic, Denmark, Germany, and Hungary) as well as the European Parliament – a total of almost six gigabytes of text data and corresponding meta information.

Once the user has submitted her or his choices, a small info box returns information on the text collection that has been searched. Figure 1 provides an example in which the user wants to learn about the prominence of migration related terms in the German Bundestag.

FIGURE 1: HOME SCREEN OF THE PLS-WORDS APP WITH EXEMPLARY USER CHOICES



The header of the app then allows the user to jump to the results aggregated over time, political parties, or speakers. This is the point in time when the data is actually called and aggregated from the AMCAT server in the backend and some few seconds of waiting time may occur about which the user is informed in a friendly modal dialogue.

Each of the three results panes then offers a *visual summary* of the key patterns the user request has produced. When hovering over the plot, users get some customization and can download the visualization as a .png file directly (if the hosting server allows). Adaptive labels of the plots as well as a small box *inform the users about what the data represent*. Moreover, each of the three results

panes provides a download button by which the user can *store the data* at the selected level of aggregation locally as .csv files to be used further with any software that can handle spreadsheets. The three subsequent figures below illustrate this along the exemplary user search seen above.

FIGURE 2: PLS-WORDS APP RESULTS AGGREGATED OVER TIME

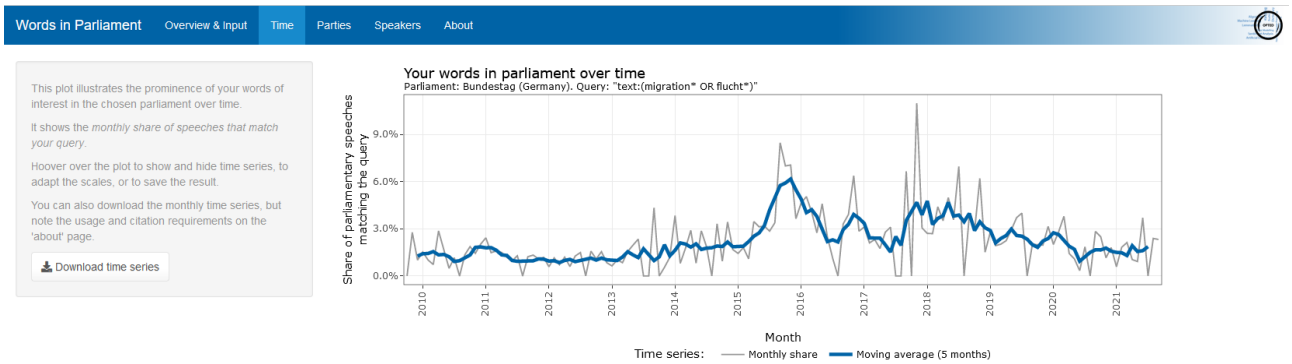


FIGURE 3: PLS-WORDS APP RESULTS AGGREGATED OVER POLITICAL PARTIES

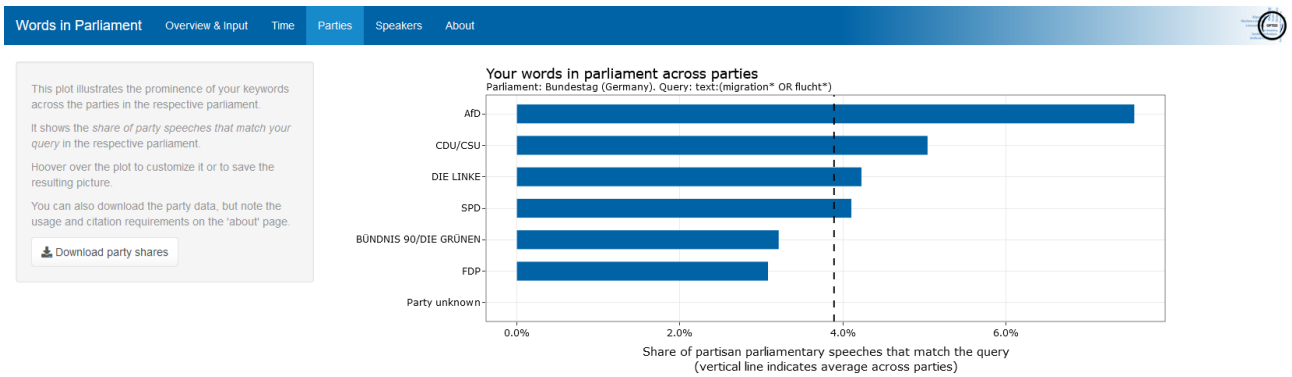
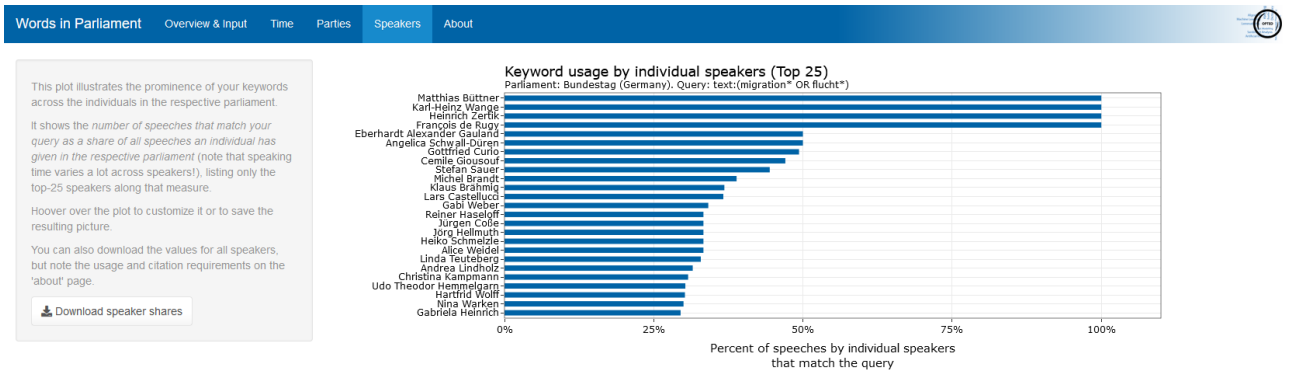
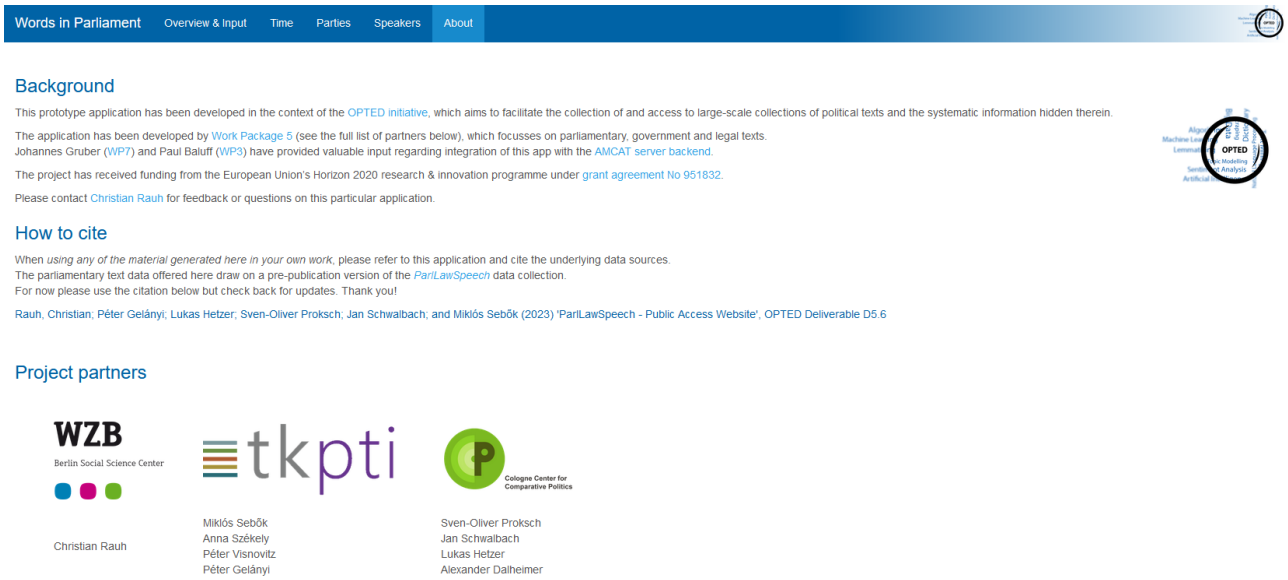


FIGURE 4: PLS-WORDS APP RESULTS AGGREGATED OVER PARLIAMENTARY SPEAKERS



Finally, the 'about' page provides users with additional background information, authors, and – importantly for academic work and public recognition – details about citation requirements (this is also referred to in all three results panes seen above).

FIGURE 5: PLS-WORDS APP – ABOUT PAGE

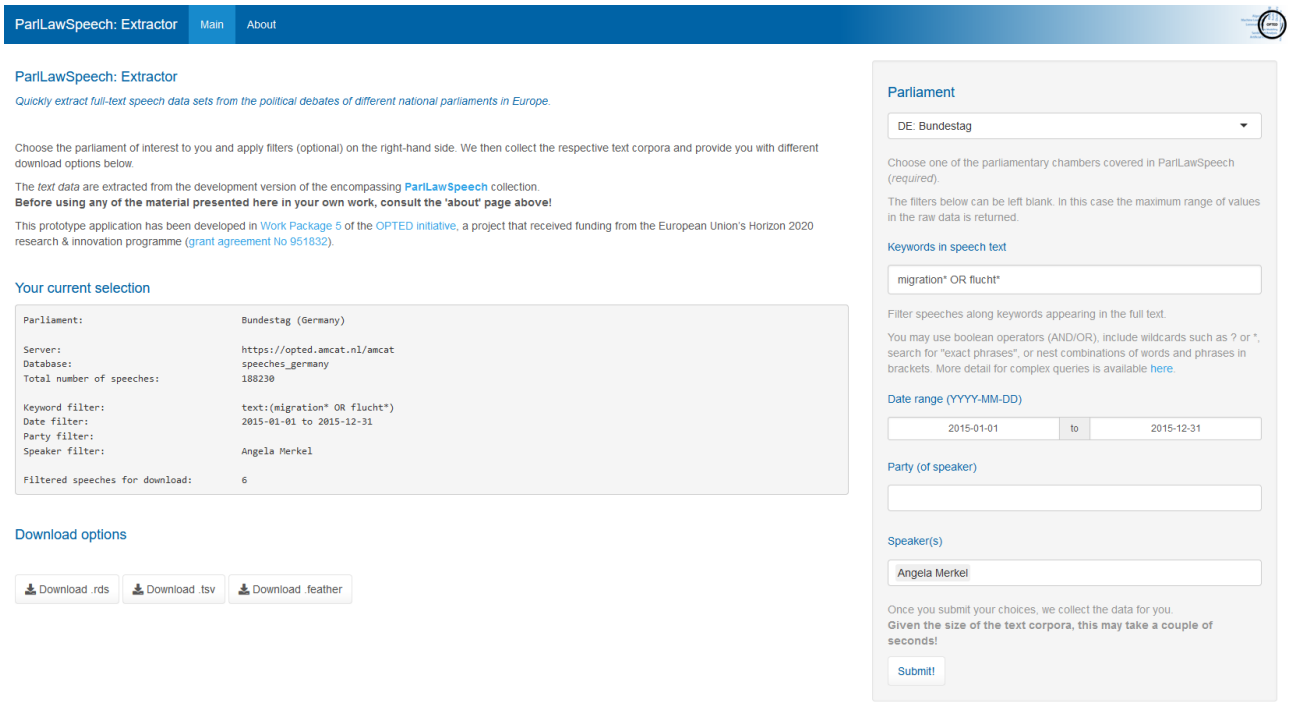


For non-technical users who have by then acquired a taste for digging further into text analysis or who want to gain additional qualitative insight from the full texts matching their interest, our second app – ***PLS-extract*** – offers the ability to extract specific data from the AMCAT servers running in the backend and download their selection for local processing by whatever means they prefer.

In terms of user interface and background information on the 'about' page we follow the same design principles and layouts as above, but the home screen is different. The user can use the right-hand box to select the parliament of interest and the apply optional filters with regard to keywords in the speech text, date of the speech, political party of speaker, or name of individual speakers (where multiple choices are allowed throughout). Upon submission of the choices, the app requests the corresponding full texts from the AMCAT server, which may take some time if the user choices result in large collections themselves. The buttons on the lower end of the page then allow users to store the resulting full-text data locally in one of three conventional formats (see above).

Figure 6 provides an example where a user is interested in any Bundestag speech by Angela Merkel that contained certain migration-related keywords in the full year 2015 - producing a small downloadable collection of six speeches.

FIGURE 6: PLS-EXTRACT APP – HOME/DOWNLOAD SCREEN



With these two apps, we hope to have demonstrated that researchers can themselves facilitate public access to very large collections of political text also for non-technical users if and when they can access standing server infrastructures and have opportunities for mutual exchange and cooperation.

Prototype 3: Cooperating with a web developer

Of course, modern web development technology offers much further potential with regard to advanced user experience and user interface design. Academic researchers alone, however, can hardly fully embark on following these technologies in detail by acquiring highly specialized skill sets.

Yet and still, we would be interested in how we could push the basic design principles developed here further. Thus, we were very happy when Peter Baluff (OPTED WP3) brought us in contact with Peter Walchhofer, a computer and data science student form the Technical University Vienna who was looking for data in the context of a term paper project. We took this opportunity to learn more about how a professional would go about increasing public access to large text collections, met up with Peter Walchofer to discuss our key design principles and to then share a development version of our ParlLawSpeech text collection.

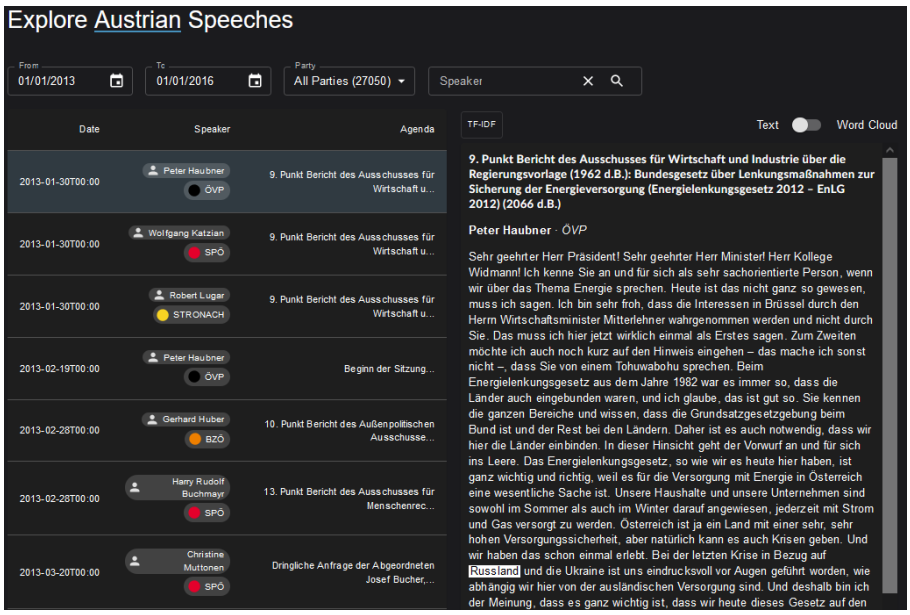
On this basis, Peter Walchhofer started to develop an application, notably also using AMCAT as a server backend, but exploiting more advanced web frameworks and user interface technologies for the frontend.[17] The resulting application "***ParlSpeechTracker"*** is visually very appealing, offers additional functionality (Figure 7), and improves markedly over our apps in terms allowing users to qualitatively engage with the full text directly and interactively in the web browser (Figure 8).

This example shows that fostering such co-operations between academics and web professionals in a more systematic manner would result in more appealing, more user-friendly, and more informative access tools to large scale text corpora also for laymen audiences. Enabling such co-operations also in the future would thus be a valuable service that a future platform for political text analysis in Europe could provide.

[17] In particular, the Python-based Flask (https://en.wikipedia.org/wiki/Flask_(web_framework)) and ReactJS for java-based user interfaces (https://react.dev/)

FIGURE 7: PARLSPEECHTRACKER – ADDITIONAL FUNCTIONALITY

FIGURE 8: PARLSPEECHTRACKER – DIRECT WEB ENGAGEMENT WITH RAW TEXT



Access to the web applications and their source code

First set of prototypes – self-contained shiny apps

As discussed above, these apps are somewhat slow and clunky for a text collection of our size but are provided for documentation and reference purposes nevertheless. Users willing to play with the ParlLawSpeech data should rather use the second and third sets of prototypes.

➢	PLS-words app:	https://shiny2.wzb.eu/rauh/PLS-words/
➢	PLS-extract app:	https://shiny2.wzb.eu/rauh/PLS-extract/
➢	Source code:	https://github.com/ChRauh/OPTED-WP5-Apps

Second set of prototypes – Shiny apps with AMCAT integration

➢	PLS-words app:	https://shiny2.wzb.eu/rauh/PLS-words-AMCAT/
➢	PLS-extract app:	https://shiny2.wzb.eu/rauh/PLS-extract-AMCAT/
➢	Source code:	https://github.com/ChRauh/OPTED-WP5-Apps-AMCAT/

Third set of prototypes – more advanced web technology

➢	ParlSpeechTracker:	https://parliaments.opted.eu/

Outlook and lessons learned for a future platform

In conclusion we think that the experiences summarized here as well as the sets of functional web applications that we provide offer two first of two proof-of-concept insights. First, online web applications can significantly lower the barriers of entry for non-technical users when it comes to accessing the wealth of information hidden in large academic text collections. Second, academics themselves can in principle build such applications at reasonable levels of investment in terms of time and resources.

However, our experiences also highlight that a future platform for political text analysis in Europe would generate significant added value for both academics as data providers and potential users such as journalists, public officials, or interested citizens in particular. Our discussion flags three services in particular by which such a platform would also support the purpose of this deliverable in the long run:

- Organize and enable regular in-depth exchange among academic withs different technical skill sets regarding text data analysis, data storage, and data visualization
- Provide a standing server structure at which academic data providers could host their applications and thus enhance public-facing communication of and access to the efforts behind large scale text data collections
- Fund respectively educated and/or skilled staff - “research engineers” - who can maintain these servers and who provide additional expertise on web platforms and user interface technologies.

Along these three steps, we think, a future platform for political text analysis in Europe would generate a sustainable foundation on which more user-friendly applications such as those proto-typed in this deliverable could grow so as to improve the wider public’s access to the systematic and democratically relevant information hidden in the large text collections that academic researchers provide.