BACHELOR'S THESIS

COMPUTER SCIENCE

DESIGN AND EVALUATION OF A WORD-CLUSTERING-TOOL FOR SOCIAL PSYCHOLOGY

LUIS KLOCKE

University of Bielefeld,
Faculty of Technology,
Research Group Knowledge Representation and Machine Learning

PROF. DR. BENJAMIN PAASSEN,
DR. ADIA KHALID

NOVEMBER 30, 2024

CONTENTS

1 INTRODUCTION

In social science research, the analysis of responses to open-ended survey questions is crucial for understanding human behavior and social phenomena, yet the manual coding process consumes extensive time and resources, potentially limiting the scope and impact of valuable research.
While automating the coding process through machine learning could dramatically reduce this burden, existing approaches often require pre-labeled training data or predefined coding schemas, making them impractical for many research scenarios.
This thesis addresses this challenge by developing an intuitive application that leverages state-of-the-art language models and unsupervised text clustering to automatically organize survey responses, without requiring prior coding schemas or technical expertise.

The initial approach to automating the coding process for survey responses was to manually engineer classifiers that work based on a matching responses to a dictionary of keywords, yet these methods fail to capture semantic nuances and require the researchers to create new lists of keywords for every survey.

More recent approaches to this problem have formulated it as a supervised classification, which reduces the overall effort in the coding process, but a subset of manually coded samples is still required.

This thesis proposes a novel approach that leverages unsupervised word clustering based on state-of-the-art Large Language Model (LLM) embeddings. By representing responses in a high-dimensional semantic space and grouping similar responses together, we can create meaningful categories that emerge naturally from the data itself. To make this approach accessible to researchers, it is implemented in an user-friendly application that aims to intuitive guide researchers through the process of setting up the clustering parameters and lets them explore the results without requiring technical expertise in machine learning or natural language processing. The perceived usability of this application is evaluated in a mini user study.

The contributions of this work are:

- The development of a free, open-source, data-confidential application that enables access to response clustering with state-of-the-art text embedding models to non-technical users
- Re-framing of automated coding for open-ended questions as an unsupervised learning problem
- Initial empirical validation of the perceived usability of the application in a mini user study with 3 researchers in social science

2 BACKGROUND AND RELATED WORK

2.1 AUTOMATED CODING FOR OPEN-ENDED QUESTIONS

In the quantitative analysis of survey data, *coding* refers to the process of assigning numbers and categories to responses for further analysis. For responses to open-ended questions, the manual coding process is time-consuming and expensive, thus, automation is highly sought after.

Automated coding is an active research topic. The most difficult kind of survey questions to code responses for are the open-ended questions, where participants can answer in their own words and the response is recorded verbatim. Their difficulty arises from the fact that it is usually not possible to create a generalized coding schema for the responses without first examining a subset of the collected responses to sample possible answer categories.

Open-ended questions provide unique value compared to other kinds of questions in research surveys: They don't carry any presumptions about the response and let the participant express their thoughts and feelings without limitation in their own words.

2.1.1 *Dictionary Based Approaches*

The initial approach to automating the coding process for survey responses was to manually engineer classifiers that work based on a dictionary. The simplest version of such a classifier maps words to specific codes. The classification process consists of matching every word in the responses to the dictionary and assigning codes wherever possible.

Viechnicki (1998, [Vie98]) compares two commercial solutions, one of which is a logical and dictionary-based system that assigns a category using boolean rules for the occurrence of keywords, while the other is a similarity-based classifier that assigns a category depending on the cosine distance of the response and category embeddings. The dictionary-based system outperformed the similarity-based method.

Pennebaker et al. (1999, [PFB99]) introduce the Linguistic Inquiry and Word Count (LIWC) software. It uses dictionaries to analyze text and allows the creation of custom dictionaries as well.

Nicolas et al. (2021, [NBF21]) develop content dictionaries specific to the stereotype domain, to be used for dictionary-based coding of open-ended questions. In a follow-up study (2022, [NBF22]) they use those dictionaries to develop a spontaneous stereotype content model. They note that this approach helped them uncover dimensions that previous stereotype models had not covered.

Iliev et al. (2015, [IDS15]) explore the application of user-defined dictionaries among others in the broad context of automated text analysis in psychology. They highlight the simple and flexible nature of the method, while also recognizing possible drawbacks, such as a blindness to context, sarcasm, metaphors, or idioms.

2.1.2 *Supervised Machine Learning Approaches*

A more recent approach has been to utilize supervised machine learning techniques for automated response coding. The typical approach consists of manually coding a small but representative subset of responses and training a classifier on it, to then be used on the rest of the data. Thus, the manual effort is spent on a task that is already known to researchers, compared to creating custom dictionaries.

Giorgetti and Sebastiani (2003, [GS03]) successfully demonstrate a Naive Bayes classifier and a Support Vector Machine classifier in the context of automated survey coding.

Schierholz (2014, [Sch14]) compares a dictionary and rule-based method to Naive Bayes, Bayesian Categorical ,and gradient boosted trees that combine the previous methods in the context of occupation coding.

Bethman et al. (2014, [Bet+14]) compare a Naive Bayes and a Bayesian Multinomial approach in the occupation coding domain.

Schonlau and Couper (2016, [SC16]) present a semi-automated categorization approach that uses multinomial boosting for classification in cases where the model is confident and manual coding for the remaining cases.

Severin et al. (2017, [SGK17]) apply a Naive Bayes classifier to survey response coding in the context of transportation planning.

He and Schonlau (2020, [HS20]) use an SVM model with a linear kernel but also report similar results using a random forest model.

Schierholz and Schonlau (2020, [SS20]) compare two dictionary-based algorithms and five machine learning algorithms for occupation coding.

2.1.3 *Transformer-based Classification Approaches*

The Bidirectional Encoder Representations from Transformers (BERT, see also section 2.3) has also recently been applied to the problem of automated response coding.

Gweon and Schonlau (2023, [GS23]) compare a fine-tuned BERT model to other machine learning models in the response classification task. They find that BERT barely wins with 100 manually coded observations but its advantage grows with more fine-tuning data.

Schonlau et al. (2023, [SWM23]) apply BERT to a multi-label response classification task, compare it to other multi-label algorithms, and find that BERT outperforms them. They conclude that for mildly multi-label classification tasks, fully automatic classification is now possible.

2.1.4 *Clustering Approaches*

Applying clustering to the automated coding of free-text responses[1] is a sparsely researched domain. By framing automated coding as an unsupervised machine learning problem, manual intervention in the coding process could theoretically be removed.

Zehner et al. (2016, [ZSG16]) develop an automatic coding process, wherein responses are preprocessed, semantically represented as a vector and clustered. Then a supervised machine learning model is trained to assign a code to each cluster. New responses are coded by choosing the most similar cluster's code. They demonstrate their method using data from the *Programme for International Student Assessment* (PISA) 2012 in Germany.

Andersen et al. (2021, [AZ21]) introduce the application shinyReCoR, written in R, which uses cluster-based methods to guide users to create automatic response classifiers with the help of annotated data.

Andersen et al. (2023, [AZG23]) combine a cluster-based method with human raters to form a semi-supervised coding method. They evaluate their method and find a large reduction in the manual coding effort without sacrificing coding accuracy.

Gefvert (2023, [Gef23]) explores advanced text representation models (SentenceBERT and Sent2Vec) for automating the clustering of free-text survey responses. By comparing multiple models and conducting manual evaluations, they find that while text representation methods show promise, the results are insufficient for fully automating the task, and manual intervention is still required.

This work also approaches automated coding as a clustering problem, but differs from previous approaches by using the current state-of-the-art large language models to embed the responses. Furthermore, these methods have been built into an application with a strong focus on the usability for non-technical users, focusing on an intuitive user interface with limited complexity.

2.2 CLUSTERING

In the field of Machine Learning, *unsupervised machine learning* refers to the practice wherein algorithms infer patterns from a dataset that is not labeled. Models learn solely based on the structure, similarities, and patterns inherent in the data itself. The training of an unsupervised machine learning model requires no human supervision or input.

Employing an unsupervised approach to automated coding eliminates the need for a pre-existing coding schema for the given data, thus creating a generalized yet data-driven methodology.

Clustering is one of the primary applications of unsupervised machine learning. The objective of a clustering algorithm is to automatically categorize data into groups (clusters) according to some similarity measure.

1 free-text responses are the response type for open-ended questions.

2.2.1 *K-Means Clustering*

k-Means clustering is one of the most fundamental and widely used algorithms in the field of unsupervised machine learning. It was independently discovered by Steinhaus et al. (1956, [Ste+56]), Lloyd (proposed in 1957, published in 1982, [Llo82]), Ball and Hall (1965, [BH65]), and MacQueen (1967, [Mac67]). Its primary objective is to partition a dataset of n observations into k clusters, where each observation belongs to the cluster with the nearest mean, which serves as the cluster prototype.

The k-means algorithm aims to minimize the within-cluster sum of squares (WCSS):

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where C_i is the set of points in cluster i and μ_i is the mean of the points in C_i.

The objective is to find the partitioning that minimizes this sum.

Finding the optimal solution to the k-means clustering problem is NP-hard (Drineas et al., 2004, [Dri+04]). Therefore, practical implementations of k-means rely on heuristic methods to find approximate solutions.

The k-means algorithm solves this problem using an iterative refinement approach, consisting of the following steps:

1. **Initialization**: Select k initial cluster centroids randomly or based on some heuristic.
2. **Assignment Step**: Assign each data point to the nearest cluster centroid.
3. **Update Step**: Recalculate the centroids as the mean of all data points assigned to each cluster.
4. **Convergence**: Repeat the assignment and update steps until the cluster assignments no longer change or the centroids stabilize.

This iterative process typically converges quickly to a local minimum, although the quality of the final clusters depends on the initial centroids.

Numerous enhancements and variations of the k-means algorithm have been proposed to address its limitations and improve its performance:

k-means++, by Arthur and Vassilvitskii (2007, [AV07]), improves the initialization step by choosing initial cluster centers in a way that spreads them out more evenly. This method significantly increases the likelihood of finding a solution close to the global optimum by reducing the probability of poor initial centroid choices. The initialization process works as follows:

1. Choose the first centroid randomly from the data points.
2. For each subsequent centroid, select a data point with a probability proportional to its squared distance from the nearest existing centroid.
3. Repeat until k centroids are chosen.

Dunn (1973, [Dun73]) proposed Fuzzy c-means, which was later improved by Bezdek (1981, [Bez81]): Unlike k-means, which assigns each data point to exactly one cluster, fuzzy c-means allows data points to belong to multiple clusters with varying degrees of membership, making it useful when clusters overlap or have ambiguous boundaries.

Proposed by Dan Pelleg and Andrew Moore (2000, [PM00]), X-means starts with an initial estimate of the number of clusters and dynamically adjusts this number by splitting and merging clusters based on a criterion such as the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC).

k-medoids, published by Kaufman and Rousseeuw (2005, [Leo90]) as Partitioning Around Medoids (PAM), uses actual data points (medoids) as cluster centers instead of means. k-medoids is less sensitive to outliers and noise, making it suitable for datasets with irregular cluster shapes.

Finding the Optimal Number of Clusters

Choosing the appropriate value for the hyperparameter K is a critical decision. Typically, the algorithm is tested with different values of K and evaluated using cluster validation measures.

Cluster validation measures are generally divided into external and internal measures; external validation measures compare the cluster assignments to ground-truth labels, while internal validation measures are based solely on the clustered data. Since ground-truth labels are rarely known, internal validation measures are more commonly used.

The primary cluster validation measures utilized in this work are the Silhouette Score and the Bayesian Information Criterion, although many others exist.

Silhouette Score

The Silhouette score is a measure of how similar an object is to its own cluster compared to other clusters. It combines both cohesion (how close data points are within a cluster) and separation (how distinct a cluster is from other clusters). The Silhouette score for each data point is calculated as follows:

1. Calculate the mean distance between the data point and all other points in the same cluster (cohesion).
2. Calculate the mean distance between the data point and all points in the nearest cluster (separation).

3. The Silhouette score $s(i)$ for a data point i is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where $a(i)$ is the cohesion and $b(i)$ is the separation.

The overall Silhouette score is the average of the Silhouette scores for all data points. The score ranges from -1 to 1, where a high value indicates that the data points are well-clustered.

Bayesian Information Criterion

The Bayesian Information Criterion (BIC) is a model selection criterion based on the likelihood of the data given the model, penalized by the complexity of the model. The BIC is defined as:

$$\text{BIC} = -2\ln(L) + p\ln(n)$$

where:

- L is the likelihood of the data given the model.
- p is the number of parameters in the model.
- n is the number of data points.

Lower BIC values indicate a better model.

2.2.2 *Agglomerative Clustering*

Hierarchical clustering is a family of clustering algorithms iteratively build clusters by merging or splitting them. Agglomerative clustering is a bottom-up approach to hierarchical clustering: Each data point starts as an individual cluster and the closest pairs of clusters are iteratively merged until all data points are contained within a single cluster or until a stopping criterion is met.

The agglomerative clustering process involves the following steps:

1. **Initialization**: Each data point is considered as a single cluster.
2. **Compute Distances**: Calculate the pairwise distances between all clusters. Common distance measures include Euclidean distance, Manhattan distance, and cosine similarity.
3. **Merge Clusters**: Identify the pair of clusters with the smallest distance and merge them into a new cluster.
4. **Update Distances**: Recompute the distances between the new cluster and all remaining clusters.
5. **Repeat**: Repeat steps 3 and 4 until a single cluster remains or the desired number of clusters is achieved.

The manner in which distances between clusters are computed and updated is defined by the linkage criterion. Several linkage criteria can be employed in agglomerative clustering:

- **Single Linkage**: The distance between two clusters is defined as the minimum distance between any single point in one cluster and any single point in the other cluster.
- **Complete Linkage**: The distance between two clusters is defined as the maximum distance between any single point in one cluster and any single point in the other cluster.
- **Average Linkage**: The distance between two clusters is defined as the average distance between all pairs of points in the two clusters.
- **Ward's Method**: This method aims to minimize the variance within each cluster. The distance between two clusters is defined based on the increase in the sum of squared deviations from the mean when the clusters are merged.

Agglomerative clustering has two main options for stopping criteria:

- Merge until a given number of clusters K is found.
- Merge until all distances exceed a threshold.

Bouguettaya et al. (2015, [Bou+15] present a hybrid approach that combines k-means and agglomerative clustering by first using k-means to create middle-level clusters represented by their centroids, followed by applying agglomerative clustering to build the final clustering hierarchy.

2.3 WORD EMBEDDINGS

Word embeddings are a type of word representation that allows words to be represented as vectors in a continuous vector space. This representation captures semantic relationships between words, making it possible for similar words to have similar vector representations. The concept of word embeddings gained prominence with the advent of neural network-based models, significantly enhancing the performance of natural language processing (NLP) tasks.

Mikolov et al. (2013. [Mik+13a], [Mik+13b]) introduced the Word2Vec model. Word2Vec consists of two model architectures for learning word embeddings: Continuous Bag of Words (CBOW) and Skip-Gram. CBOW predicts the target word from the context words, while Skip-Gram predicts the context words given the target word. Word2Vec embeddings are learned through a shallow neural network, making the process efficient and scalable.

Pennington et al. (2014, [PSM14]) introduced GloVe (Global Vectors for Word Representation). GloVe combines the advantages of global matrix factorization and local context window methods. It constructs a co-occurrence matrix from a corpus and uses this matrix to produce word vectors, capturing both local and global statistical information.

Bojanowski et al. (2016, [Boj+17]) developed FastText, which extended Word2Vec by incorporating subword information. Instead of learning vectors for entire words, FastText learns vectors for character n-grams and represents words as the sum of these n-gram vectors. This approach allows the model to generate representations for out-of-vocabulary words and capture morphological information, enhancing performance on languages with rich morphology.

Devlin et al. (2018, [Dev+19]) introduced BERT (Bidirectional Encoder Representations from Transformers). BERT revolutionized the field of NLP by providing context-aware word embeddings. Unlike previous models that generated static word embeddings, BERT uses a transformer architecture to create dynamic embeddings that consider the context of the entire sentence. This is achieved through a two-step process: pre-training and fine-tuning. During pre-training, BERT is trained on a large corpus using masked language modeling and next sentence prediction tasks. Fine-tuning then adapts the model for specific downstream tasks.

BERT's embeddings are context-dependent, meaning that the same word can have different embeddings based on its context in the sentence. This represents a significant advancement over earlier models like Word2Vec and GloVe, which produce a single, context-independent embedding for each word.

2.3.1 *Sentence Embeddings*

While word embeddings capture semantic information at the word level, sentence embeddings aim to represent entire sentences or phrases as single vectors. Sentence embeddings are crucial for tasks requiring an understanding of the relationships between sentences, such as semantic textual similarity, paraphrase detection, and document classification.

Early approaches to sentence embeddings often involved simple aggregation methods, such as averaging the word embeddings of the words in a sentence. However, these methods failed to capture the complexity of sentence semantics adequately.

Kiros et al. (2015, [Kir+15]) introduced Skip-Thought, which extends the idea of Word2Vec's Skip-Gram model to the sentence level. It trains an encoder-decoder architecture to predict surrounding sentences given a center sentence.

Cer et al. (2018, [Cer+18]) introduced the Universal Sentence Encoder (USE). USE marked a significant advancement in creating robust sentence embeddings. USE leverages a transformer-based architecture, similar to BERT, to encode sentences into high-dimensional vectors that capture their semantic content. USE has been widely adopted for various NLP tasks due to its ability to produce meaningful sentence representations.

Conneau et al. (2018, [Con+18] developed InferSent, which uses supervised learning with labeled data from the Stanford Natural Language Inference (SNLI) dataset (Bowman et al., 2015, [Bow+15]) and the MultiNLI (Williams et al., 2018, [WNB18]) to train a siamese bidirectional LSTM (BiLSTM) network. This approach demonstrated that universal sentence representations trained using the supervised data of the Stanford Natural Language Inference datasets can consistently outperform unsupervised methods like SkipThought vectors on a wide range of transfer tasks.

Reimers and Gurevych (2019, [RG19]) introduced Sentence-BERT (SBERT), an extension of the BERT model to produce high-quality sentence embeddings. SBERT fine-tunes BERT using a Siamese network architecture and a combination of supervised and unsupervised learning objectives. This architecture allows SBERT to efficiently compute sentence embeddings while maintaining high performance on various sentence-level tasks.

SBERT addresses a key limitation of BERT: the computational inefficiency of generating sentence embeddings. Traditional BERT requires multiple inference steps to compute sentence embeddings for pairs of sentences, making it impractical for large-scale applications. SBERT, on the other hand, uses a Siamese network to encode sentences into fixed-size vectors, allowing for efficient similarity calculations through cosine similarity or other distance metrics.

Muennighoff et al. (2022, [Mue+23]) introduced the Massive Text Embedding Benchmark (MTEB), which spans multiple embedding tasks across numerous datasets and languages. The *BAAI/bge-large-en-v1.5* model, published by Xiao et al. (2023, [Xia+23]), achieves an average score of 64.23 on the English language MTEB (ranked 42nd as of November 2024) despite being only 1.24 GB in size, making it a strong candidate for a local embedding model.

2.4 COSINE SIMILARITY

To cluster or compare highly dimensional vectors such as sentence embeddings, it is vital to choose a fitting similarity measure. In highly dimensional spaces, the magnitude of vectors can vary widely, affecting magnitude-based metrics. Cosine similarity is a measure that quantifies the similarity between two vectors by the cosine of the angle between them.

The cosine similarity between two vectors A and B is defined as:

$$S_C(A, B) = \frac{A \cdot B}{\|A\| \|B\|}$$

where $A \cdot B$ represents the dot product of the vectors A and B, and $\|A\|$ and $\|B\|$ denote their magnitudes.

The value of cosine similarity ranges from -1 to 1, where 1 indicates that the vectors are identical, 0 means they are orthogonal (i.e., no similarity), and -1 signifies that the vectors are diametrically opposed.

The cosine similarity is very efficient to calculate by first normalizing all the vectors and then calculating their inner products.

The cosine distance is a distance measure typically defined as

$$D_C(A, B) := 1 - S_C(A, B).$$

The cosine distance can be expressed as a monotonic transformation of the Euclidean distance when dealing with unit length vectors:

$$D_C(A, B) = \frac{\|A - B\|^2}{2}, \quad \text{when} \quad \|A\|^2 = \|B\|^2 = 1.$$

Therefore, substituting the cosine distance with the Euclidean distance is possible when the vectors are normalized. This way, algorithms that use the Euclidean distance, like k-means, can use a metric that carries the same relative information as the cosine distance.

2.5 SYSTEM USABILITY SCALE

The System Usability Scale (SUS) is a scale of perceived usability and was introduced by Brooke (1995, [Bro95]). It was originally intended as a "quick and dirty" low-cost assessment of usability in industrial systems evaluation. Since then it has become the most widely used standardized questionnaire for the assessment of perceived usability.

The standard SUS consists of 10 statements that the user assigns a score from 1 to 5, ranging from strongly disagree to strongly agree. It contains alternating positive and negative toned questions to control acquiescent response bias and to easily identify serial extreme responders.

Sauro and Lewis (2011, [SL11]) note 3 major potential disadvantages to the system of alternation:

- Users may respond differently to negatively worded items
- Users may accidentally answer to a question with the wrong side of the scale
- Researchers may miscode the data by forgetting to reverse the scales when scoring

They introduce a positive version of the SUS without alternating items:

1. I think that I would like to use the website/application frequently.
2. I found the website/application to be simple.
3. I thought the website/application was easy to use.

4. I think that I could use the website/application without the support of a technical person.

5. I found the various functions in the website/application were well integrated.

6. I thought there was a lot of consistency in the website/application.

7. I would imagine that most people would learn to use the website/application very quickly.

8. I found the website/application very intuitive.

9. I felt very confident using the website/application.

10. I could use the website/application without having to learn anything new.

They also found that the scores between the standard SUS and the positive SUS were similar, which is why they recommend that researchers use the positive version of the SUS for usability evaluations.

To score the SUS, researchers must first fill in missing responses with the neutral answer on the scale, which is a raw item score of 3. The benefit of the positive SUS is that the score is simpler to calculate. The score is computed by converting the raw item scores to adjusted scores ranging from 0-4. The adjusted scores are summed up and multiplied by 2.5 to get the final score per participant, ranging from 0 - 100.

Sauro and Lewis (2016, [SL16]) created a curved grading scale (CGS) interpretation for SUS scores based on percentile rank (depicted in table 2.1).

SUS Score Range	**Grade**	**Percentile Range**
84.1–100	A+	96–100
80.8–84.0	A	90–95
78.9–80.7	A–	85–89
77.2–78.8	B+	80–84
74.1–77.1	B	70–79
72.6–74.0	B–	65–69
71.1–72.5	C+	60–64
65.0–71.0	C	41–59
62.7–64.9	C–	35–40
51.7–62.6	D	15–34
0.0–51.6	F	0–14

Table 2.1: SUS Score to Grade and Percentile Range

2.6 SOCIAL CASINO GAMERS

Kim et al. (2023, [Kim+23]) investigated individuals' motivations for playing social casino games[2]. They inquired about participants' motives for playing social casino games, as well as their reasons for transitioning between social casino games and gambling, both to and from.

They graciously made their survey data publicly available on the OpenScience framework ([FD17]), which is why it was selected as the example dataset for the application introduced in this work.

2.7 ELECTRON

Electron is an open-source framework developed by GitHub and now maintained by the OpenJS Foundation[3] that allows for the creation of cross-platform desktop applications using web technologies such as HTML, CSS, and JavaScript. It combines the Chromium rendering engine and the Node.js runtime, enabling developers to leverage the extensive ecosystem of web development tools and libraries. Electron is very useful for developers who are usually familiar with web technologies that want to create desktop applications.

2.7.1 *Process Model*

Understanding the underlying process model of Electron is crucial to understanding how Electron applications work.

The **main process** serves as the singular core of the application and is responsible for controlling the application's lifecycle. It is essentially a Node.js environment with full access to Node.js APIs, allowing it to interact with the file system, create windows, handle events, and perform backend tasks.

Each window in an Electron application runs in its own **renderer process**. These processes are isolated from each other and run in a Chromium sandbox environment, meaning they follow web standards which include the following functionalities:

- Having a single HTML file entry point
- Enabling UI styling with CSS
- Executable JavaScript can be added through <script> elements.

The renderer process is responsible for rendering the application's user interface and executing front-end code.

2 A social casino game is a type of online game that mimics traditional casino games without the use of real money for wagering. These games typically operate within a social media environment or app, allowing players to interact with each other, share experiences, and sometimes compete for virtual rewards or status.

3 https://openjsf.org/

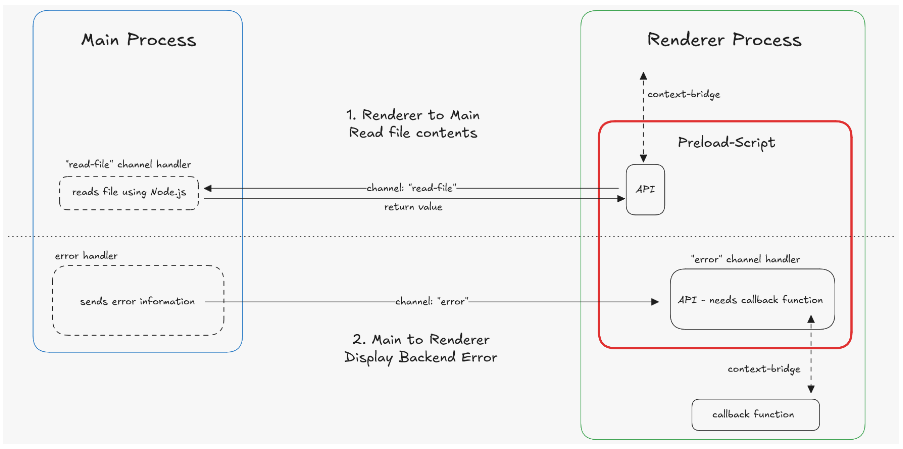


Figure 2.1: Inter-Process Communication

Importantly, the renderer process does not have direct access to Node.js APIs for security reasons. It is considered *context-isolated*.

Preload scripts run code inside a renderer process before the web contents begins loading. During this time they have the context of the renderer process but also have access to Node.js APIs. Preload scripts are used to expose specific APIs over a so-called context-bridge to the renderer process.

2.7.2 *Inter-Process Communication*

Because the main and renderer processes have different responsibilities in Electron's process model, Inter-process communication (IPC, see Figure 2.1) is the only way to perform many common tasks such as calling a native API from the user interface. IPC strictly follows developer-defined channels. The *ipcRenderer* and *ipcMain* modules contain method for asynchronous communication across such channels from renderer to main and main to renderer process, respectively.

Renderer to main process - Example: Read file contents

1. Preliminary steps before the renderer's web contents have begun loading:

    - Preload script exposes an API to the renderer that takes a file path argument and invokes the *ipcRenderer* on the *read-file* channel.

    - Main process registers a handler using the *ipcMain* module on the *read-file* channel that expects a string argument for the path.

2. Renderer code (i.e. as a reaction to user interaction) calls the exposed API

3. Main process handler receives the argument on the *read-file* channel and executes the handler, which calls the Node.js filesystem API to read the file contents and returns the data asynchronously
4. Renderer code receives the data after the exposed preload script method finishes waiting for the data.

Main to renderer process - Example: Clustering script encounters an error

1. Preliminary steps before the renderer's web contents have begun loading:
   - Preload script exposes a method to the renderer that takes a callback function as an argument and registers it as an event handler with the *ipcRenderer* on the *error* channel that executes the callback function.
   - Renderer code registers a callback function to the exposed API that displays the error in the user interface
2. Main process receives message from the script that an error has occurred.
3. Main process sends a signal to the renderer process on the *error* channel
4. *ipcRenderer* event handler in the preload script triggers on the *error* channel and calls the registered callback function of the renderer

2.8 REACT

React is a popular open-source JavaScript library for building user interfaces, particularly single-page applications where data dynamically changes over time. Developed by Meta, React enables developers to create web applications that can update and render efficiently in response to data changes. The core philosophy of React is based on components and a unidirectional data flow, making the development of complex UIs more manageable and maintainable. Combined with Electron, React allows for the creation of interactive user interfaces in desktop applications as well.

2.8.1 *Components*

Components are the building blocks of a React application. A component in React is a reusable piece of UI that can be nested, managed, and handled independently. They accept props (short for properties) as arguments from their parent component and return React elements (such as HTML or other Components) that describe what should appear on the screen.

2.8.2 *State Management*

State is a crucial concept in React, representing a component's dynamic data that can change over time. Unlike props, which are read-only and passed down from parent components, state is managed within the component itself.

When state is updated, the component managing the state is re-rendered, then all its children are recursively re-rendered.

2.9 PYTHON

Python is a high-level, interpreted programming language known for its simplicity and readability. It was developed by Guido van Rossum. Python scripts, which are plain text files containing Python code, can be executed directly by the Python interpreter, making development and testing processes straightforward.

As one of the most popular languages for machine learning and data science, python has accumulated a rich ecosystem of libraries and frameworks specifically designed for machine learning, such as PyTorch, Scikit-learn, and Sentence-Transformers. These tools provide pre-built modules and functions that simplify the implementation of complex algorithms and models.

2.9.1 *Pyinstaller*

PyInstaller is a tool that compiles Python applications into stand-alone executables. It packages Python scripts and all their dependencies, including a minimal Python interpreter, into a single executable file. This allows Python applications to be distributed to users who do not have Python installed on their systems.

It's important to note that executables created with PyInstaller are platform-specific. This means that the binary must be compiled on the same operating system that it will run on.

3 METHODS

The primary objectives of the application developed in this work are as follows: First, to provide an intuitive user interface that ensures accessibility for non-technical researchers. This focus on user-friendliness aims to bridge the gap between advanced computational methods and users without a technical background. Second, the application incorporates access to state-of-the-art embedding models. Lastly, the design emphasizes data confidentiality, ensuring that sensitive data is handled securely and making the application suitable for use in data-sensitive fields where privacy is a critical concern.

3.1 APPLICATION ARCHITECTURE

The application architecture (see Figure 3.1) is organized into four major components: the Electron application, the user interface, the clustering script, and the results directory.

The **Electron application** serves as the core of the system, managing the creation of the application window, user settings, and the interaction between the user interface and native APIs. It has access to result data in the results directory. Electron was chosen as the framework for this application due to its ability to develop user interfaces with web technologies and its simplification of the compilation process.

The **user interface** is constructed using React components, which facilitate a nested approach to building the interface. This design allows for fast and seamless page transitions while maintaining the central state of the chosen clustering parameters. Electron's renderer process renders the user interface in the application window. The React framework accelerated the development of a dynamic and responsive user experience.

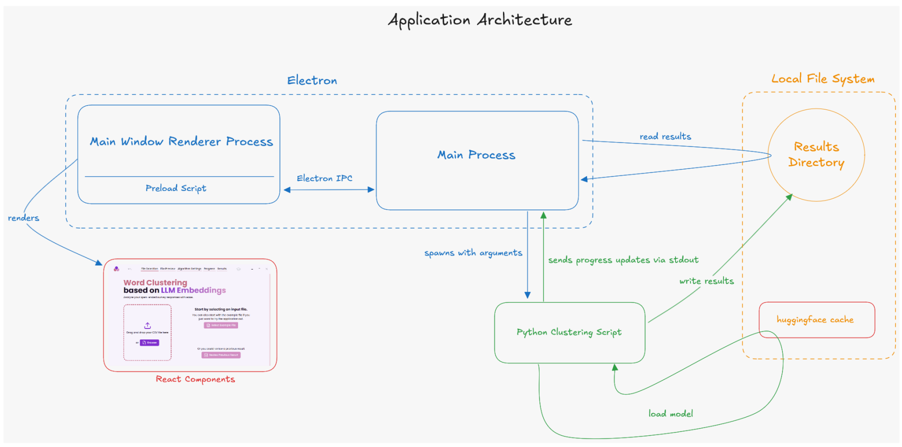


Figure 3.1: Application Architecture

The **clustering script** is implemented in Python, simplifying the implementation of clustering algorithms and inference on the embedding model. The script is compiled using PyInstaller and packaged with the rest of the Electron application. It functions as a standalone script that accepts command line arguments for clustering parameters, creates the response embeddings, performs clustering steps, communicates progress to the Electron main process, and saves run artifacts in the results directory. The electron main process can run the script and supply the required parameters. To uphold data confidentiality, the script uses exclusively local models.

The **results directory** contains the artifacts generated in a run:

- The main output file (CSV) is the original input file with added columns for cluster indices.
- The cluster assignments file (CSV) contains responses, their cluster indices, and their similarity to the cluster center.
- The cluster similarities file (JSON) provides between-cluster similarity values for each pair of clusters.
- The outliers file (JSON) lists outliers with response content, mean similarity to the k-nearest neighbors, and the global outlier similarity threshold.
- The cluster mergers file (JSON) documents mergers. Each merger is represented by their list of clusters (with their most representative responses) and all between-cluster similarities
- The time stamps file (JSON) records each step of the clustering process with Unix epoch timestamps.
- The arguments file (JSON) includes the file settings and the algorithm settings used for the run, and the location of the run's results directory.
- The automatic cluster count evaluation file (PNG) plots the cluster validation measures and the chosen compromise between them.

The results directory simplifies the process of persisting results and ensures that both the clustering script and the main process can access them efficiently. The results directories for all runs are stored in a central location.

3.2 USER INTERFACE

The primary objective of the user interface design is to create an intuitive and efficient experience for both non-technical and experienced users. The interface is designed to reward experienced users with speed while enabling novice users to develop expertise through consistent use. To achieve these goals, the application employs a linear,

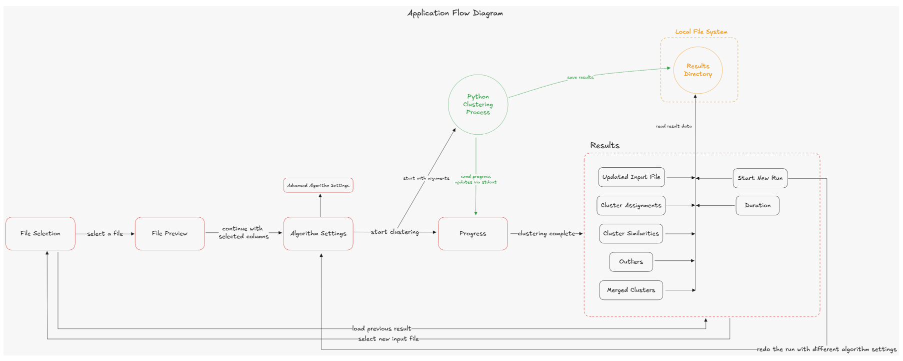


Figure 3.2: Application Flow Diagram

unidirectional structure that distributes complexity across multiple pages[1], groups semantically related settings, and minimizes the complexity on any single page. Additionally, a tutorial mode is integrated to assist beginners.

The application is structured linearly, meaning that each step logically follows and builds upon the previous one. This design choice ensures that users understand the process flow and can easily follow the sequence of steps necessary to complete their tasks. The navigation bar at the top of the interface highlights the user's current position within the process, enhancing orientation and overall user experience.

By distributing the complexity of the process across multiple pages and limiting the complexity on any single page, the application maintains an intuitive and user-friendly interface. This design approach ensures that both novice and experienced users can navigate the application efficiently, achieving their goals with minimal friction.

To support novice users, a tutorial mode is available, providing hover hints for most options within the application. This feature can be disabled by experienced users who do not require guidance, ensuring the interface remains uncluttered for those familiar with the process.

Each process step is explored in detail below.

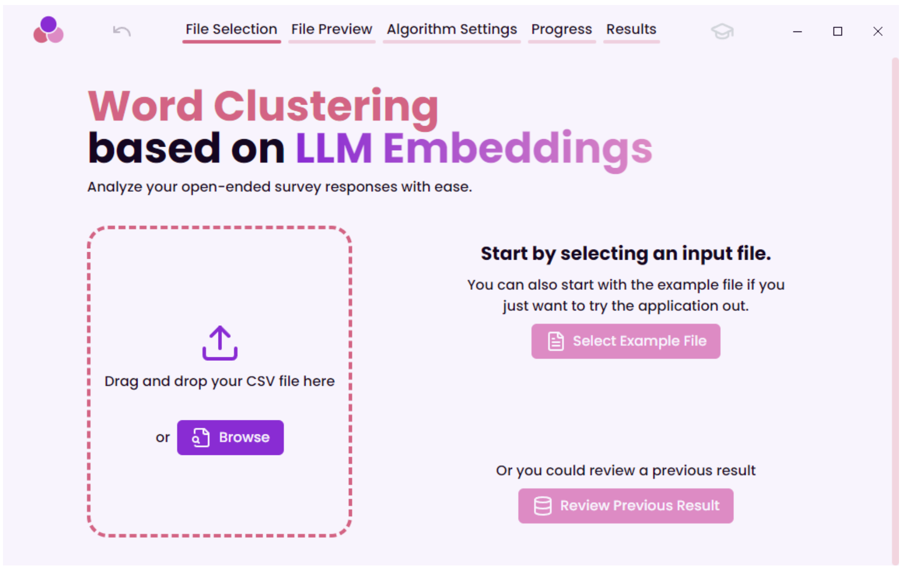
3.2.1 *File Selection*

The file selection (see Figure 3.3) serves as the starting point of the application. It enables the user to initiate the clustering process by selecting an input file or to review a previous result. The primary objective of this page is to create a welcoming interface that facilitates the most critical decision in the clustering process: selecting which file to cluster.

This page is designed to be a straightforward entry point, allowing experienced users to quickly proceed to the next step, while also providing a welcoming experience for first-time users.

The file can be selected either by dragging and dropping it into the highlighted box or by clicking on "Browse" and selecting it from

1 Page refers to the different distinctive sections of the application



Word Clustering
based on LLM Embeddings
Analyze your open-ended survey responses with ease.

Start by selecting an input file.
You can also start with the example file if you just want to try the application out.

Or you could review a previous result

Figure 3.3: File Selection

the native file browser. Offering both options ensures that users can choose the method that they find most comfortable and efficient.

Including an option to select an example file allows users to quickly explore and understand the application's features without requiring their own data. This is especially useful for onboarding or demonstration purposes. The example file used is the dataset on self-generated motives for social casino gamers (see Section 2.6).

Reviewing a past result becomes a central part of the workflow for experienced users, as repeating runs with adjusted algorithm settings allows users to maximize the effectiveness of the clustering process. Placing this option on the welcome screen is necessary because it is the only exception to the linear structure of the application, as it bypasses the entire clustering process. The button opens a modal displaying all previous runs by their run name and creation date. Clicking on a run transitions to its results page.

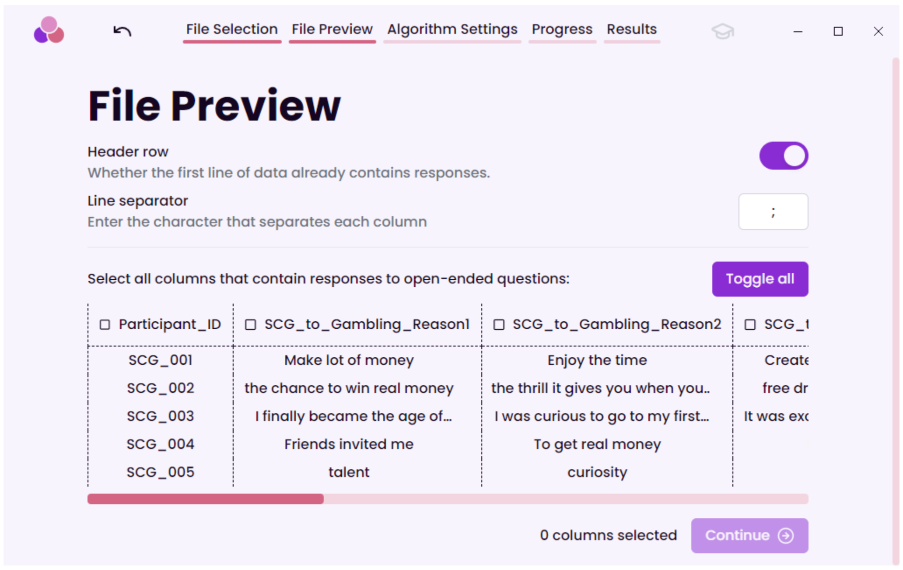
Selecting a file automatically triggers a page transition to the file preview.

3.2.2 *File Preview*

The file preview page (see Figure 3.4) allows users to select file-related settings while displaying a preview of the data. The primary objective of this section is to enable the user to select the correct column delimiter[2], determine whether the data contains a header, and identify the columns that contain responses to open-ended questions.

The file preview displays the first five rows of the data. Each column has a clickable header, which allows the user to mark the column for analysis inclusion. Since input files may contain a large number of

2 The symbol splitting the columns in a CSV file, typically a comma.



File Preview

Header row
Whether the first line of data already contains responses.

Line separator
Enter the character that separates each column

;

Select all columns that contain responses to open-ended questions:

☐ Participant_ID	☐ SCG_to_Gambling_Reason1	☐ SCG_to_Gambling_Reason2	☐ SCG_t
SCG_001	Make lot of money	Enjoy the time	Create
SCG_002	the chance to win real money	the thrill it gives you when you..	free dr
SCG_003	I finally became the age of...	I was curious to go to my first...	It was exc
SCG_004	Friends invited me	To get real money	
SCG_005	talent	curiosity	

0 columns selected

Figure 3.4: File Preview

columns, the preview is side-scrollable. A "toggle-all" button enables users to invert the current column selection, which is particularly useful when there are many columns, of which most or very few should be selected for analysis.

A key feature of this page is that the file preview immediately adapts to the user's selected settings. This allows users to observe how their settings affect the parsed file. It reinforces the user's decisions by providing real-time feedback, allowing them to verify that the data is correctly spread across the preview columns.

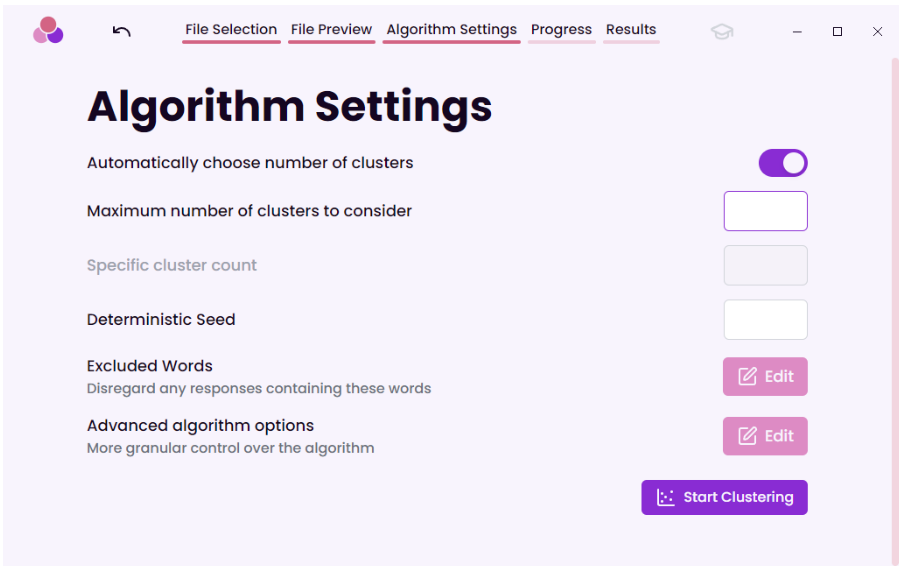
To enhance the preview's utility, any empty responses are removed and replaced with non-empty responses from the input file.

The application automatically selects an initial recommended delimiter based on the data, testing several common delimiters and selecting the one that yields the best parsing. This streamlines the process for most input files, while still allowing users to change the delimiter to meet specific requirements.

The "Continue" button remains disabled until the user has selected at least one column for clustering. The number of currently selected columns is displayed next to the "Continue" button, ensuring that the user is aware of their full selection, even when some columns do not fit on the page.

3.2.3 *Algorithm Settings*

The algorithm settings page (see Figure 3.5) allows users to adjust the parameters for the clustering process. The primary objective is to provide users with the option to fine-tune the clustering process according to their use case and level of experience, while also ensuring reproducibility, such that a run can be repeated with modified settings based on results analysis.



Algorithm Settings

Automatically choose number of clusters

Maximum number of clusters to consider

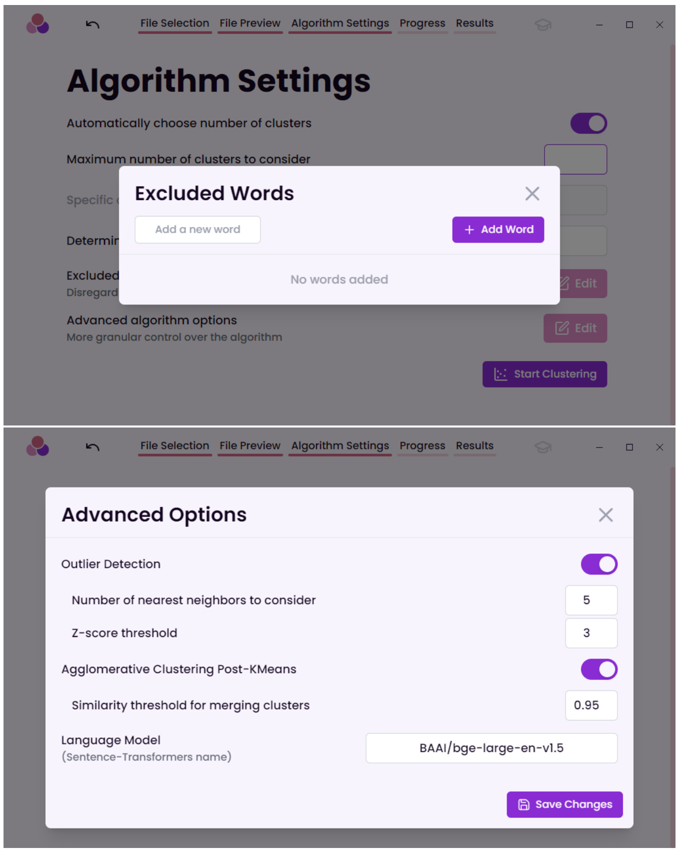
Specific cluster count

Deterministic Seed

Excluded Words
Disregard any responses containing these words

Advanced algorithm options
More granular control over the algorithm

Figure 3.5: Algorithm settings


File Selection
File Preview
Algorithm Settings
Progress
Results
Algorithm Settings
Automatically choose number of clusters
Maximum number of clusters to consider
Specific
Determin
Excluded
Disregard
Advanced algorithm options
More granular control over the algorithm
Edit
Edit
Start Clustering

Excluded Words

No words added

Advanced Options

Outlier Detection

Number of nearest neighbors to consider 5

Z-score threshold 3

Agglomerative Clustering Post-KMeans

Similarity threshold for merging clusters 0.95

Language Model
(Sentence-Transformers name) BAAI/bge-large-en-v1.5

Figure 3.6: Algorithm Settings Modals

Users can toggle the option to use automatic cluster count selection for k-means. The maximum number of clusters to consider is only available when automatic cluster count selection is enabled. This value limits the automatic cluster count selection and, by default, is set to half of the number of responses. The specific cluster count option is only available when automatic cluster count selection is disabled. This option directly sets the initial k-means cluster count. Disabling the unavailable option when toggling the automatic cluster count selection makes it clear to the user that these options are mutually exclusive. Furthermore, when the toggle is disabled, setting the specific cluster count becomes mandatory to initiate the clustering.

The deterministic seed option is used to set a seed for the random generators in the clustering algorithms to ensure reproducible results. If the deterministic seed is left empty, it is assigned a random value and persisted, ensuring that the result of the run is reproducible, even if the user did not explicitly set or anticipate the need to set a seed.

The excluded words (see Figure 3.6) can be edited in a modal. Responses containing these words are excluded from the analysis. The modal displays all currently excluded words and allows the user to enter new ones. The excluded words editor is housed in a modal to provide sufficient space for displaying potentially long lists and to emphasize its independence from the other settings on the page.

Finally, the Advanced Algorithm Options modal (see Figure 3.6) provides users with access to more technical configurations.

This modal includes a toggle option for outlier detection, as well as controls for tuning the number of nearest neighbors and the z-score threshold for outlier detection.

Additionally, the modal contains an option to run an additional agglomerative clustering procedure after the initial k-means clustering, merging similar clusters. Users can adjust the similarity threshold required for clusters to be merged.

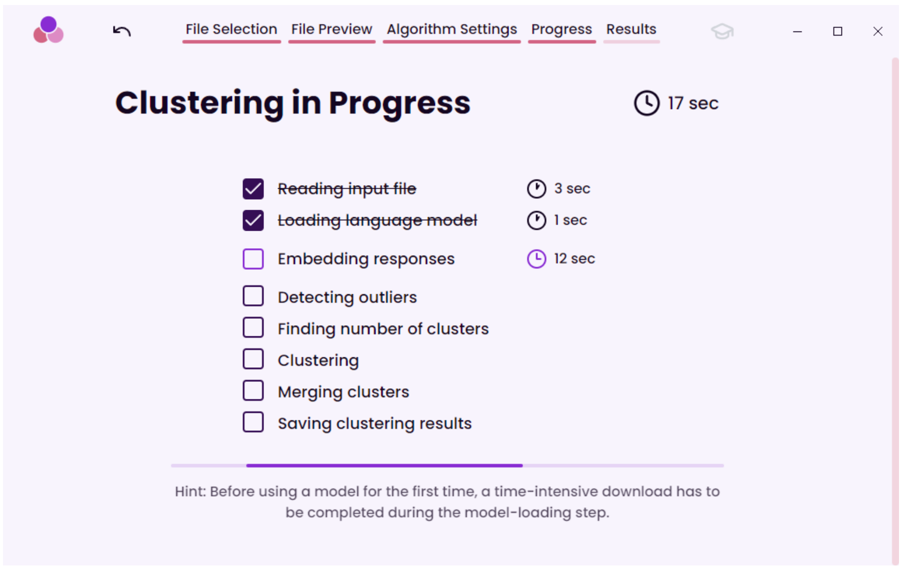
There is also an option to change the language model to a different embedding model available within the Sentence-Transformers library.

The advanced options are concealed in a modal to indicate that they are intended for experienced users. The save button ensures that an inexperienced user cannot inadvertently modify these settings.

3.2.4 *Progress*

The progress page (see Figure 3.7) displays the status of the ongoing clustering process. The primary objective of this page is to enhance the transparency of the clustering process by allowing the user to observe the completion progress and the elapsed time for each step.

Upon initialization, the progress page receives all scheduled clustering steps from the clustering script and generates a to-do list. The specific steps scheduled depend on the selected settings. As the clustering script executes, it sends progress updates that are displayed on this page (see Section 3.4 for details on the main-cluster process communication). The to-do list, with steps being checked off upon



Clustering in Progress

17 sec

✓	~~Reading input file~~	3 sec
✓	~~Loading language model~~	1 sec
☐	Embedding responses	12 sec
☐	Detecting outliers	
☐	Finding number of clusters	
☐	Clustering	
☐	Merging clusters	
☐	Saving clustering results	

Hint: Before using a model for the first time, a time-intensive download has to be completed during the model-loading step.

Figure 3.7: Progress Page

completion, is designed to provide a more engaging and satisfying progression for the user.

The total elapsed time is tracked by a prominent timer at the top of the page, while the elapsed time for each individual step is displayed next to it. An indeterminate loading bar is positioned at the bottom of the page, accompanied by a message indicating model download when a non-default model is used for the first time. The indeterminate loading bar and the increasing timers serve as indicators that the process is progressing and not stalled.

The progress page transitions to the results page once the clustering process is complete.

3.2.5 *Results*

The results page (see Figure 3.8) presents the complete results of a clustering run in one centralized location. Users are provided with options to view the updated input file[3], start a new run, rename the run, or review the duration of the process. Additionally, several result-analysis options are available directly within the application:

- Cluster Assignments
- Cluster Similarities
- Outliers (if present)
- Merged Clusters (if present)

3 Not to be confused with the actual input file, the updated input file is the main output file.

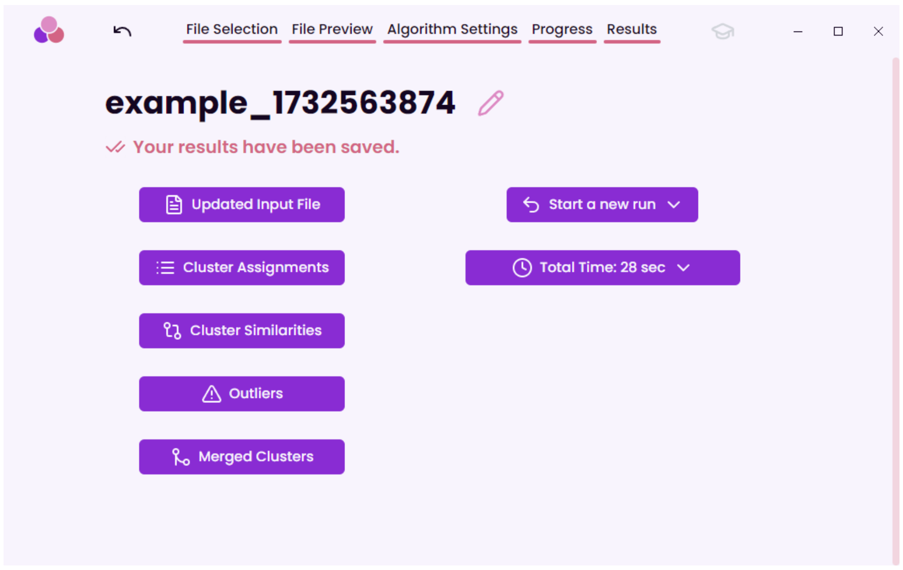


Figure 3.8: Results Page

The primary goal of this page is to serve as the final step in the clustering process, providing easy access to all results, including detailed views and the option to initiate a new run with modified algorithm settings.

The page displays the name of the current run, which is automatically generated but can be modified by clicking on the pen icon next to the name. Allowing users to change the run name facilitates easy retrieval of the correct run for future reference.

The "updated input file" button opens the location of the output file from the clustering process in the native file browser. This file is stored in the results directory alongside other result files. Direct access to this file is intended to simplify further analysis by researchers who wish to examine the output in other qualitative analysis programs.

The “start a new run” button opens a dropdown with two options:

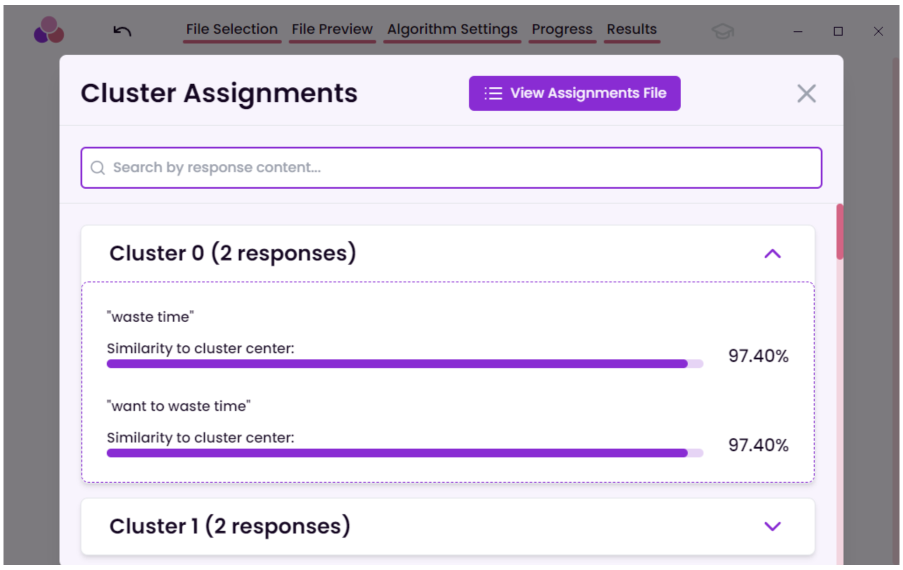
- Change the algorithm settings (keep the same file)
- Select a new file (reset all settings)

Providing the option to initiate a new run by modifying only the algorithm settings enables users to efficiently explore changes based on the results. The correct seed is automatically set for reproducibility.

The total time dropdown displays the total elapsed time for the clustering process and, when expanded, provides the duration for each individual process step.

The other buttons each open additional analysis options in modals, facilitating quick navigation and comparison between different result types. Similarity values between responses or clusters are presented as percentage values to ensure ease of interpretation for non-technical users.

Each analysis option is discussed in detail below:



Cluster 0 (2 responses)

"waste time"
Similarity to cluster center: 97.40%

"want to waste time"
Similarity to cluster center: 97.40%

Cluster 1 (2 responses)

Figure 3.9: Cluster Assignments

Cluster Assignments

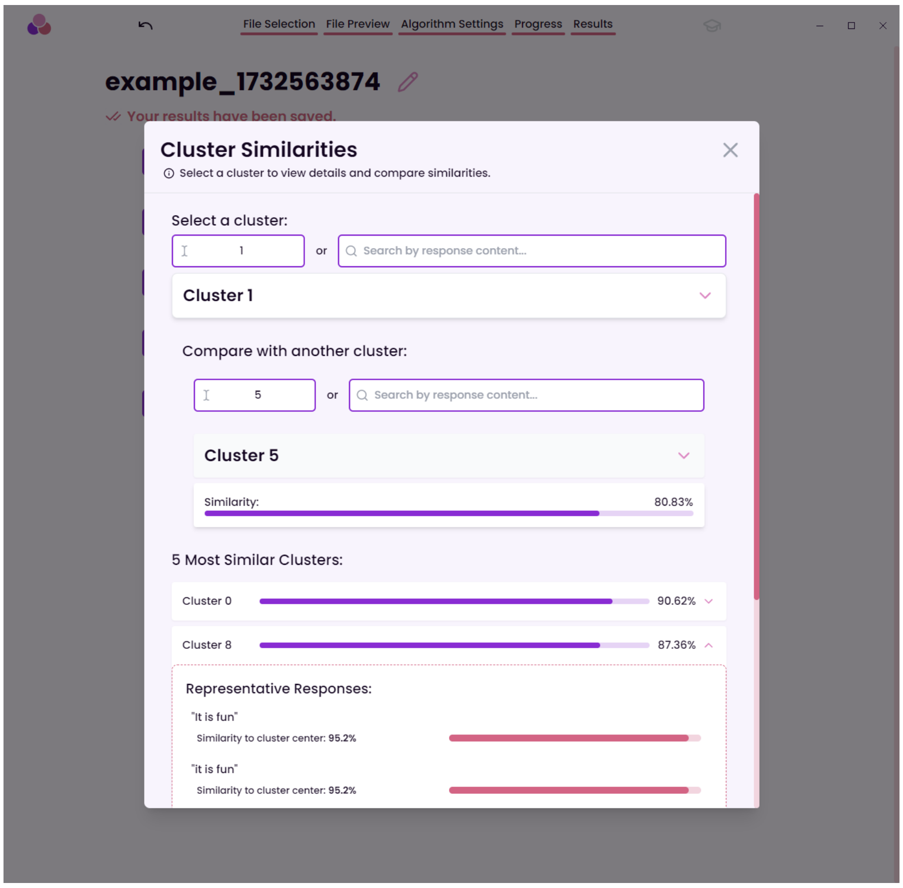
The cluster assignments modal (see Figure 3.9) provides insights into each generated cluster and the responses contained within them. Users can enter a search term to filter and view only those clusters that contain matching responses.

The primary goal of this modal is to allow users to examine the existing clusters and gain an overview of the types of responses represented by each cluster. This enables users to assess the success of the clustering process and determine whether the clustering achieves the desired level of granularity.

The main section of the display consists of a list of clusters, each accompanied by its response count. Displaying the total response count for each cluster allows users to quickly evaluate the balance of the clusters.

Each cluster can be expanded to reveal the responses it contains, along with their similarity to the cluster center. To optimize performance, only the first 50 responses are displayed for clusters containing a large number of responses; however, the full list of responses can be accessed through the corresponding results file by clicking the 'view assignments file' button.

A search bar enables users to search for specific response content. The cluster display is then filtered to show only those clusters containing responses that match the search term. This allows users to quickly identify the cluster associated with a particular response. As the search results are updated iteratively, users can observe the progression of the filter, providing greater transparency regarding the search process.



example_1732563874

Your results have been saved.

Cluster Similarities

Select a cluster to view details and compare similarities.

Select a cluster: 1 or Search by response content...

Cluster 1

Compare with another cluster: 5 or Search by response content...

Cluster 5

Similarity: 80.83%

5 Most Similar Clusters:

Cluster 0 90.62%

Cluster 8 87.36%

Representative Responses:

"It is fun"
Similarity to cluster center: 95.2%

"it is fun"
Similarity to cluster center: 95.2%

Figure 3.10: Cluster Similarities

Cluster Similarities

The cluster similarities modal (see Figure 3.10) enables the user to compare different clusters and obtain an overview of which clusters exhibit similarities.

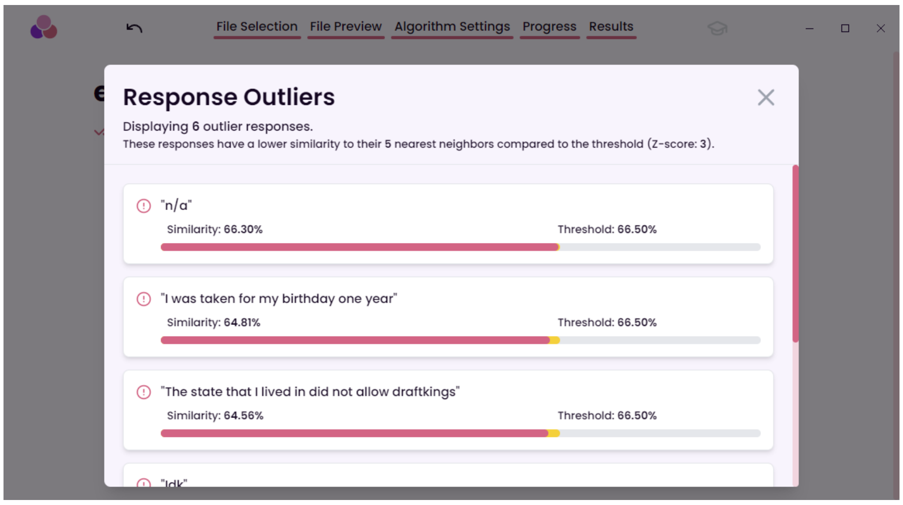
This modal is designed to provide users with details about the structure of the clusters, their similarities, and the responses that represent each cluster.

Upon opening the modal, the user is prompted to select a primary cluster. This can be done either by entering the cluster ID (starting from 0) or by searching for response content and selecting a cluster that contains the desired content.

Once the primary cluster is selected, a secondary cluster selection option becomes available for comparison.

Additionally, the five most similar clusters to the primary cluster are displayed at the bottom of the modal. Displaying these clusters at the bottom allows users to quickly identify which clusters are most similar without having to navigate through all the clusters. This feature is particularly useful for assessing how close a pair of clusters was to merging and for adjusting the merging similarity threshold in a subsequent run to facilitate their merger.

The user interface is designed to present options in a step-by-



Response Outliers

Displaying 6 outlier responses.
These responses have a lower similarity to their 5 nearest neighbors compared to the threshold (Z-score: 3).

"n/a"
Similarity: 66.30% Threshold: 66.50%

"I was taken for my birthday one year"
Similarity: 64.81% Threshold: 66.50%

"The state that I lived in did not allow draftkings"
Similarity: 64.56% Threshold: 66.50%

"Idk"

Figure 3.11: Response Outliers

step manner, allowing the user to make decisions without feeling overwhelmed by too many options.

The non-primary clusters are displayed with their similarity value, expressed as a percentage, relative to the primary cluster. All displayed clusters can be expanded to reveal the five most representative responses, along with their similarity to the cluster center, also expressed as a percentage.

Cluster-related information is visually distinguished from response-related information by the use of color, enhancing the clarity and ease of interpretation.

Outliers

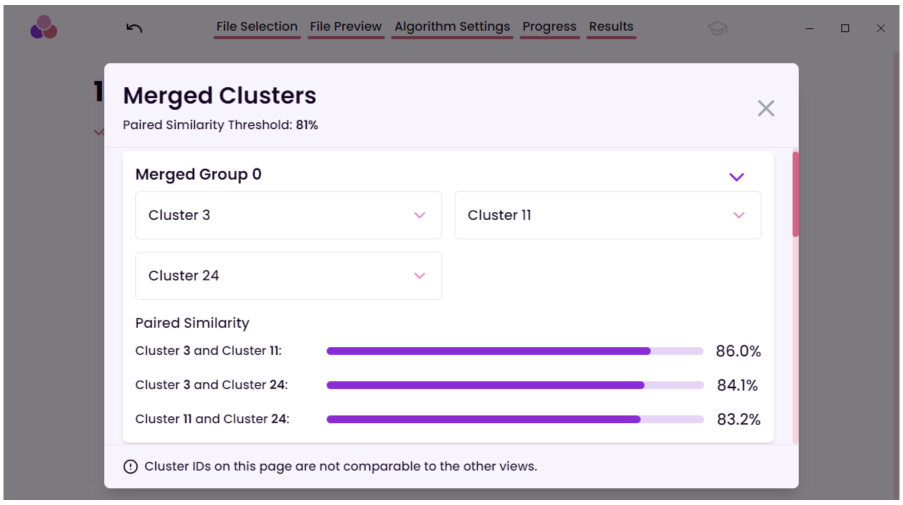
The outliers modal (see Figure 3.11) displays the number of responses marked as outliers and excluded from the analysis. Each response is shown with its similarity value to its nearest neighbors, relative to the threshold.

The purpose of this modal is to enable users to quickly identify abnormal or unique responses and provide sufficient information to make informed decisions about tuning the outlier detection settings.

The total number of outliers is displayed at the top, along with the run-specific outlier detection settings.

The main display consists of a list showing each outlier, the text of the response, and the similarity to the nearest neighbors as a percentage relative to the threshold. The difference between the similarity and the threshold is highlighted in yellow on the bar, indicating whether a response was barely considered an outlier or clearly an outlier. This visualization helps users determine how to adjust the outlier detection z-score threshold.

Users can also identify if the algorithm has recognized any responses as false outliers, potentially due to spelling errors or foreign words. This information allows users to manually correct these values in the output file if necessary.



Merged Clusters
Paired Similarity Threshold: 81%

Merged Group 0

Cluster 3
Cluster 11
Cluster 24

Paired Similarity
Cluster 3 and Cluster 11: 86.0%
Cluster 3 and Cluster 24: 84.1%
Cluster 11 and Cluster 24: 83.2%

Cluster IDs on this page are not comparable to the other views.

Figure 3.12: Merged Clusters

Merged Clusters

The merged clusters modal (see Figure 3.12) displays each cluster merger along with the similarity values between the merged clusters.

The objective of this modal is to indicate which clusters were merged, their similarity values, and the responses representing each merged cluster, thereby allowing the user to assess whether the current merging settings are appropriate or need adjustment.

The paired similarity threshold setting for merging clusters in the current run is displayed at the top.

The main display consists of a list of cluster mergers. Each merged cluster in the list can be expanded to view the five most representative responses, and the paired similarity between every cluster in the merger is displayed which enables the user to evaluate how close the merger was to not occurring based on the current merging threshold.

Including the most representative responses for each cluster aids the user in visualizing the nature of the responses each cluster represents. This visualization facilitates intuitive judgments on whether the merger was appropriate and whether the merging settings should be adjusted for a subsequent run.

The cluster IDs displayed in this modal do not necessarily match those in the rest of the results, as the cluster IDs are adjusted post-merger.

3.3 CLUSTERING

The clustering is handled by a Python script (see Figure 3.13) created by Electron's main process. The user settings are passed to the script via command-line arguments, based on which, the script publishes[4] a to-do list of steps that will be performed.

4 To publish refers to sending some information to the main process, more details in section 3.4

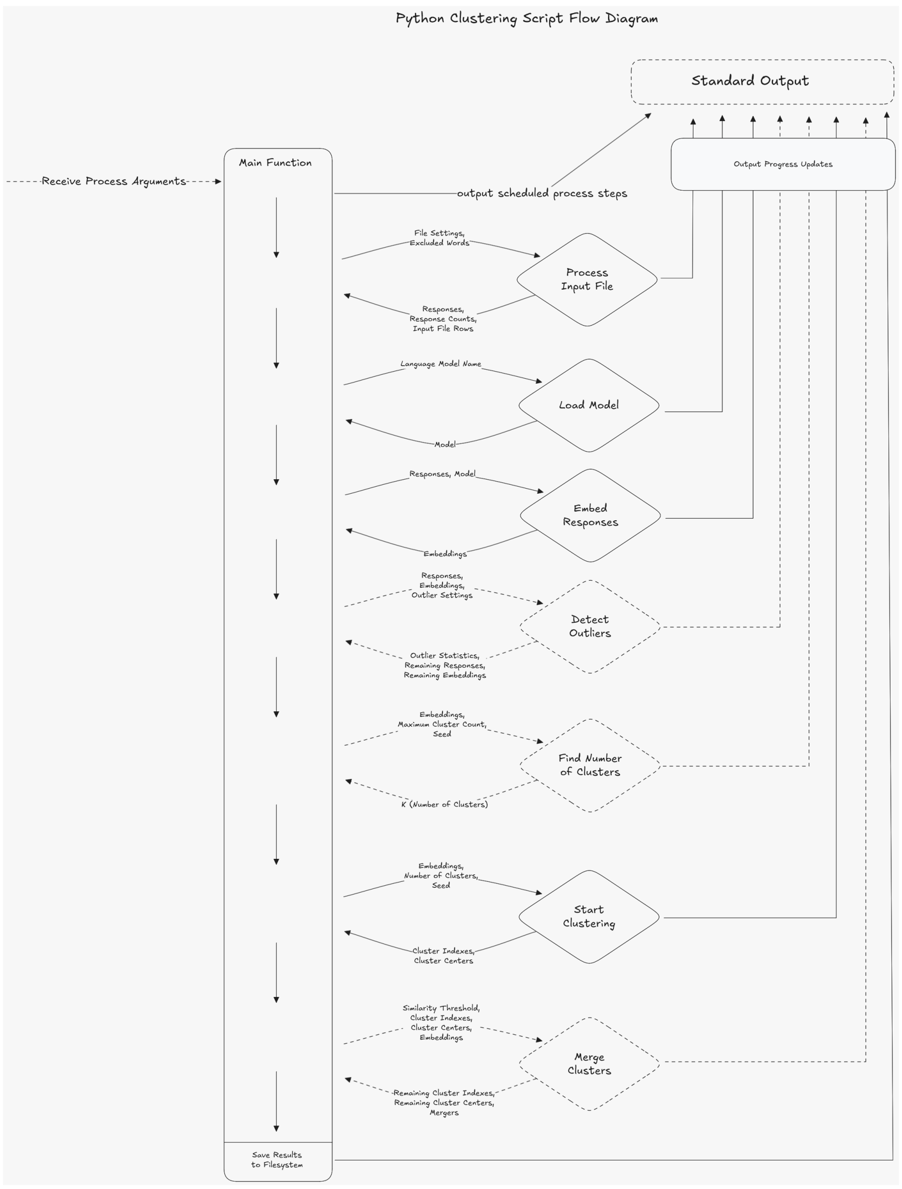

Python Clustering Script Flow Diagram
Standard Output
Output Progress Updates
Receive Process Arguments
Main Function
output scheduled process steps
File Settings, Excluded Words
Process Input File
Responses, Response Counts, Input File Rows
Language Model Name
Load Model
Model
Responses, Model
Embed Responses
Embeddings
Responses, Embeddings, Outlier Settings
Detect Outliers
Outlier Statistics, Remaining Responses, Remaining Embeddings
Embeddings, Maximum Cluster Count, Seed
Find Number of Clusters
K (Number of Clusters)
Embeddings, Number of Clusters, Seed
Start Clustering
Cluster Indexes, Cluster Centers
Similarity Threshold, Cluster Indexes, Cluster Centers, Embeddings
Merge Clusters
Remaining Cluster Indexes, Remaining Cluster Centers, Mergers
Save Results to Filesystem

Figure 3.13: Clustering Flow Diagram

For each step, the script sends a progress update to the main process when the step has started and when it has been completed. The main process updates the progress display based on these published updates.

The script is based on a Python implementation used by Morgenroth et al. (2024, [Mor+24]).

This section provides a detailed explanation of the steps involved in the clustering process.

3.3.1 *Input File Processing*

The initial step involves processing the input file to extract all responses relevant for the analysis.

Based on the selected file settings, the CSV file is parsed. Responses containing words from the excluded words list (case-insensitive) are ignored, while the others are retained.

Keeping track of the frequency of each unique response is essential for creating sample weights for the clustering algorithm.

The complete CSV file rows are also retained for recreating the input file and updating it with the cluster indices.

3.3.2 *Model Loading*

The model loading step involves loading the selected language model into memory and managing its download if it is not yet present in the cache directory. This step aims to ensure that the model is in memory and ready for inference.

The model name is retrieved from the advanced algorithm options. The system searches for the model in the Huggingface[5] cache directory. If the model is not found, it is downloaded from the Huggingface servers. Once the model is available on disk, it is then loaded into memory.

The default model should already be downloaded at this point, so no download would be necessary (see Section 3.5).

This step is isolated due to the potential duration of the model download. Combining this step with the response embedding step could lead users to mistakenly believe that the embedding process is excessively time-consuming.

3.3.3 *Embedding the Responses*

Once the model is ready for inference, the responses can be embedded. The objective is to transform each response into a fixed-length high-dimensional vector, where the dimensionality corresponds to the output layer dimensionality of the embedding model.

The list of responses is sorted by length and then divided into batches of 32. Each batch is tokenized by the model into a dictionary containing tensors for input IDs, attention masks, and token type IDs. The tokenized batch is then fed into the model to generate embeddings.

The resulting embeddings are normalized to unit length to ensure uniform vector magnitudes, which simplifies the cosine similarity computation to a dot product calculation. Furthermore because k-means uses the Euclidean distance measure, it is important that the embeddings are normalized to achieve a proxy measure for the cosine distance.

5 huggingface.co

3.3.4 *Outlier Detection*

Outlier detection is a crucial step in data analysis, aiming to identify and remove responses that deviate significantly from the norm.

The outlier detection step computes the cosine similarity between all embeddings and their nearest neighbors and then excludes responses that fall beyond a specified threshold, measured in standard deviations from the mean value.

Let N be the number of responses, k the number of nearest neighbors to consider per data point, and z the z-score threshold (i.e., the threshold number of standard deviations from the mean value).

Given that the embeddings are normalized, the cosine similarity matrix can be derived by computing the dot product of the embedding tensor with its transpose.

To identify outliers, the following steps are undertaken:

1. Calculate $\mu_{k,i}$ for $i \in 1, \ldots, N$: This represents the mean similarity of data point i to its k nearest neighbors.

2. Determine the overall mean μ_k and standard deviation σ_k of the mean similarities.

3. Compute the threshold: $\hat{\mu_k} = \mu_k - z \cdot \sigma_k$

4. Remove outliers: Exclude all data points i for which $\mu_{k,i} < \hat{\mu_k}$

This approach is simple, fast to calculate, and easy for users to interpret. This step can be disabled or adjusted by tuning k and z in the advanced algorithm options.

Which responses were excluded and how their average neighbor similarity compares to the threshold is documented for the outlier result display.

3.3.5 *Determining the Number of Clusters*

This section outlines the approach taken to identify the optimal K by employing the silhouette score and the Bayesian Information Criterion (BIC) as evaluation metrics.

The goal is to find the optimal value of K based on a combination of cluster evaluation measures.

To begin, a range of potential values for K must be specified. Since a greater value of K slows down the clustering algorithm, the step size between each value increases as K exceeds certain thresholds. The maximum value for K is defined as

$$K_{\text{max}} = \min\left(K_{\text{usermax}}, \frac{N}{2}\right),$$

where K_{usermax} is the maximum number of clusters, potentially set by the user in the algorithm options, and N is the number of responses.

For each K value, the k-means clustering of the normalized embeddings is computed, using k-means++ initialization (with the run's deterministic seed) and the previously extracted sample weights. Each clustering result is evaluated using the silhouette score and a simplified BIC[6].

The resulting scores are normalized, and the BIC scores are inverted, such that the best K maximizes the product of the silhouette score and the inverted BIC score.

Users can also choose to skip this step by disabling it and providing a specific value of K in the algorithm options.

A plot showing the normalized scores and the optimal compromise is saved to the results directory.

3.3.6 *Clustering*

Once the value for K has been finalized, the main clustering process can be completed. The normalized embeddings are clustered using k-means with the chosen value of K, employing k-means++ initialization, the deterministic seed, and the sample weights. The resulting cluster centers are subsequently normalized.

The objective is to partition the embeddings into K groups, each consisting of embeddings with similar semantics.

The deterministic seed is used to maintain the random state of the algorithm, ensuring reproducibility across multiple runs. The sample weights are applied to give more importance to frequently occurring responses during the clustering process.

After clustering, the cluster centers are normalized, which facilitates the calculation of the between-cluster similarity using the dot product.

3.3.7 *Cluster Merging*

Because the number of clusters suggested by the cluster evaluation measures is often quite large for word or sentence embeddings, an additional cluster merging step can be performed.

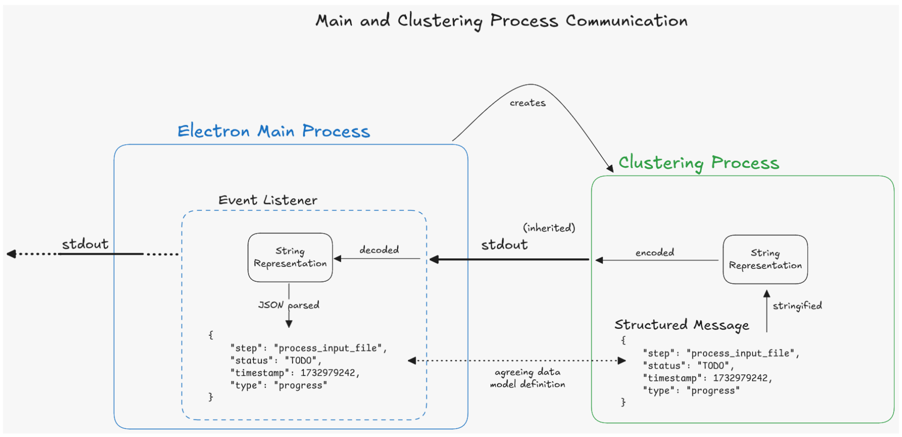
In this merging step, agglomerative meta-clustering is applied to merge clusters that have a similarity greater than the user-selected threshold. The goal is to combine clusters that represent the same or very similar themes.

The cluster centers from the initial k-means clustering are used as the samples for the agglomerative meta-clustering.

Let $s_{\max}$ denote the similarity threshold for clusters, as chosen by the user. The minimum distance, $d_{\min}$, between clusters that will not be merged is then defined as

$$d_{\min} = 1 - s_{\max}.$$

6 We use a simplified BIC that approximates the penalty term as just K, as otherwise scaling it by the number of data points would cause the penalty term to dominate.



```
{
    "step": "process_input_file",
    "status": "TODO",
    "timestamp": 1732979242,
    "type": "progress"
}
```

```
{
    "step": "process_input_file",
    "status": "TODO",
    "timestamp": 1732979242,
    "type": "progress"
}
```

Figure 3.14: Main Process - Clustering Process Communication

The agglomerative clustering is performed using complete linkage, where all cluster centers closer than d_{min} are merged.

By setting the metric of the agglomerative clustering to cosine distance, the final similarity values between clusters align with the user-defined merging threshold. If another metric were used, clusters with a similarity greater than the merging threshold might remain in the final clustering without being merged.

After merging, the remaining cluster centers are re-centered to the weighted mean of their corresponding points and normalized to unit length. The re-centering step is crucial for identifying the most representative responses for each merged cluster, as these are the closest to the new cluster center. Normalizing the cluster centers ensures that the final cluster similarities can be calculated using the dot product.

The list of merged clusters and their paired similarities is saved to the results directory.

3.3.8 *Results Saving*

The final step involves saving all collected results in separate files within the results directory. The goal of this step is to persist all relevant information about the run, allowing for result analysis within the application and enabling the user to re-run the clustering process with slightly modified parameters.

A run name is automatically generated based on the input file name and the current timestamp. This name is then published to the Electron main process to ensure consistency across the application.

For a complete list of the saved artifacts, please refer to Section 3.1.

3.4 COMMUNICATION BETWEEN MAIN AND CLUSTERING PROCESS

Communication from the clustering process to the main process (see Figure 3.14) plays a crucial role in keeping the user informed about the clustering progress, as well as in detecting when the run is completed or when an error occurs.

The goal of this section is to establish a mechanism for sending structured messages from the clustering process to the main process, where they can be received, parsed, and handled appropriately.

The standard output (stdout) serves as the default output stream across operating systems. By leveraging this existing functionality, there is no need to create additional communication channels such as pipes or sockets.

Since the main process initiates the clustering process, it can attach an event handler to the clustering process's stdout, which is inherited from the main process. This event handler receives data from the buffer, which can be decoded into a string.

The clustering process can print a string to stdout, which is typically line-buffered. To ensure that the buffer is emptied for each individual message, the stream should be explicitly flushed after printing.

To enable the transmission of structured messages (beyond simple strings), both the clustering and main processes must define a shared message model, which can be serialized into a JSON string.

The full procedure is as follows:

1. The clustering script initializes a message model containing the relevant data to be sent, such as the current step, status, and timestamp.
2. This model is then serialized into a JSON string.
3. The string is printed to the standard output stream, and the stream is flushed to ensure the message is transmitted immediately.
4. The main process's event handler reads the standard output stream and receives the data.
5. The data is decoded into a string.
6. The string is parsed into a JavaScript object, and the data is typed according to the message model.
7. The attributes of the typed JavaScript object can now be safely accessed, and the message content can be processed accordingly.

3.5 FIRST-LAUNCH EXPERIENCE

When the application is first launched, the default embedding model is downloaded before the app's main window is displayed. The goal

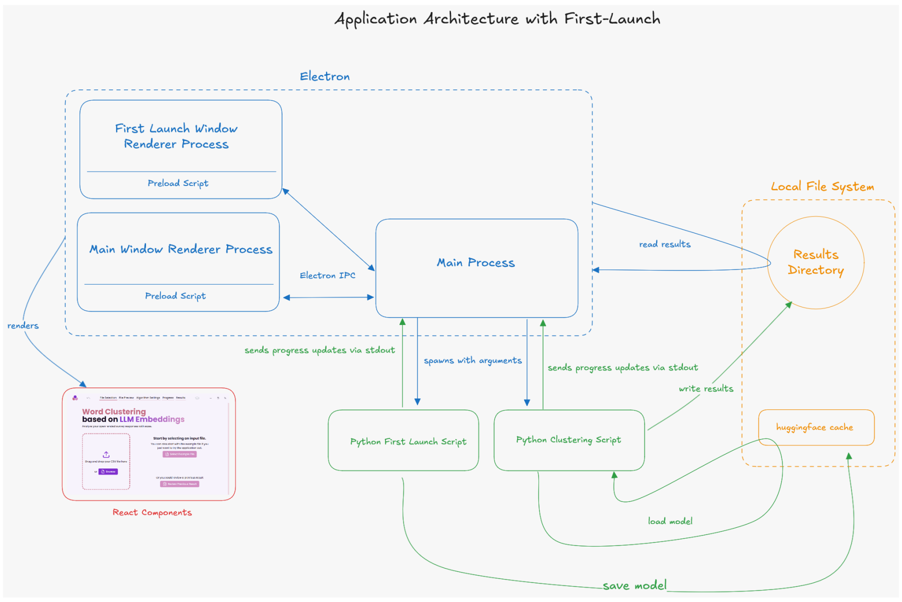
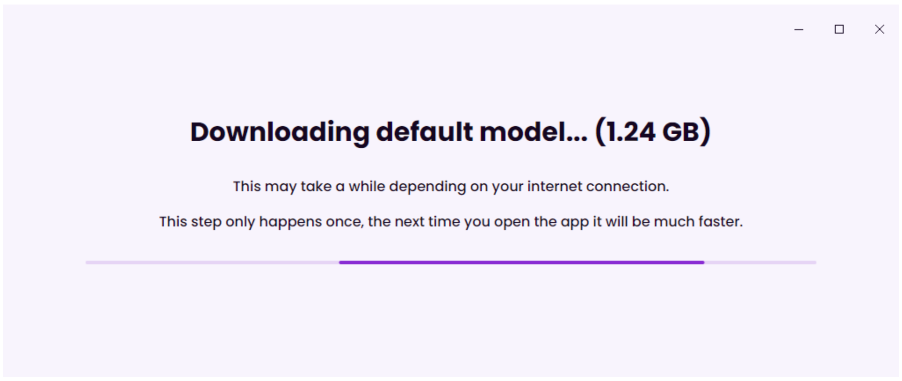

Application Architecture with First-Launch
Electron
First Launch Window Renderer Process
Preload Script
Main Window Renderer Process
Preload Script
Electron IPC
Main Process
renders
React Components
Word Clustering based on LLM Embeddings
sends progress updates via stdout
spawns with arguments
sends progress updates via stdout
Python First Launch Script
Python Clustering Script
read results
write results
Local File System
Results Directory
huggingface cache
load model
save model

Figure 3.15: Application Architecture including First Launch


Downloading default model... (1.24 GB)

This may take a while depending on your internet connection.

This step only happens once, the next time you open the app it will be much faster.

Figure 3.16: First Launch Window

is to ensure that the user has the default embedding model installed prior to running the clustering algorithm for the first time.

This feature was especially important in relation to the user study, as it ensures that the required model download time is handled before the user's appointment. It is considered a poor user experience to have the clustering process take longer than expected during the first run, as it could give a misleading impression of the overall speed of the application.

For an overview of the application architecture including the first launch modules, see Figure 3.15.

Upon the user's first launch, a special window (see Figure 3.16) is displayed, informing the user about the download process. This window shows the size of the default model and emphasizes that this step is a one-time exception.

An indeterminate loading bar is used to indicate that the program is not stuck and that the download is in progress. The user has the option to interrupt the process at any time, with progress being saved so that the download can resume from where it left off when the application is launched again.

The main process initiates a special first-launch Python script, downloads the model. Upon completion, the script sends a message to the main process (see Section 3.4). The model is downloaded to the Hugging Face cache directory, where the clustering script will later check for it during the model loading step.

4 EXPERIMENTS

4.1 USER STUDY

To evaluate whether the developed application was satisfyingly intuitive and user-friendly, a user study evaluating the perceived usability of the application was conducted.

The user study involved $n = 3$ participants, all of whom were researchers in the social sciences, therefore representing the target user base.

Two participants were recruited directly and social media was used to find other interested individuals, who were then contacted via email. Information materials provided an overview of the application, its functionality, and data confidentiality measures. Unfortunately only one more person was able to participate.

The Bielefeld University Ethics Review Board reviewed the study application in accordance with the ethical guidelines of the German Association of Psychology (Deutsche Gesellschaft für Psychologie; DGPs), which align with those of the American Psychological Association (APA). Upon review, the Ethics Review Board deemed the study to be ethically appropriate.

Participants were also given material with pre-meeting instructions, detailing how to download and install the application, how to deal with potential obstructions and how to completely download the default embedding model.

The dataset to be clustered in the study was reduced from the application's example dataset to 100 responses.

The study was held through online conferencing and participants could ask questions or give feedback during the entire study.

The study was divided into three parts:

- **Manual Coding**: Participants were instructed to manually assign codes (cluster indexes) to the responses for 10 minutes. The goal was to familiarize participants with the dataset and to offer a direct comparison between the effort to manually code the data and the effort for the automatic coding with the application.

- **Automatic Coding with the Application**: Participants were guided through the application using the instruction material. They were instructed on how to cluster the dataset using the application and were then guided through the results analysis within the application. Finally, they were shown how to rerun a result with different algorithm settings. This section was intended to show off the complete functionality of the application on a running example. Participants were also indirectly shown the effort required for automatic coding with the application, even as a first time user.

- **Survey**: Participants completed a final feedback form containing the ten positive SUS items. They were also asked three additional items (with response options ranging from strongly disagree to strongly agree, just like in the SUS) aimed at evaluating the perceived applicability of the application to the researcher's personal coding process:
  - "I would use the application to assist with coding large datasets that are tedious to code manually."
  - "I could save time by integrating the application into my coding workflow."
  - "I found that most of the clustering decisions by the application were reasonable."

  There was also a free-text response option for additional feedback on the application.

4.1.1 *Results*

The mean SUS score was 93.33, and all three participants strongly agreed with the three additional items regarding the perceived applicability of the application.

The positive feedback, received from the form or during the study, highlighted the application's beautiful interface, intuitive design, smooth functionality, useful visualization of clustering progress, speed compared to the manual process, and the appreciated cluster similarity comparison with representative responses.

The following information about desired features was gathered:

- More information about accepted input file types and structure, particularly regarding potentially infesting responses.
- More detailed information in the settings, especially concerning the role of the deterministic seed.
- Functionality for automatic suggestions for data-based cluster names with an option to edit (and persist) these names.
- Information about which response corresponds to which participant, including how many participants were coded at least once per cluster.
- Soft cluster assignments, especially when considering longer responses.

One issue encountered during the study was a problem with saving run names, which did not work as expected on 2 researcher's devices.

4.1.2 *Interpretation*

Based on the curved grading scale by Sauro and Lewis, the application receives a grade of A+ in perceived usability. The extraordinarily high perceived usability grade and strong agreement from all three participants regarding the additional items suggests initial positive perceptions of the application and is an encouraging sign that the application may be addressing user needs effectively.

However the small sample size means that these findings may not be representative of the broader user base. Sauro (2013, [Sau13]) discussed in a blog post the minimum acceptable sample size for the SUS, considering 5 users to be the minimal sample size for reporting results. Furthermore such a high SUS score has to be examined more critically. Sauro (2011a, [Sau11]) found that in a review of 241 studies, the total number of mean SUS scores above 90 was only two. The probable explanation lies in the small sample size, but another possibility for the potentially inflated score is that participants were not entirely neutral in their assessment. An asynchronous study, in which participants try out the applications purely based on the instruction material, may produce more neutral responses. The selection process for participants may have also biased the outcome of the study, as direct recruitment may lead to selecting individuals who are already familiar with or favorably disposed towards the researchers or the application.

While the sample size of the user study is not large enough to draw definitive conclusions about the perceived usability of the general target userbase, these initial positive responses should inspire further research and development. They suggest that it might be worthwhile to gather more data to validate these findings.

The desired features feedback should also be used to guide further development.

5 CONCLUSION

For this work, an application was developed for the automatic coding of responses to open-ended questions using a clustering approach based on state-of-the-art embedding models. The application was designed specifically for non-technical users, with an emphasis on an intuitive user interface that minimizes complexity. Given that researchers constitute the primary user base, data confidentiality was prioritized as a non-negotiable requirement. Consequently, the application was designed such that all computations were done using on-device models, ensuring that sensitive data remains secure.

The application was structured around an Electron framework, with a Python subprocess handling the clustering computations.

To evaluate the application, it was tested by three researchers who assessed its perceived usability using the positive SUS. The application achieved an exceptionally high mean SUS score of 93.33, suggesting outstanding perceived usability. However, this result raises concerns about potential biases, emphasizing the need for further evaluation with a larger sample size to confirm the findings.

5.1 LIMITATIONS

Despite the promising potential of the application, several limitations and areas for improvement have been identified, which should be addressed to enhance its functionality and usability.

Besides, the interpretation of the user study results remains inconclusive due to the small sample size. Both the perceived usability and the applicability of the application to the researchers' workflow should be examined with a larger sample to obtain more robust conclusions.

Several shortcomings are present in the current version of the application:

1. Due to limited available data, the non-configurable settings of the clustering algorithm have not been empirically validated. While the results for the example dataset appear reasonable, there is no assurance that these settings will perform consistently across different input files. In particular, input files with varying response lengths may produce unexpected outcomes.

2. The application contains several unhandled error scenarios. For instance, input files with foreign characters will break the application. Overall, the error handling is not user-friendly, as users are often only notified that an error has occurred, with the only suggestion being to consult the log file for more details.

3. The current cluster evaluation measures are neither explained to the user nor configurable. Additionally, the effectiveness of these measures has not been validated.

4. Saving results to multiple files and using the directory name as the run identifier has proven to be unreliable, leading to issues during the user study. A more robust result management system should be implemented to ensure consistency.

5. Currently, the application is unavailable for macOS, limiting its accessibility to a wider range of researchers.

5.2 FUTURE WORK

The future development of the application presents many opportunities for enhancement, driven by the feedback collected from the study participants.

- The current system for managing saved results requires significant improvement. A future version could implement a database, such as SQLite, to store results in a more structured manner. This would allow for advanced querying capabilities and improve between-run analysis, offering greater flexibility for exploring and comparing previous results.
- The application could benefit from the integration of dimensionality-reduction based visualization options, such as PCA, to explore high-dimensional clustering results. Additionally, the optimal cluster count evaluation plot could be directly incorporated into the interface.
- Enhancements are needed in the input file processing pipeline to increase robustness, particularly in handling malicious inputs and different text encodings. These improvements would ensure that the application can handle a wider range of input data without crashing or producing incorrect results.
- Expanding the algorithm's settings would allow for the use of alternative clustering approaches, such as Gaussian Mixture Models (GMMs), which provide soft cluster assignments. This would enable users to assign a response to multiple clusters, offering more flexibility in handling ambiguous or multi-faceted responses.
- The application could benefit from the integration of large language models in additional areas, such as automatically extracting editable and persistent theme names for clusters based on their associated responses. This would streamline the process of understanding and labeling clusters, improving the overall user experience.
- Including participant information as part of the responses could provide further opportunities for analysis. This would enable researchers to segment responses based on participant characteristics and uncover patterns related to different demographic or experimental groups.

- A version of the application for macOS should be developed and made available to a wider user base. This would increase the accessibility of the tool for researchers working on different operating systems, facilitating a broader adoption.

5.3 END

In conclusion, the application developed as part of this work received a highly positive initial response from users, but further research is needed to confirm this result and further development is needed to expand and improve its functionality.

BIBLIOGRAPHY

[AV07] David Arthur and Sergei Vassilvitskii. “k-means++: the advantages of careful seeding”. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. ISBN: 9780898716245.

[AZ21] Nico Andersen and Fabian Zehner. “shinyReCoR: A Shiny Application for Automatically Coding Text Responses Using R”. In: *Psych* 3.3 (2021), pp. 422–446. ISSN: 2624-8611. DOI: 10.3390/psych3030030. URL: https://www.mdpi.com/2624-8611/3/3/30.

[AZG23] Nico Andersen, Fabian Zehner, and Frank Goldhammer. “Semi-automatic coding of open-ended text responses in large-scale assessments”. In: *Journal of Computer Assisted Learning* 39.3 (2023), pp. 841–854. DOI: https://doi.org/10.1111/jcal.12717. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jcal.12717. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jcal.12717.

[Bet+14] Antje Bethmann et al. “Automatic coding of occupations: Using machine learning algorithms for occupation coding in several German panel surveys”. In: *Beyond traditional survey taking: adapting to a changing world. Proceedings of Statistics Canada Symposium 2014*. Ed. by Statistics Canada. Statistics Canada, 2014, pp. 1–6.

[Bez81] James Bezdek. *Pattern Recognition With Fuzzy Objective Function Algorithms*. Springer International Publishing, Jan. 1981. ISBN: 978-1-4757-0452-5. DOI: 10.1007/978-1-4757-0450-1.

[BH65] Geoffrey H. Ball and David J. Hall. *ISODATA, a novel method of data analysis and pattern classification*. Technical report NTIS AD 699616. Stanford, CA: Stanford Research Institute, 1965.

[Boj+17] Piotr Bojanowski et al. *Enriching Word Vectors with Subword Information*. Originally published in 2016. 2017. arXiv: 1607.04606 [cs.CL]. URL: https://arxiv.org/abs/1607.04606.

[Bou+15] Athman Bouguettaya et al. “Efficient agglomerative hierarchical clustering”. In: *Expert Systems with Applications* 42.5 (2015), pp. 2785–2797. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.09.054. URL: https://www.sciencedirect.com/science/article/pii/S0957417414006150.

[Bow+15] Samuel R. Bowman et al. “A large annotated corpus for learning natural language inference”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Ed. by Lluís Màrquez, Chris Callison-Burch, and Jian Su. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 632–642. DOI: 10.18653/v1/D15-1075. URL: https://aclanthology.org/D15-1075.

[Bro95] John Brooke. “SUS: A quick and dirty usability scale”. In: *Usability Eval. Ind.* 189 (Nov. 1995).

[Cer+18] Daniel Cer et al. *Universal Sentence Encoder*. 2018. arXiv: 1803.11175 [cs.CL]. URL: https://arxiv.org/abs/1803.11175.

[Con+18] Alexis Conneau et al. *Supervised Learning of Universal Sentence Representations from Natural Language Inference Data*. 2018. arXiv: 1705.02364 [cs.CL]. URL: https://arxiv.org/abs/1705.02364.

[Dev+19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Originally published in 2018. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[Dri+04] P. Drineas et al. “Clustering Large Graphs via the Singular Value Decomposition”. In: *Machine Learning* 56.1 (July 2004), pp. 9–33. ISSN: 1573-0565. DOI: 10.1023/B:MACH.0000033113.59016.96. URL: https://doi.org/10.1023/B:MACH.0000033113.59016.96.

[Dun73] J. C. Dunn. “A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters”. In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57. DOI: 10.1080/01969727308546046. eprint: https://doi.org/10.1080/01969727308546046. URL: https://doi.org/10.1080/01969727308546046.

[FD17] Erin D Foster and Ariel Deardorff. “Open Science Framework (OSF)”. In: *J. Med. Libr. Assoc.* 105.2 (Apr. 2017).

[Gef23] Anton Gefvert. “Text Curation for Clustering of Free-text Survey Responses”. MA thesis. Linköping University, Department of Computer and Information Science, 2023, p. 54.

[GS03] Daniela Giorgetti and Fabrizio Sebastiani. “Automating survey coding by multiclass text categorization techniques”. In: *Journal of the American Society for Information Science and Technology* 54.14 (2003), pp. 1269–1277. DOI: https://doi.org/10.1002/asi.10335. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.10335. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.10335.

[GS23] Hyukjun Gweon and Matthias Schonlau. “Automated Classification for Open-Ended Questions with BERT”. In: *Journal of Survey Statistics and Methodology* 12.2 (May 2023), pp. 493–504. ISSN: 2325-0992. DOI: 10.1093/jssam/smad015. URL: http://dx.doi.org/10.1093/jssam/smad015.

[HS20] Zhoushanyue He and Matthias Schonlau. “Automatic Coding of Open-ended Questions into Multiple Classes: Whether and How to Use Double Coded Data”. In: *Survey Research Methods* 14.3 (Aug. 2020), pp. 267–287. DOI: 10.18148/srm/2020.v14i3.7639. URL: https://ojs.ub.uni-konstanz.de/srm/article/view/7639.

[IDS15] RUMEN ILIEV, MORTEZA DEHGHANI, and EYAL SAGI. “Automated text analysis in psychology: methods, applications, and future developments”. In: *Language and Cognition* 7.2 (2015), pp. 265–290. DOI: 10.1017/langcog.2014.30.

[Kim+23] Hyoun S. Kim et al. “Self-Generated Motives of Social Casino Gamers”. In: *Journal of Gambling Studies* 39.1 (Mar. 2023), pp. 299–320. ISSN: 1573-3602. DOI: 10.1007/s10899-022-10135-5. URL: https://doi.org/10.1007/s10899-022-10135-5.

[Kir+15] Ryan Kiros et al. *Skip-Thought Vectors*. 2015. arXiv: 1506.06726 [cs.CL]. URL: https://arxiv.org/abs/1506.06726.

[Leo90] Peter J. Rousseeuw Leonard Kaufman. “Partitioning Around Medoids (Program PAM)”. In: *Finding Groups in Data*. John Wiley & Sons, Ltd, 1990. Chap. 2, pp. 68–125. ISBN: 9780470316801. DOI: https://doi.org/10.1002/9780470316801.ch2. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470316801.ch2. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470316801.ch2.

[Llo82] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[Mac67] J MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability/University of California Press*. 1967.

[Mik+13a] Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. arXiv: 1310.4546 [cs.CL]. URL: https://arxiv.org/abs/1310.4546.

[Mik+13b] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[Mor+24] Thekla Morgenroth et al. "Dissecting Whiteness: consistencies and differences in the stereotypes of lower- and upper-class White US Americans". In: *Self and Identity* 23.1-2 (2024), pp. 70–94. DOI: 10.1080/15298868.2024.2322179. eprint: https://doi.org/10.1080/15298868.2024.2322179. URL: https://doi.org/10.1080/15298868.2024.2322179.

[Mue+23] Niklas Muennighoff et al. *MTEB: Massive Text Embedding Benchmark*. Originally published in 2022. 2023. arXiv: 2210.07316 [cs.CL]. URL: https://arxiv.org/abs/2210.07316.

[NBF21] Gandalf Nicolas, Xuechunzi Bai, and Susan T. Fiske. "Comprehensive stereotype content dictionaries using a semi-automated method". In: *European Journal of Social Psychology* 51.1 (2021), pp. 178–196. DOI: https://doi.org/10.1002/ejsp.2724. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ejsp.2724. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ejsp.2724.

[NBF22] Gandalf Nicolas, Xuechunzi Bai, and Susan T Fiske. "A spontaneous stereotype content model: Taxonomy, properties, and prediction". en. In: *J. Pers. Soc. Psychol.* 123.6 (Dec. 2022), pp. 1243–1263.

[PFB99] James Pennebaker, Martha Francis, and Roger Booth. "Linguistic inquiry and word count (LIWC)". In: (Jan. 1999).

[PM00] Dan Pelleg and Andrew W. Moore. "X-means: Extending K-means with Efficient Estimation of the Number of Clusters". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 727–734. ISBN: 1558607072.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[RG19] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: https://arxiv.org/abs/1908.10084.

[Sau11] Jeff Sauro. *A practical guide to the system usability scale: Background, benchmarks & best practices*. Measuring Usability LLC, 2011.

[Sau13] Jeff Sauro. *10 Things to Know About the System Usability Scale (SUS)*. Accessed: 2024-11-29. 2013. URL: https://measuringu.com/10-things-sus/.

[SC16] Matthias Schonlau and Mick P. Couper. "Semi-automated categorization of open-ended questions". In: *Survey Research Methods* 10.2 (Aug. 2016), pp. 143–152. DOI: 10.18148/srm/2016.v10i2.6213. URL: https://ojs.ub.uni-konstanz.de/srm/article/view/6213.

[Sch14] Malte Schierholz. *Automating survey coding for occupation*. FDZ-Methodenreport 201410 (en). Institut für Arbeitsmarkt- und Berufsforschung (IAB), Nürnberg [Institute for Employment Research, Nuremberg, Germany], 2014. URL: https://ideas.repec.org/p/iab/iabfme/201410(en).html.

[SGK17] Karl Severin, Swapna S. Gokhale, and Karthik C. Konduri. "Automated quantitative analysis of open-ended survey responses for transportation planning". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 2017, pp. 1–7. DOI: 10.1109/UIC-ATC.2017.8397567.

[SL11] Jeff Sauro and James Lewis. "When designing usability questionnaires, does it hurt to be positive?" In: *When designing usability questionnaires, does it hurt to be positive?* May 2011, pp. 2215–2224. DOI: 10.1145/1978942.1979266.

[SL16] Jeff Sauro and James R. Lewis. "Chapter 8 - Standardized usability questionnaires". In: *Quantifying the User Experience (Second Edition)*. Ed. by Jeff Sauro and James R. Lewis. Second Edition. Boston: Morgan Kaufmann, 2016, pp. 185–248. ISBN: 978-0-12-802308-2. DOI: https://doi.org/10.1016/B978-0-12-802308-2.00008-4. URL: https://www.sciencedirect.com/science/article/pii/B9780128023082000084.

[SS20] Malte Schierholz and Matthias Schonlau. "Machine Learning for Occupation Coding—A Comparison Study". In: *Journal of Survey Statistics and Methodology* 9.5 (Nov. 2020), pp. 1013–1034. ISSN: 2325-0984. DOI: 10.1093/jssam/smaa023. eprint: https://academic.oup.com/jssam/article-pdf/9/5/1013/41727199/smaa023.pdf. URL: https://doi.org/10.1093/jssam/smaa023.

[Ste+56] Hugo Steinhaus et al. "Sur la division des corps matériels en parties". In: *Bull. Acad. Polon. Sci* 1.804 (1956), p. 801.

[SWM23] Matthias Schonlau, Julia Weiß, and Jan Marquardt. *Multi-label classification of open-ended questions with BERT*. 2023. arXiv: 2304.02945 [stat.AP]. URL: https://arxiv.org/abs/2304.02945.

[Vie98] Peter Viechnicki. "A performance evaluation of automatic survey classifiers". In: *Grammatical Inference*. Ed. by Vasant Honavar and Giora Slutzki. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 244–256. ISBN: 978-3-540-68707-8.

[WNB18] Adina Williams, Nikita Nangia, and Samuel Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1112–1122. DOI: 10.18653/v1/N18-1101. URL: https://aclanthology.org/N18-1101.

[Xia+23] Shitao Xiao et al. *C-Pack: Packaged Resources To Advance General Chinese Embedding*. 2023. arXiv: 2309.07597 [cs.CL].

[ZSG16] Fabian Zehner, Christine Sälzer, and Frank Goldhammer. "Automatic Coding of Short Text Responses via Clustering in Educational Assessment". In: *Educational and Psychological Measurement* 76.2 (2016), pp. 280–303. DOI: 10.1177/0013164415590022. eprint: https://doi.org/10.1177/0013164415590022. URL: https://doi.org/10.1177/0013164415590022.