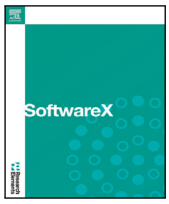
SoftwareX 31 (2025) 102258

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

AgriCode: Automated coding for qualitative research and its application to the valorization of agricultural residues

Maksim Koptelov [a,*], Jan Linck [b], Pierre Bisquert [a], Patrice Buche [a], Mathieu Roche [c]

[a] UMR IATE, Univ. Montpellier, INRAE, Institut Agro, 34060, France
[b] ECOZEPT, Freising, 85354, Germany
[c] CIRAD, UMR TETIS, F-34398 Montpellier, France

ARTICLE INFO

Dataset link: https://doi.org/10.57745/91BXIY

Keywords:
Text classification
Natural language processing
Data augmentation
Retrieval-augmented generation
Automatic coding
Qualitative research
Bioeconomy

ABSTRACT

Qualitative research, widely employed across various academic fields, explores phenomena using non-numerical data, with a particular focus on understanding the meanings, experiences, and perspectives of participants. In contrast to other type of research, it seeks to answer how, where, what, when and why individuals behave or respond in certain ways toward specific issues or topics. Qualitative research involves collecting and analyzing textual data, with interviews playing a central role in gathering expert knowledge. An essential part of data analysis is coding, using specially developed code system hierarchy that helps to categorize and organize responses and facilitates the retrieval of insights. Manual data coding is labor-intensive, and to automate this process we developed the AgriCode tool based on machine learning and manually annotated data. To address data scarcity and improve the prediction quality of our offline classifiers, we perform data augmentation using Retrieval-Augmented Generation (RAG), a state-of-the-art method originally designed for online Q&A systems. Our tool automates the coding of interview responses within the Horizon Europe Agriloop project, which focuses on agricultural waste in the food industry. AgriCode predicts a subset of a predefined code system hierarchy, assisting a human coder by accelerating the process and identifying errors in manual coding. Although initially designed for the valorization of agricultural residues, AgriCode's methodology can be adapted for any qualitative research domain characterized by data scarcity and the need of automated textual analysis. To achieve this, responses from the first round of interviews must be manually annotated using dedicated code system hierarchy. They can then be used for fine-tuning the model, while the RAG method can be employed to address the lack of data for certain classes.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00131
Permanent link to Reproducible Capsule	
Legal Code License	GPL-3.0 license
Code versioning system used	none
Software code languages, tools, and services used	Python 3.11
Compilation requirements, operating environments & dependencies	streamlit (1.38.0), transformers (4.44.2), sentencepiece (0.1.99), nltk (3.8.1), torch (2.1.2), numpy (1.24.3)
If available Link to developer documentation/manual	
Support email for questions	pierre.bisquert@inrae.fr

1. Motivation and significance

Qualitative research is a widely used method across various academic fields, including social sciences, market analysis, and more [1,2]. In contrast to applied and practical research, qualitative research seeks to answer how, where, what, when and why questions by exploring and explaining individuals' behavior toward specific matters [3].

* Corresponding author.
E-mail address: maksim.koptelov@inrae.fr (M. Koptelov).

https://doi.org/10.1016/j.softx.2025.102258
Received 26 February 2025; Received in revised form 26 June 2025; Accepted 2 July 2025
Available online 31 July 2025

The qualitative research methodology consists of two main components: data collection and data analysis. Interviews play a key role in data collection, allowing researchers to gather unwritten knowledge from experts on a particular topic. An essential part of data analysis in qualitative research is data coding, which not only helps to categorize and organize the data but also facilitates the retrieval of analytical insights [4]. Specifically, in the case of interviews, responses are coded according to specially defined categories, with similarly coded responses grouped together. This process simplifies subsequent tasks such as searching for specific information, summarizing results, comparing and contrasting arguments or finding a consensus [1].

Manual data coding requires significant human effort, making automation a particularly valuable solution. Several previous studies have explored this direction. The first steps toward automating coding were taken by [5], where the authors used a statistical approach to partially automate the coding of large volumes of free-form textual data. Later, [6] investigated semi-automatic coding, comparing a simple machine learning classifier based on bag-of-words with a rule-based method, both of which showed promise. The findings of [7] further confirmed the potential of ML-based approaches. As a follow-up, [8] developed an interactive coding system that combines predefined rules, supervised learning using logistic regression and user feedback. In contrast, [9] implemented an automatic prediction system based on *transfer learning*, where a classifier is pre-trained on a large corpus of linguistic data and fine-tuned on specific data for a particular task [10]. They employed the BERT (Bidirectional Encoder Representations from Transformers) model [11], which they fine-tuned on the manually coded interviews. Finally, popular qualitative data analysis software, such as MAXQDA [12], includes a built-in function for automatic coding, but it is limited to predefined patterns called search hits [13].

With the emergence of LLMs (Large Language Models), [14] explored their use to assist in deductive coding processes, demonstrating that models like GPT-3.5 can perform coding tasks with a level of agreement comparable to human coders. [15] proposed best practices (such as specific prompting strategies) for adapting traditional codebooks (i.e., sets of classes with definitions) for use with LLMs. [16] introduced the QualiGPT tool for qualitative coding. However, their evaluation was limited to existing datasets and synthetic data, without testing on real-world project data. The main limitation of the LLM based approach is its dependence on clearly defined classes, which may not lead to optimal performance with heterogeneous or ambiguous classes. Furthermore, LLMs lack transparency, reproducibility, and require API-usage fees, which is another constraint. In this work, we therefore implement an instance of [9] as a more traditional method that is not subject to these limitations. We adapt it to our setting with the focus on achieving more refined predictions at the sentence level and extend it with data augmentation using a state-of-the-art methodology based on an LLM.

The Agriloop project aims to address the significant issue of agricultural waste in the food industry, which generates a huge amount of agricultural waste such as harvested crops, industrial food waste, and processing residues like pomace and seeds. For example, tomato processing in Europe produces 500,000 tons of pomace annually [17], posing economic and environmental challenges. In the context of our project, we investigate the market readiness of Agriloop innovations for the B2B (business-to-business) sector. We follow the principles of qualitative research and conduct iterative, multi-stage interviews with experts from end-user groups and main value-chains. The interviewing process is designed using the Delphi method [18]. The objective of this methodology is to obtain a reliable consensus from the expert group after a certain number of rounds [19].

Once the interviews are collected, they are manually annotated, or *coded*, by our partner, Ecozept,[1] using a code system specifically developed for the project. These annotated interviews are used for collecting expert knowledge on the valorization of agricultural residues. For example, one research question explored is: "What are the main issues and challenges for the valorization of agricultural by-products?" The coded responses from the first round of interviews helped to identify criteria relevant to this question: transportation, limited seasonal availability of feedstock, legislative barriers, and others. To automate the interview coding process and assist our partner, we developed the AgriCode tool, which we present in this article.

To address the data limitations and enhance prediction quality, we implement data augmentation. In the literature, data augmentation is used to address lack of data [20] or improve class imbalance [21]. Text augmentation can be achieved by shuffling or removing words [22], replacing certain parts of speech with synonyms [20], and back translation [9]. While these methods introduce slight variations, they often produce semantic errors. LLMs can generate semantically similar text [23], but the resulting phrases may differ too much from the original data and lack important linguistic features. To improve the quality of LLM-generated text, Retrieval-Augmented Generation (RAG) [24] can be employed. RAG enriches a user query by incorporating external knowledge through contextual search and is widely used in online Q&A systems based on LLMs [24]. In this work, we employ RAG for data augmentation in *offline* classifiers (i.e., those operating outside of chatbot systems). This approach enables us to exploit data from a relevant project and fine-tune offline classifiers, which are less resource-intensive than LLMs, avoid API fees, and provide reproducible results.

In this article, we present the AgriCode tool, which features a graphical interface accessible online: https://ico.iate.inrae.fr/agricode. While developed for our project, the tool might also be useful for similar projects requiring coding of interviews on the valorization of agricultural residues. In such cases, a mapping between the project's code system and ours can be performed, allowing existing labels to be replaced in the tool interface. Additionally, our code can be adapted to develop similar systems for automating the coding of interviews or other textual data in any domain, which is a common practice in qualitative research. To achieve this, responses from the first round of interviews must be manually annotated according to a new code system. The new classifiers need to be fine-tuned using this annotated data, and our RAG-based method can be employed to augment underrepresented classes. The tool can then be used to annotate subsequent rounds of interviews.

2. Software description

This section provides details on the architectural composition and the different functionalities offered by AgriCode.

2.1. Software architecture

The workflow of AgriCode's backend is divided into four main phases: data preparation, data augmentation, learning, and prediction (Fig. 1). The following sections explain each phase, with the final section detailing the implementation and its interaction with the frontend.

2.1.1. Data preparation

Our data originates from the first round of interviews (R1) conducted as part of the Agriloop project. The data was first manually coded using the MAXQDA software and the specially developped code system (Fig. 2). Given the limited amount of annotated data (Tables B.2, B.3 in Appendix B), AgriCode currently predicts the first and second levels of the code system hierarchy, which correspond to 7 and 15 classes, respectively. The results of this step are the coded transcripts in CSV (Comma-Separated Values) format.

At the next step, the data is parsed to separate titles and paragraphs, and for the sentence classification method, paragraphs are further

[1] https://ecozept.com

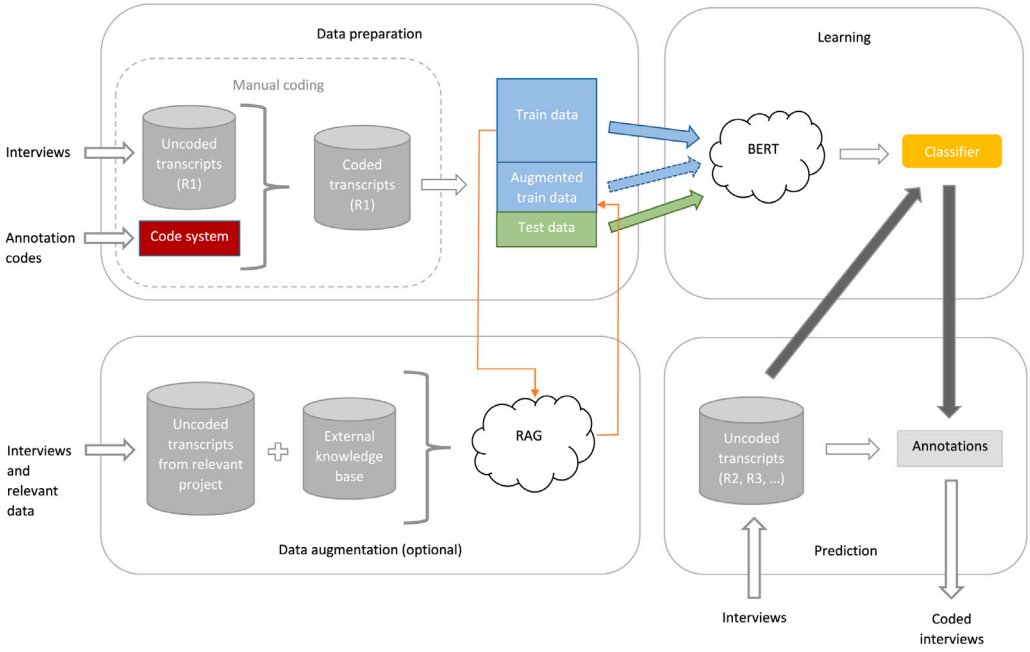


Fig. 1. The AgriCode workflow showing various phases and their interactions (R1, R2 and R3 refer to rounds 1, 2 and 3 of the Delphi method, respectively).

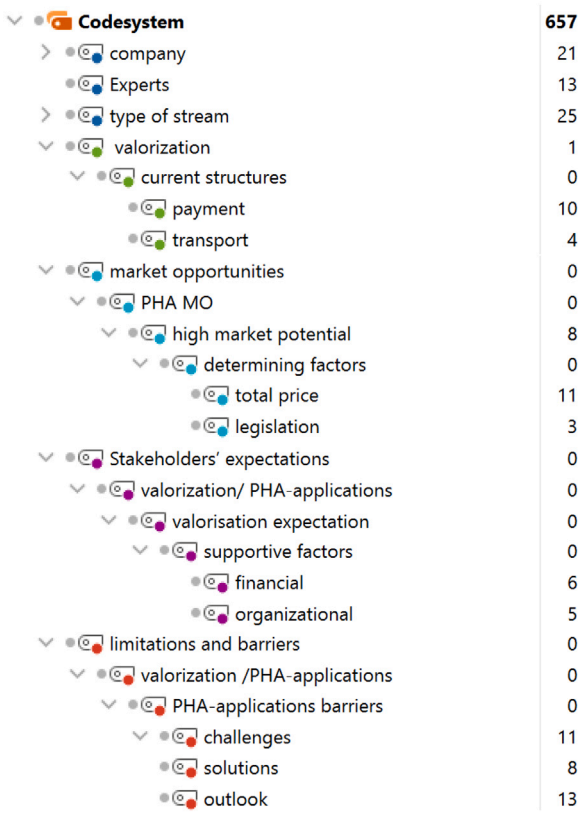


Fig. 2. An extract from the Ecozept's code system of R1, containing 145 codes. The values for unfolded categories (v) indicate the number of instances of each code without considering subcategories, and the values for folded categories (>) indicate the total number of instances for that code. The value in bold indicates the total number of instances.

divided into sentences. The annotated examples are then processed to avoid any overlap [25]. To achieve this, an example (whether a paragraph or sentence, depending on the setting) is removed if it is labeled with more than one class.

To improve prediction quality, titles (questionnaire name, block name, and question) are added to each paragraph or sentence, depending on the setting.

Finally, to evaluate our models, we use *stratified* sampling, implemented as follows: the data are split into two parts, with 80% of the segments used for learning and the remaining 20% used for validation. The split is performed in such a way that the proportion of training and testing examples for each class remains consistent.

This phase is implemented in the **data_loader_paragraphs.py** and **data_loader_sentences.py** files for paragraphs and sentences respectively.

2.1.2. Data augmentation

In this phase, we augment selected classes by generating artificial examples. Therefore, we improve performance of our approach by reducing the problem of imbalance between classes. Classes for augmentation are chosen experimentally based on low performance in the original (non-augmented) data. The selection process follows a specific methodology. We begin by augmenting classes whose performance needs improvement. We then evaluate whether this augmentation leads to performance gains for the targeted class without negatively affecting overall performance. Otherwise, we try different combinations and select the one that provides the highest overall performance (refer to Appendix C for examples). It is important to note that we augment only the training data, leaving the test data unchanged.

Regarding the choice of augmentation method, our preliminary experiments show that traditional methods such as replacing certain POS-tagged words with synonyms derived using BERT [26] do not provide sufficient improvement.[2] We refer to this approach as the *BERT-like method*. One limitation is its dependence on the vocabulary, and replacing certain parts of speech with synonyms does not give the expected performance. Therefore, a more intelligent method is

[2] For detailed results, please refer to *annex/survey_classification.xlsx* in our supplementary materials: https://forgemia.inrae.fr/maksim.koptelov/agricode.

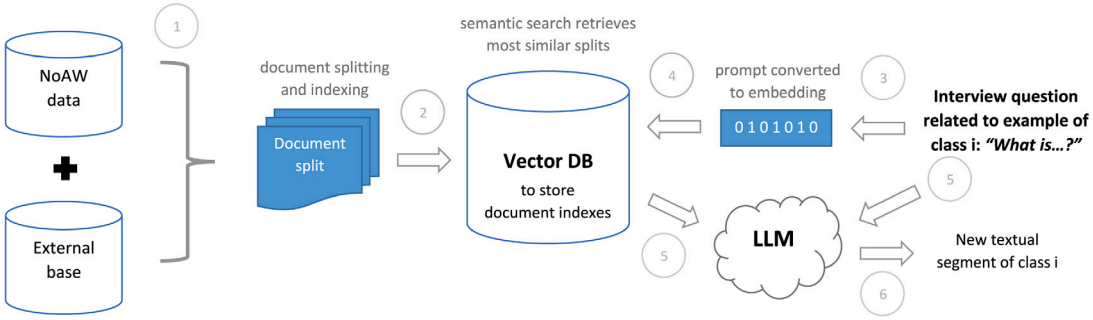


Fig. 3. The RAG workflow detailing various steps (indicated by circled numbers) of the data augmentation phase.

necessary. An *LLM-based approach*, specifically using ChatGPT in a few-shot learning setting (generating a new example based on 2–3 examples), also failed to deliver the expected results.[2] We believe that the new vocabulary introduced by ChatGPT is often unrelated to the context of our interviews, leading to more noise than improvement. More recent models, such as Llama [27], Mistral [28] and Qwen [29], most probably have the same issue, as the context of our project is very specific.

However, our data is well-structured and contains clearly formulated questions, allowing us to exploit external knowledge sources to improve generated data. Therefore, we employ RAG [24], a paradigm widely used in online Q&A systems based on LLMs. In the *RAG method*, external data are split into smaller documents (Step 1), which are then indexed and stored in a vector database (Step 2). Next, a user prompt, consisting of the interview question and the desired response class,[3] both taken from an original training example, is converted into an embedding (Step 3) and used in a semantic search to retrieve the most relevant document sections from an external knowledge source (Step 4). These retrieved documents are appended to the query and provided to an LLM as additional context (Step 5), allowing the model to generate a response[4] based on this enriched input (Step 6). In our workflow, Steps 1–2 are performed once to preprocess the data, while Steps 3–6 are repeated for each example in the training set (Fig. 3).

As an external source of knowledge, we use data from NoAW (No Agricultural Waste),[5] a relevant project in the agricultural domain. This data set contains transcripts of interview responses from different respondents and on different questions, but within the same context. The interviews are focused on the topic of Plastics in agriculture, specifically targeting producers, distributors, and end users of mulch films and biodegradable horticultural pots made from PHBV. There are slight variations in questions depending on these two applications, with an average of 14 questions for each application. There are responses from a total of 25 stakeholders, organized into 25 text files, with each file containing the role of the respondent and the application type, along with the questions and answers. The total word count for these files is 51044 words. This data is unannotated and comes in plain text format. In addition to the NoAW data, we also use the Agro Q&A data set [30], which contains 4153 question-response pairs, as another external knowledge base. We combine these two sources in our augmentation process, and according to our experiments, this approach provides the best performance in most settings.[2] The entire RAG process is applied only once to augment the training data, after which a classifier trained on this augmented data set is used offline.

Finally, we set the temperature parameter to 0 to minimize hallucinations during data generation. While minor hallucinations may still occur, their impact on performance is expected to be minimal. This is because the synthetic data is used mostly to enrich the vocabulary of specific classes and improve model fine-tuning, rather than for direct inference.

The data augmentation phase is implemented in the **data_augmentation** notebook files. Separate notebooks are provided for the BERT-like method, the LLM-based approach, and the RAG method (which includes versions for both paragraphs and sentences, each available for the 7-class and 15-class settings).

2.1.3. Learning

Given the limited amount of data available to train our classifiers, we choose to fine-tune a BERT model, which has demonstrated strong performance on text classification tasks [31,32]. BERT is a pre-trained language model that learns contextual relationships between words in a text by analyzing sequences of input tokens (words or their parts) and generating fixed-size representations for each token. These representations capture the token's meaning within the context of the entire sequence. They are produced using a transformer-based architecture that employs self-attention mechanisms [11]. Once these contextual embeddings are obtained, a classification layer is used to predict a class label for the input sequence. BERT can be fine-tuned on labeled data to assign texts (such as interview responses) to predefined categories based on their semantic content. Its prediction quality can be further enhanced when fine-tuned on the augmented data, such as that generated using RAG [33], which provides additional relevant context. In our workflow, before fine-tuning BERT on the labeled data, external knowledge (e.g., similar interview responses or documents) is retrieved using RAG and the existing labeled examples. These retrieved texts are appended to the original input, enriching it with supplementary information and reducing potential ambiguity. This allows BERT to produce more accurate and contextually relevant representations for the input tokens, leading to improved classification performance, especially in cases where training data is limited.

As for the choice of BERT parameters, we follow the recommendations provided in [11]: a learning rate of $2 \cdot 10^{-5}$ and $\epsilon = 1 \cdot 10^{-8}$, the epsilon constant of Adam optimizer's update rule [34]. The learning rate was fine-tuned empirically among $5 \cdot 10^{-5}$, $4 \cdot 10^{-5}$, $3 \cdot 10^{-5}$ and $2 \cdot 10^{-5}$ through hyperparameter optimization. For epsilon, the default value from [34] was adopted, as no additional tuning was necessary. We also set the number of epochs to 10 and the batch size to 16. The batch size of 16 has been selected empirically. It affects the stability of the model and its training speed. The common values are 16 and 32; however, 16 was preferred due to its lower memory requirements, which is important given the constraints of the Google Colab environment we are using. The number of epochs was fixed at 10, selected empirically from between 5 and 10. This choice gives better results and decreases

[3] An example of a prompt composed of an interview question and **market opportunities** as the desired label: *"For which agricultural applications is PHA not suitable and why? Discuss market opportunities"*.

[4] An example of a response to the aforementioned prompt: *"PHA is not suitable for agricultural applications where longer-lasting synthetic plastics would be necessary as PHA breaks down relatively quickly in natural environments. Despite its limitations, there are market opportunities for PHA in agricultural applications where biodegradability and minimal environmental impact are the key criteria"*.

[5] https://noaw2020.eu

Algorithm 1 Different steps in the *prediction()* function for predicting the class index of a text segment.

Require: segment_text ← string
Ensure: Predicted class label
test_ids = ∅
test_attention_mask = ∅

`# Apply the tokenizer`
encoding ← Tokenize *segment_text*

`# Extract IDs and Attention Mask`
test_ids += *encoding['input_ids']*
test_attention_mask += *encoding['attention_mask']*
Concatenate *test_ids* along dimension 0
Concatenate *test_attention_mask* along dimension 0

`# Forward pass, calculate logit predictions`
Disable gradient computation
output ← Forward pass through model with:

- *input_ids* = *test_ids*
- *token_type_ids* = *None*
- *attention_mask* = *test_attention_mask*

logits ← Extract logits from *output*
prediction ← Apply *argmax()* to *logits* and flatten the result
return *prediction*

the chance of underfitting. We repeat each experiment 5 times to address the model instability problem [35] and report the best and average results, which we compute using the mean function.

The main goal in this work is to correctly classify relevant content (true positives), even if this results in some false positives. Therefore, we have prioritized the use of precision and recall, which are more standard in text classification tasks, especially in the context of information retrieval. More precisely, to perform the evaluation, we use precision, recall, F_1 score and weighted version of accuracy, as defined in [36]. The precision, recall, F_1 score is used to assess the prediction quality of each class, while the weighted accuracy is employed to select the best-performing classifier across multiple experiments. To ensure conciseness, we report only the F1 scores and weighted accuracy.[2]

The model fine-tuning is implemented in the **segment_classification** notebooks, which are available for both the 7-class and 15-class settings.

2.1.4. Prediction

In our pipeline, new uncoded data from upcoming rounds of interviews are labeled by a fine-tuned classifier. In practice, we use four separate classifiers: one for paragraphs and one for sentences, each fine-tuned for both 7-class and 15-class settings. In addition, we have separate classifiers for original and augmented data. By default, the tool employs classifiers that are fine-tuned on augmented data. Alternatively, this can be adjusted if such an option is available.

Prediction of the class of an unlabeled text, for both fine-tuning the classifier and annotating input data in the tool's interface, is implemented in the **prediction()** function (Algorithm 1).

2.1.5. Implementation

AgriCode is fully written in Python 3. In the tool's workflow, we used the *NLTK* library [37] to manipulate textual data. We used *LangChain* [38] as an API for LLM implementation and RAG, and *ChatGPT 3.5-turbo* as the specific instance of the LLM. We used the *BertForSequenceClassification* model from the HuggingFace library [10] as the implementation of BERT. Finally, we used the *scikit-learn* library [39] to implement quality measures used for evaluation.

The tool's frontend is implemented using *Streamlit* [40] to provide an interface to the backend:

- **Home.py** – homepage of the web application, implementing the selection between the Paragraph classification and Sentence classification methods
- **pages/Paragraphs.py** – implementation of Paragraph classification
- **pages/Sentences.py** – implementation of Sentence classification

In the next section, we present its functionalities.

2.2. Software functionalities

AgriCode is designed to accurately detect and code information related to the valorization of agricultural residues. The tool features a user-friendly interface, demonstrated in the video showing how users interact with the tool, provided with the electronic version of the article (also available by the link: https://youtu.be/Vid89wLL7Cg).

The interface allows users to construct input segments by selecting the questionnaire, block name and question from a predefined list, with an additional option for plain text input. Depending on the setting, the tool classifies these segments at the paragraph or sentence level. In the following sections, we detail its functionalities.

2.2.1. Segment construction

In our tool, it is possible to predict a class for each paragraph or sentence within a given textual segment. This segment can be either structured or unstructured. In the first option, the segment must include titles (questionnaire name, block name, and question) and paragraphs with respondent answers, with each title and paragraph separated by empty lines. This approach generally enhances prediction accuracy [26], especially in the sentence setting, where the textual examples have very limited vocabulary.[2] In the second option, the text is analyzed as a whole, and titles, if present, are treated as independent paragraphs.

For text input, there are two modes: "Segment constructor" and "Plain text". To simplify text input, the questionnaire name, block name, and question can be selected from predefined lists in the "Segment constructor" mode (Fig. 4). Once selected, these names and the question can be modified in the "Plain text" mode. The predefined lists are used in the context of the Agriloop project, but they can be modified in the code if necessary.

2.2.2. Paragraph classification

AgriCode performs text annotation using two principal methods: by paragraphs and by sentences. In the paragraph classification method, AgriCode highlights each paragraph with a color code corresponding to 7 or 15 classes, depending on the setting, with an automatically generated legend provided. The legend is defined in the code and can be modified if necessary. The legend and highlighted paragraphs appear after the 'Predict' button is clicked. By default, the tool utilizes an improved classifier based on an augmented corpus (see Section 2). This option can be deactivated via a designated checkbox (Fig. 4). This feature is particularly useful for comparing prediction results between the improved classifier and the standard classifier.

2.2.3. Sentence classification

In the sentence classification method, AgriCode highlights each sentence with a color code corresponding to 7 or 15 classes, depending on the setting. Similarly to the paragraph classification method, the legend and highlighted sentences appear after the 'Predict' button is clicked (Fig. 4).

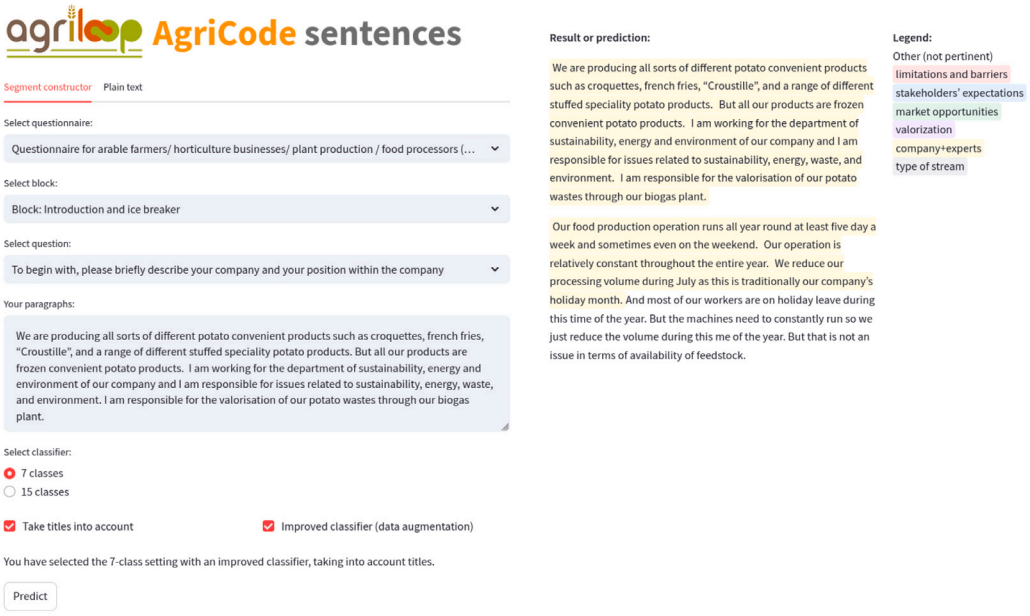


Fig. 4. The AgriCode interface in the "Segment constructor" mode for the sentence classification method, and the result of sentence annotation in the 7-class setting.

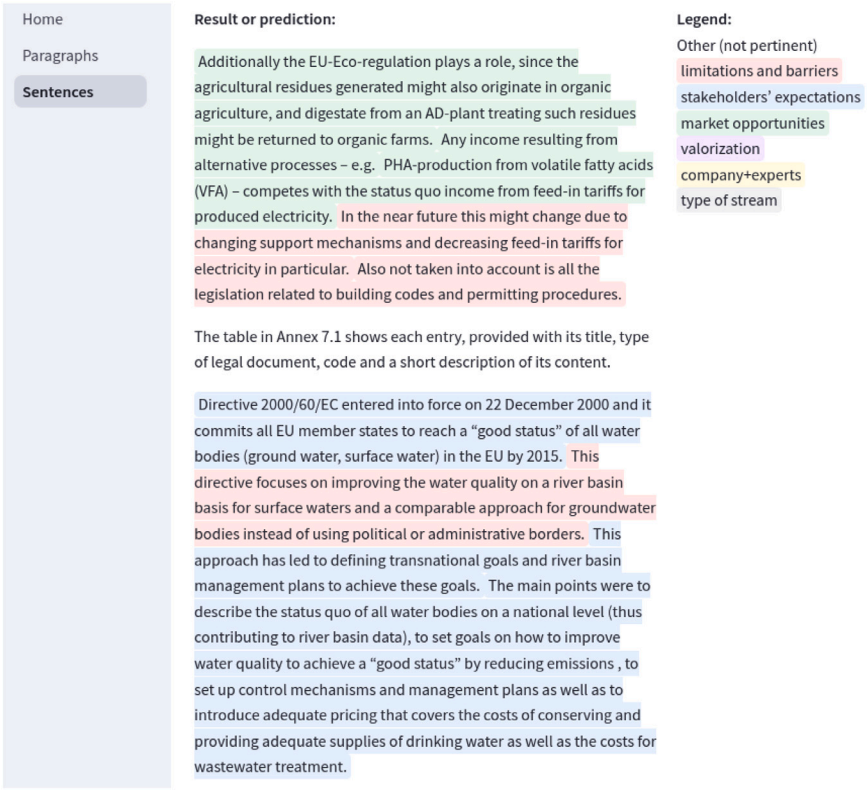


Fig. 5. Annotation of an abstract from the NoAW project report in sentence mode for unstructured text and the 7-class setting.

3. Illustrative examples

AgriCode has the capability to accurately detect and code information related to the valorization of agricultural residues in both structured and unstructured textual data. To illustrate its capabilities with structured data, consider the example in Fig. 4. In the provided paragraphs, the tool successfully identified information related to company descriptions and experts working within the companies by highlighting corresponding sentences with the **company+experts** class. This functionality is particularly useful for analyzing interview responses.

For unstructured data, a broader purpose of our tool is to provide a comprehensive understanding of agricultural contexts within textual information. To evaluate this capability, consider an extract from a

Table 1
Summary of the evaluation results using different settings.

Method	Data	Accuracy (weighted)	
		7 classes	15 classes
Paragraphs	original	0.87	0.79
	augmented	0.88	0.82
Sentences	original	0.79	0.75
	augmented	0.82	0.78

report from the NoAW project (Fig. 5). In this example, the tool identified:

- **market opportunities**: various market opportunities in the agricultural and bioeconomy sectors, particularly in sustainable practices, organic farming, and alternative energy production (only extract).
- **limitations and barriers**: challenges such as decreasing feed-in tariffs, complex regulatory requirements, including building codes and permitting procedures (first extract), as well as potential regulatory and geographic obstacles (another extract).
- **stakeholders' expectations**: mentions of goals and commitments outlined by directives (first extract) and the expectations and responsibilities placed on stakeholders to achieve objectives related to water management (another extract).

These examples demonstrate AgriCode's effectiveness in processing both structured and unstructured data to derive valuable insights.

4. Impact

Automatic coding of interviews in qualitative research is a highly desired functionality among researchers in this field. We developed the AgriCode tool to address this need in our project. To demonstrate its effectiveness, the next paragraph presents the results of the tool's experimental evaluation.

The results demonstrate strong performance achieving up to 87% accuracy for paragraphs and up to 79% for sentences (Table 1). Fine-tuning the classifier on augmented data using RAG improved overall performance by up to 3% (for detailed results per class please refer to Appendix C). We also evaluated the model on the second round data, which shows promising potential for the tool (for the results please refer to Appendix D).

AgriCode can be particularly useful for annotation support and error verification, not only in the next rounds of Delphi for the Agriloop project but also in other projects with a similar context. Furthermore, the tool's workflow can be adapted for any other domain where automated coding of textual data is needed.

5. Conclusions

We presented the AgriCode tool for coding interviews used to collect expert knowledge on valorization of agricultural residues. The tool's workflow includes data augmentation using RAG to address data scarcity, followed by the application of an offline classifier. This approach improved prediction quality for specific classes where enhancement was needed and resulted in a slight overall performance improvement.

Compared to an LLM, AgriCode's workflow is less demanding in terms of infrastructure as it only requires a single GPU or access to a Google Colab for fine-tuning BERT classifiers. Additionally, AgriCode does not require permanent API usage fees, only during the data augmentation phase. This makes AgriCode a more accessible option in a limited resource setting. The results provided by AgriCode's fine-tuned classifiers are also reproducible, which may be important for certain applications. However, the tool's scalability remains a limitation. To add more classes or use another code system, the whole process needs to be repeated from data augmentation to classifier fine-tuning, and updating the legend in the interface. This can make it less efficient for larger-scale projects that require frequent updates to the code system.

Currently, the tool supports predictions up to two levels of the category system, and performance for fine-grained settings could still be improved. We plan to continue this work by incorporating the data from the upcoming rounds of interviews within the Agriloop project. After completing all rounds and collecting more data, it will be possible to increase the number of classes to include up to three levels of the code system hierarchy. Finally, we aim to explore the development of a system, based on an LLM and few-shot learning, capable of coding interviews and other texts in this context using any given hierarchy of classes with only a few examples per class.

CRediT authorship contribution statement

Maksim Koptelov: Writing – original draft, Software, Methodology, Data curation, Conceptualization. **Jan Linck:** Writing – review & editing, Validation, Data curation, Conceptualization. **Pierre Bisquert:** Writing – review & editing, Methodology, Funding acquisition. **Patrice Buche:** Writing – review & editing, Methodology, Funding acquisition, Data curation. **Mathieu Roche:** Writing – review & editing, Methodology, Conceptualization.

Ethics statement

The interview response data used for fine-tuning the classifiers and for data augmentation originally contained brand names and personal information of the respondents. These data were pseudonymized with the respondents' permission. Any resemblance to real brands or individuals, if present, is purely coincidental.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors acknowledge the support of the European Union's Horizon Europe research and innovation program AgriLoop (project ID 101081776).

Appendix A. Code definitions

This appendix contains the list of codes for the 7-class and 15-class settings implemented in AgriCode.

A.1. The 7-class setting codes

C0 **other (not pertinent)**
C1 **market opportunities**
C2 **limitations and barriers**
C3 **stakeholders' expectations**
C4 **valorization**
C5 **company+experts**
C6 **type of stream**

Table B.2
Corpus size of the first (R1) and second (R2) rounds for the 7-class setting (C0-C6) for paragraphs and sentences, for the original (orig) and augmented (augm) data.

Entries	Data	C0	C1	C2	C3	C4	C5	C6	Total
Paragraphs	R1 (orig)	126	102	98	96	54	30	25	531
	R1 (augm)	157	102	115	96	54	30	25	579
	R2 (cleaned)	39	55	17	23	1	17	11	163
Sentences	R1 (orig)	784	296	247	314	124	112	58	1935
	R1 (augm)	784	367	247	314	160	112	62	2046
	R2 (cleaned)	146	192	49	55	8	86	61	597

Table B.3
Corpus size of the first (R1) and second (R2) rounds for the 15-class setting (C0-C14) for paragraphs (P) and sentences (S), for the original (orig) and augmented (augm) data.

Entries	Data	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	Total
P	R1 (orig)	126	65	59	45	35	32	30	26	25	22	17	16	11	10	9	528
	R1 (augm)	126	65	59	45	35	32	30	26	25	22	42	16	25	10	9	567
	R2 (cleaned)	39	1	4	4	4	1	17	30	11	8	13	15	8	0	0	155
S	R1 (orig)	784	117	102	151	82	81	112	120	58	74	74	81	56	29	14	1935
	R1 (augm)	784	117	102	237	123	81	112	169	58	74	74	81	71	29	32	2144
	R2 (cleaned)	146	1	5	7	6	8	86	99	61	43	55	42	5	0	0	564

Table C.4
Evaluation results for the 7-class setting (C0-C6) for paragraphs (P) and sentences (S) on the original (orig) and augmented (augm) corpus. Acc corresponds to weighted accuracy, and F_1 to the run with the highest Acc.

Method	Data	C0	C1	C2	C3	C4	C5	C6	Acc	Acc
		F_1	F_1	F_1	F_1	F_1	F_1	F_1	max	mean
P	orig	0.68	0.88	0.85	0.90	0.95	0.92	1.00	0.87	0.83
	augm	0.74	0.86	0.86	0.90	0.95	0.92	1.00	0.88	0.85
S	orig	0.78	0.84	0.88	0.86	0.42	0.93	0.78	0.79	0.78
	augm	0.78	0.88	0.85	0.80	0.70	0.95	0.72	0.82	0.78

A.2. The 15-class setting codes

C0 **other (not pertinent)**
C1 **stakeholders' expectations > valorization/PHA-applications**
C2 **limitations and barriers > valorization /PHA-applications**
C3 **market opportunities > PHA MO**
C4 **market opportunities > PHA-Applications MO**
C5 **valorization > current structures**
C6 **company+experts**
C7 **limitations and barriers > Main issues and challenges for extracted/microbial protein**
C8 **type of stream**
C9 **stakeholders' expectations > PHA expectation**
C10 **limitations and barriers > Main issues and challenges for PHA**
C11 **market opportunities > MP MO**
C12 **stakeholders' expectations > MP**
C13 **valorization > satisfaction**
C14 **valorization > advantages**

Appendix B. Data set properties

This appendix presents data set properties used in the AgriCode's workflow (Tables B.2, B.3). It includes corpus sizes from the first (R1) and second (R2) rounds of data for both paragraphs and sentences, for the 7-class and 15-class settings. The tables show the differences between the original and augmented R1 corpora, which were used for both training and validation, as well as their comparison with the cleaned R2 data, used for validation only (see Appendix D for more details).

Appendix C. Experimental evaluation

This appendix presents evaluation results of AgriCode.

To demonstrate the efficacy of the tool, we evaluated it using test data for both the paragraph and sentence methods. For each method, we evaluated both the 7-class and 15-class settings. Finally, we assessed both the original and augmented data using the same (non-augmented) test split. The data set properties are presented in Tables B.2, B.3 in Appendix B, while the results are provided in Tables C.4, C.5 in this appendix. For the augmented classifiers, only the best results are presented.[2]

As seen from the results, the original classifier for paragraphs achieves very good performance across almost all classes in the 7-class setting (88%–100% F_1 score). The least performing classes are C0 and C2. While we experimented with other combinations,[2] augmenting these two classes resulted in the expected performance improvement (raising C0 to an acceptable level of 74% and C2 to 86%, with the average performance improving from 83% to 85%). This was achieved by augmenting the training sets of the C0 and C2 classes by 30% and 20%, respectively (Table B.2). On the other hand, in the 7-class sentence setting, the most problematic class is C4, which exhibits low performance. Through data augmentation, we improved the performance of this class to an acceptable level (70% F_1), while maintaining the same average weighted accuracy (78%). This improvement was achieved by augmenting the C4 training set by 30%. Second class in this setting which performance could be improved is C0 (78%). However, augmenting it did not lead to the desired results and was therefore replaced by other classes (C1, C4 and C6). This helped reduce class imbalance and led to an overall performance improvement from 79% to 82%. Interestingly, the performance of the sentence classifier for the C0 class is higher than that of the paragraph classifier. We believe this is because shorter fragments of the "not pertinent" class contain less key vocabulary found in specific classes, which reduces the likelihood of misleading the classifier. The same behavior is observed in the 15-class setting, further supporting our hypothesis.

The 15-class setting is more challenging. We achieved acceptable performance for most classes (F_1 scores ranging from 67% to 96% for both paragraphs and sentences), except for C0 and C10. While we managed to improve the performance of class C10 in both settings, we were unable to significantly enhance the performance of class C0 without compromising overall performance. Moreover, class C0 consistently

Table C.5
Evaluation results for the 15-class setting (C0-C14) for paragraphs (P) and sentences (S) on the original (orig) and augmented (augm) corpus. Acc corresponds to weighted accuracy, and F_1 to the run with the highest Acc.

Method	Data	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	Acc	Acc
		F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	F_1	max	mean
P	orig	0.56	0.84	0.96	0.84	0.93	0.86	0.83	0.77	1.00	0.73	0.50	0.86	0.80	1.00	0.67	0.79	0.73
	augm	0.58	0.84	0.96	0.94	0.93	0.93	0.92	0.83	1.00	0.73	0.80	0.86	0.67	1.00	0.67	0.82	0.79
S	orig	0.63	0.71	0.67	0.77	0.76	0.90	0.93	0.84	0.92	0.73	0.23	0.96	0.82	0.73	0.80	0.75	0.74
	augm	0.63	0.60	0.73	0.68	0.86	0.95	0.97	0.84	0.96	0.71	0.48	0.92	0.85	0.92	1.00	0.78	0.76

Table D.6
Summary of the validation results on the R2 data.

Method	Data	Accuracy (weighted)	
		7 classes	15 classes
Paragraphs	Original	0.80	0.64
	Augmented	0.74	0.78
Sentences	Original	0.61	0.51
	Augmented	0.67	0.63

shows low performance in both paragraph and sentence settings, and class C10 performs very poorly in the sentence setting. This can be explained by the fact that we did not augment these classes, simply because effective augmentation was not feasible. There are two main reasons for this. First, we found that a BERT-type classifier struggles to effectively distinguish non-pertinent instances (class C0) in multi-class setting, as these instances lack class-specific vocabulary and contain a significant amount of general vocabulary common to other classes. As the number of classes increases, the vocabulary differences between them become more blurred. As a result, augmenting these instances do not provide the expected improvements. Second, augmenting C10 actually led to a complete drop in performance for this class.[2] This may be because C10 is very similar to C2 as both address limitations and barriers of PHA, but with slightly different focuses (C10 targets producers of PHA, while C2 focuses on PHA end-products). The BERT classifier can be misled due to the minimal vocabulary difference between these two classes. Therefore, we focused on augmenting other classes where performance could still be improved (C3, C4, C7, C12 and C14). This helped reduce class imbalance and led to performance gains for most of these classes, contributing to a slight overall improvement.

Appendix D. Validation on R2 data

The tool was also evaluated on data from the second round of interviews (Table D.6). To perform this evaluation, we cleaned the data by retaining only responses to questions similar to those in R1. We applied the Levenshtein distance [41], setting it to 20 for paragraphs and 15 for sentences, with the exact values determined experimentally.

The peak performance in this setting reached 80% for paragraphs and 61% for sentences. The drop in performance for R2 data, particularly for sentence-level classification, is likely due to changes in both the questions and the code system in the second round. This had a significant impact on sentence-level classification since the questions contribute a substantial portion of the vocabulary used for classification. The augmented classifier improved performance by 6%–14% except in the paragraph-level 7-class setting. We believe this is due to the evolved code system and difficulties in mapping certain classes. This issue could be avoided if the code system remains consistent across all rounds of interviews.

Finally, the drop in performance could be explained by a potential bias in the annotated training data. To minimize this risk, it is important to ensure both high data quality and a sufficient number of labeled examples. One way to improve annotation quality is by developing clear annotation guidelines and involving multiple annotators to label the same data. This helps to make the labels more consistent and less subjective. To deal with the limited amount of data, we already employ a RAG-based approach, which enhances the dataset by retrieving relevant external content. This has improved the accuracy of predictions, especially for challenging categories such as "other (not pertinent)" and some underrepresented specific classes. We believe that continuing to improve this method could lead to even better results. An alternative solution we are considering is a sequential training, where models trained on R1 and R2 data are used to predict labels for R3, and so on through subsequent rounds. Since this approach is highly data-dependent, increasing the amount of annotated data across rounds would be beneficial for both performance and robustness.

Data availability

The manually annotated data, as well as the synthetically generated data used for fine-tuning the models, are publicly available at the following link: https://doi.org/10.57745/91BXIY.

References

[1] Knott E, Rao AH, Summers K, Teeger C. Interviews in the social sciences. Nat Rev Methods Prim 2022;2(1):p. 73.

[2] Marjaei S, Yazdi FA, Chandrashekara M. MAXQDA and its application to LIS research. Libr Philos Pr 2019;1–9.

[3] Oun MA, Bach C. Qualitative research method summary. Qual Res 2014;1(5):252–8.

[4] Thorne S. Data analysis in qualitative research. Evidence- Based Nurs 2000;3(3):68–70.

[5] Perrin AJ. The CodeRead system: Using natural language processing to automate coding of qualitative data. Soc Sci Comput Rev 2001;19(2):213–20.

[6] Crowston K, Liu X, Allen EE. Machine learning and rule-based automated coding of qualitative data. Am Soc Inf Sci Technol 2010;47(1):1–2.

[7] Chen NC, Drouhard M, Kocielnik R, Suh J, Aragon CR. Using machine learning to support qualitative coding in social science: Shifting the focus to ambiguity. Trans Interact Intell Syst 2018;8(2):1–20.

[8] Rietz T, Maedche A. Cody: An AI-based system to semi-automate coding for qualitative research. In: Proceedings of the 2021 conference on human factors in computing systems. 2021, p. 1–14.

[9] Baumgartner P, Smith A, Olmsted M, Ohse D. A framework for using machine learning to support qualitative data coding. Tech. rep., Center for Open Science; 2021, http://dx.doi.org/10.31219/osf.io/fueyj.

[10] Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, et al. Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. ACL; 2020, p. 38–45.

[11] Devlin J, Chang M, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 annual conference of the North American chapter of the ACL: human language technologies. 2019, p. 4171–86.

[12] Kuckartz U, Rädiker S. Analyzing qualitative data with MAXQDA. Springer; 2019.

[13] MAXQDA 24 manual. The smart coding tool. 2025, https://www.maxqda.com/help-mx24/work-with-coded-segments/smart-coding-tool. [Accessed 30 January 2025].

[14] Chew R, Bollenbacher J, Wenger M, Speer J, Kim A. LLM-assisted content analysis: Using large language models to support deductive coding. 2023, Preprint arXiv:2306.14924.

[15] Dunivin ZO. Scalable qualitative coding with LLMs: Chain-of-thought reasoning matches human performance in some hermeneutic tasks. 2024, Preprint arXiv:2401.15170.

[16] Zhang H, Wu C, Xie J, Rubino F, Graver S, Kim C, et al. When qualitative research meets large language model: Exploring the potential of QualiGPT as a tool for qualitative coding. 2024, Preprint arXiv:2407.14925.

[17] Escórcio R, Bento A, Tome AS, Correia VG, Rodrigues R, Moreira CJ, et al. Finding a needle in a haystack: Producing antimicrobial cutin-derived Oligomers from Tomato pomace. ACS Sustain Chem Eng 2022;10(34):11415–27.
[18] Linstone HA, Turoff M. The Delphi method. MA: Addison-Wesley Reading; 1975.
[19] Sablatzky T. The Delphi method. Hypothesis: Res J Heal Inf Prof 2022;34(1).
[20] Laifa A, Gautier L, Cruz C. Impact of textual data augmentation on linguistic pattern extraction to improve the idiomaticity of extractive summaries. In: Proceedings of the 23rd international conference on big data analytics and knowledge discovery. Springer; 2021, p. 143–51.
[21] Afzal S, Maqsood M, Nazir F, Khan U, Aadil F, Awan KM, et al. A data augmentation-based framework to handle class imbalance problem for Alzheimer's stage detection. IEEE Access 2019;7:115528–39.
[22] Damodaran P. Parrot: Paraphrase generation for NLU. 2021, Computer software, version 1.0, https://github.com/PrithivirajDamodaran/Parrot_Paraphraser.
[23] Tang R, Han X, Jiang X, Hu X. Does synthetic data generation of LLMs help clinical text mining? 2023, Preprint arXiv:2303.04360.
[24] Gao Y, Xiong Y, Gao X, Jia K, Pan J, Bi Y, et al. Retrieval-augmented generation for large language models: A survey. 2023, Preprint arXiv:2312.10997.
[25] Xiong H, Wu J, Liu L. Classification with classoverlapping: A systematic study. In: Proceedings of the 1st international conference on e-business intelligence. Atlantis Press; 2010, p. 303–9.
[26] Koptelov M, Holveck M, Cremilleux B, Reynaud J, Roche M, Teisseire M. Towards a (semi-) automatic urban planning rule identification in the french language. In: Proceedings of the 10th international conference on data science and advanced analytics. IEEE; 2023, p. 1–10.
[27] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux MA, Lacroix T, et al. Llama: Open and efficient foundation language models. 2023, Preprint arXiv:2302.13971.
[28] Jiang AQ, Sablayrolles A, Mensch A, Bamford C, S. Chaplot D, Casas D, et al. Llama: Open and efficient foundation language models. 2023, Preprint arXiv:2310.06825.
[29] Bai J, Bai S, Chu Y, Cui Z, Dang K, Deng X, et al. Qwen technical report. 2023, Preprint arXiv:2309.16609.
[30] Sandireddy R. Agro dataset QA final. 2024, dataset, https://huggingface.co/datasets/Rahulrayudu/Agro_Dataset_QA_Final.
[31] Sun C, Qiu X, Xu Y, Huang X. How to fine-tune bert for text classification? In: Proceedings of the 18th China national conference on Chinese computational linguistics. Springer; 2019, p. 194–206.
[32] González-Carvajal S, Garrido-Merchán EC. Comparing BERT against traditional machine learning text classification. 2020, Preprint arXiv:2005.13012.
[33] Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Adv Neural Inf Process Syst 2020;33:9459–74.
[34] Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014, Preprint arXiv:1412.6980.
[35] Zhang T, Wu F, Katiyar A, Weinberger KQ, Artzi Y. Revisiting few-sample BERT fine-tuning. In: Proceedings of the 9th international conference on learning representations. 2020.
[36] Koptelov M, Holveck M, Cremilleux B, Reynaud J, Roche M, Teisseire M. A manually annotated corpus in french for the study of urbanization and the natural risk prevention. Sci Data 2023;10(1):818.
[37] Bird S, Klein E, Loper E. Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc; 2009.
[38] Chase H. LLM app development framework. 2025, https://langchain.com. [Accessed 30 January 2025].
[39] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[40] Majed A, Martin A, Andrej B, Teichtmann A, Sehmi A, Neo B, et al. Streamlit: A faster way to build and share data apps. 2025, https://streamlit.io. [Accessed 30 January 2025].
[41] Levenshtein V. Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady, vol. 10, (no. 8):Soviet Union; 1966, p. 707–10.