Using Chat-GPT for coding properties in semantic memory studies

Diego Ramos Á.
Sergio Chaigneau
diego.ramos.a@edu.uai.cl
sergio.chaigneau@uai.cl
Universidad Adolfo Ibáñez
Santiago, Metropolitana, Chile

Sebastián Moreno
Enrique Canessa
sebastian.moreno@uai.cl
ecanessa@uai.cl
Universidad Adolfo Ibáñez
Viña del Mar, Valparaíso, Chile

Abstract

In this paper, we propose Chat-GPT for the codification of a Property Listing Task (PLT). PLTs are a standard method to study semantic memory (understanding how people represent concepts coded in their minds). In a PLT, a group of participants is asked to list properties/features for a concept (e.g., “horse”). Given that different properties could have the same meaning (e.g., “quadruped” and “four legs”), the mentioned properties must be codified before any analysis. Currently, the codification process is carried out by at least two human coders, making it a slow and non-replicable process (given the variability of codes assigned by the coders). Automating this codification process through Chat-GPT will speed up the codification, reduce the variability of the human codification process, and allow replicable results. We compare Chat-GPT with AC-PLT (the first semi-automatic codification framework for PLTs), using accuracy on two datasets. The experiment compares AC-PLT with GPT-3.5-turbo-0125 (using one-shot prompting and fine-tuning) and GPT-4o (using one-shot prompting). GPT-3.5-turbo-0125 with fine-tuning shows comparable performance with AC-PLT, opening a possible area of research for this codification process.

Keywords

Semantic Memory, Large Language models, LLMs, codification, Property Listing Task, PLT

ACM Reference Format:
Diego Ramos Á., Sergio Chaigneau, Sebastián Moreno, and Enrique Canessa. 2025. Using Chat-GPT for coding properties in semantic memory studies. In *Proceedings of the Workshop "SciSoc LLM Workshop Large Language Models for Scientific and Societal Advances" @ 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25), August 3, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 Introduction

Large language models (LLMs) are increasingly used for multiple purposes, including automating certain tasks in cognitive psychology [11]. One of the most studied fields in this area is semantic memory, the general human knowledge of concepts and verbal symbols [15]. This field focuses on understanding how people comprehend concepts through language. Despite their importance, LLMs have not been fully used in this area's research.

To conduct semantic memory studies, it is necessary to collect data from individuals. Specifically, it is necessary to collect words corresponding to the properties mentioned by a group of participants in response to a particular stimulus, which varies according to the study's objective. Among the most common methods to generate this type of data are the Semantic Fluency Task (SFT) [7], the Associative Word List Task (AWLT) [16], and the Feature Listing Task (FLT) [2], also called the Property Listing Task (PLT), being PLT the focus of this work. In a PLT, participants are asked to list the properties or characteristics associated with a particular concept. For example, given the concept “horse”, participants can mention “quadruped”, “four legs”, “loyalty”, and other properties. Once these properties are collected, they are summarized in a frequency matrix, which is called a ‘Conceptual Property Norm’, where the rows represent concepts, the columns represent properties, and a cell corresponds to the number of people who mention that specific codified property for a particular concept [6, 9, 10, 13, 17, 19]. To enhance understanding of this work and avoid term confusion, we show in Table 1 the definitions of the terms Property and Code, so that the reader can easily refer to them.

To transform a PLT into a CPN, properties must be assigned a code through a multi-step process called coding. Given that people might refer to the same property using different wordings, a unique representative code must be assigned to those wordings. In the previous example, where the concept is “horse”, “quadruped” and “four legs” refer to the same property and are assigned the same code. Coding is an extensive process where a human coder reviews, analyzes, and assigns codes to different listed properties following a set of rules and instructions. First, the rules for code creation are defined. Second, a human coder has to assign each property a corresponding code. Third, this process is repeated at least once with a new coder because coders interpret the data differently. If an assigned code differs among coders, the coders discuss the situation and agree on a single code, or a new coder is used to solve the disagreement. Unfortunately, on average, it can take a coder months to code even small datasets.

To date, we do not know an automatic codification process for PLT data in the literature. Even though Large Language Models have been extensively used as annotators for different tasks [1, 18, 12], most of these tasks are focused on sentences and a few classes (usually binary). In contrast, in a PLT, we have a small quantity of information (usually a single word instead of a sentence)

Term	Definition	Example
Property	The original sentence or word listed by a participant in a PLT for a given concept	For the concept “horse” a participant might list the sentence “has four legs”
Code	The standardized label (code) assigned to a word/sentence by a coder or a coding system	For the sentence “has four legs”, the assigned code might be “quadruped”

Table 1: Definition of some terms used in this work

and multiple classes (the smallest dataset we use here has 1,296 classes and 4,941 properties). In the psychology area, only a few methods have been developed to assist the codification process. For example, the Buchanan method “cleans” the properties and counts the number of words and sequences of words that appear in the whole dataset [5]. The RK-processor applies text cleaning to the properties to help human coders reduce the time required for codification [22]. Finally, the Assisted Coding for Property Listing Task (AC-PLT) is considered the first semi-automatic framework that can suggest codes, where it learns from previous datasets to suggest codes for new properties [21], but it needs to be trained on coded datasets.

This work applies GPT-3.5-turbo and GPT-4o in a PLT codification process. We apply GPT-3.5-turbo and GPT-4o with the one-shot prompting technique and GPT-3.5-turbo fine-tuned. We compare these results with the AC-PLT framework. Our results show that one-shot prompting cannot replicate the codification process of human coders, but important progress can be obtained with the fine-tuning approach. Optimizing this process through LLMs would significantly impact the quality of semantic memory studies, speeding up the codification process and allowing the replication of the codification process, avoiding the subjectivity of human coders.

2 Literature Review

The property codification process has been largely studied in psychology [6]. While several studies focused on the issues and methods to improve this process [14, 3], few works are related to the automation of it. One of the most advanced tools for coding PLT data is the Assisted Coding for Property Listing Task (AC-PLT).

According to the authors, the AC-PLT framework is an automatic codification process divided into three steps: text cleaning, word embedding, and classification/recommendation. First, using the Spacy library [20], a text-cleaning process helps to reduce the variability of the properties. It starts by tokenizing (isolating each word and special character into a single string), then sets every letter to lowercase, and finally discards all special characters and stop words (words that do not contribute to the final meaning, such as “the”). Second, AC-PLT applies word embedding, a technique to transform each word into a vector of real numbers [23]. The proposed framework uses a Spanish version of the Word2Vec model with 300 dimensions [8]. In the case of properties composed of 2 or more words/tokens, it sums the vectors of each word and divides this sum by their Euclidean norm to create a representative vector of the property. Third, it uses the k-Means model for classification. For this purpose, the model clusters the property vectors from the training data into k groups, where each group has been labeled with a single code (the code's mode of each cluster). To classify a new property, AC-PLT applies the cleaning and embedding processes, calculates the distance to each cluster, and returns the assigned code of the closest cluster. AC-PLT can also return a list of m codes corresponding to the codes of the m closest clusters.

As mentioned by its authors, the AC-PLT framework has several weaknesses, which impact its performance. First, the embedding model Word2Vec is context-insensitive, i.e., Word2Vec creates a unique vector for each word independently of the context of the words. So, in polysemy (words with multiple meanings) or homonymy (words written equally but with different meanings), the generated vectors are the same for a given word, although the word may have different meanings. Additionally, this framework uses a clustering model for classification, limiting the number of possible classes to even lower than k (the number of clusters). In some cases, two or more clusters could have the same code.

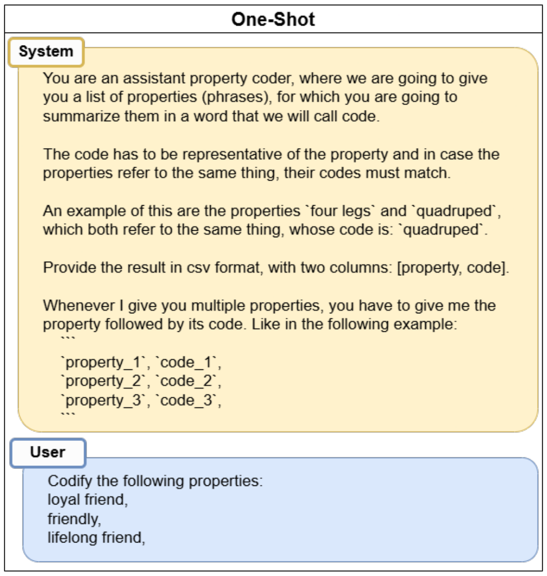
3 Proposal

This paper proposes to use Chat-GPT (GPT-3.5-turbo-0125 and GPT-4o versions) to replicate the codification process of a human coder, comparing the one-shot prompting technique and fine-tuning (only for GPT-3.5-turbo-0125). The first technique, one-shot, receives instructions on codifying the properties, giving one example in the prompt. In the second technique, fine-tuning, we train the LLM model with examples of inputs and their expected output [4], updating the initial weights. Note, a prompt is the input text for the LLM, which is limited by its input and output length (number of tokens/words). In the case of GPT-3.5-turbo-0125, it has an input length (context window) of 16,385 tokens and a max output of 4,096 tokens; alternatively, the model GPT-4o has an input length of 128,000 tokens and a max output of 16,384 tokens.

For the experimentation with GPT-3.5-turbo-0125, we have a shorter context window, and we cannot upload a CSV file to process all the data at once. Because of these limitations, we use a JSONL file format, where each line of the file represents a query for the model. Each query can have three objects: ‘system’ (the model setup for the initial task), ‘user’ (the main instruction of codification and the properties), and ‘assistant’ (the output we want from the model, only used for fine-tuning). In contrast, GPT-4o has a larger context window and supports files. In this case, we only give the prompt and upload the corresponding file with the properties to codify.

3.1 One-Shot prompting

In the one-shot prompting technique, we give the instruction and context of the codification process to the model, and, additionally, we provide an example of the codification of a property. For the model GPT-3.5-turbo-0125, we give the instruction shown in Figure 1. Given that the original data is in Spanish, this instruction corresponds to a direct translation to English of the original prompt written in Spanish. Something to note is that we specify to the model to "summarize them in a word that we will call code" instead of simply saying "codification", normally associated with writing computer codes. Once the main instructions of the model are defined, we proceed to ask for the codification of multiple properties, limited by the maximum output allowed by the model. In this case, we apply the instruction multiple times, using the prompt: "Codify the following properties: " following each property with a space for the code.


One-Shot

System

You are an assistant property coder, where we are going to give you a list of properties (phrases), for which you are going to summarize them in a word that we will call code.

The code has to be representative of the property and in case the properties refer to the same thing, their codes must match.

An example of this are the properties `four legs` and `quadruped`, which both refer to the same thing, whose code is: `quadruped`.

Provide the result in csv format, with two columns: [property, code].

Whenever I give you multiple properties, you have to give me the property followed by its code. Like in the following example:

```
`property_1`, `code_1`,
`property_2`, `code_2`,
`property_3`, `code_3`,
```

User

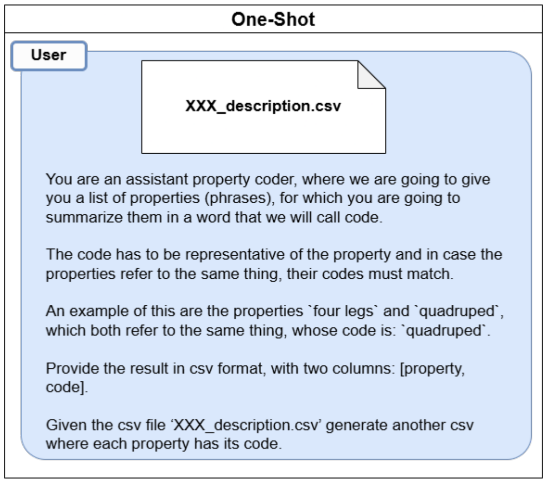
Codify the following properties:
loyal friend,
friendly,
lifelong friend,

Figure 1: Prompt used for one-shot on GPT-3.5-turbo-0125

The prompt used for the GPT-4o model is displayed in Figure 2. As can be seen, the system message was used as a user instruction, changing only the final paragraph. Also, in this model, we can include a file with one column and multiple rows, where each row is a property from the original dataset. Finally, the model provides a CSV file as output. In this model, we use the prompt once, given that the number of properties to codify is less than the maximum number of tokens.

3.2 Fine-tuning

In the fine-tuning technique, we trained the GPT-3.5-turbo-0125 model to specialize in our task, adjusting the weights of the original model [24]. The used prompt is displayed in Figure 3. As can be seen, it is similar to the original prompt on Figure 1. In the 'System', we give the initial instructions on how to codify. In the 'User', we define the properties that we want to codify. Finally, we give the training data in the 'Assistant', corresponding to some properties and the expected output. For a fair comparison, we train the model using a 5-fold cross-validation, as done in the original AC-PLT. Unfortunately, given the input and output limitations of GPT-3.5-turbo-0125, we had to separate the prompt into multiple instructions to be able to compare the models.


One-Shot

User

XXX_description.csv

You are an assistant property coder, where we are going to give you a list of properties (phrases), for which you are going to summarize them in a word that we will call code.

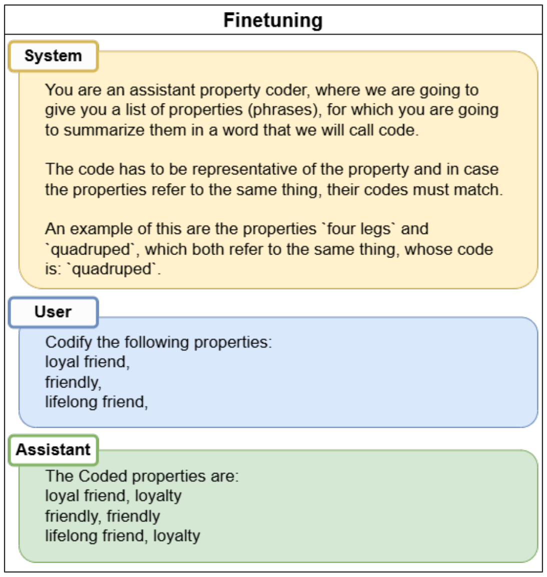
The code has to be representative of the property and in case the properties refer to the same thing, their codes must match.

An example of this are the properties `four legs` and `quadruped`, which both refer to the same thing, whose code is: `quadruped`.

Provide the result in csv format, with two columns: [property, code].

Given the csv file 'XXX_description.csv' generate another csv where each property has its code.

Figure 2: Prompt used for one-shot on GPT-4o


Finetuning

System

You are an assistant property coder, where we are going to give you a list of properties (phrases), for which you are going to summarize them in a word that we will call code.

The code has to be representative of the property and in case the properties refer to the same thing, their codes must match.

An example of this are the properties `four legs` and `quadruped`, which both refer to the same thing, whose code is: `quadruped`.

User

Codify the following properties:
loyal friend,
friendly,
lifelong friend,

Assistant

The Coded properties are:
loyal friend, loyalty
friendly, friendly
lifelong friend, loyalty

Figure 3: Prompt used for fine-tuning on GPT-3.5-turbo-0125

4 Experiments

We evaluated our approach using two datasets (codified by human coders) and compared the classification accuracy with AC-PLT using k-fold cross-validation. The first dataset, CPN27, has 4,941 properties and 1,296 different codes. The second dataset, CPN120, has 27,138 properties and 1,785 different codes. These datasets were previously used by AC-PLT, where the authors set the number of clusters (i.e., maximum possible codes) to 500.

To evaluate the performance of the LLM models compared to AC-PLT on the datasets codified by human coders (CPN27 and CPN120), we compared the accuracy of the LLM models with the top 1 and 5 accuracy of the AC-PLT framework. In the top 5 accuracy, the codification by AC-PLT is considered correct if the 'human assigned code' is among the first five suggested codes of AC-PLT.

We replicated the training of the original AC-PLT experiment using a 5-fold cross-validation (4 folds to train and one for testing) on the two datasets (CPN27 and CPN120).

Regarding hyperparameters, we searched for new hyperparameters in the AC-PLT, but we used default hyperparameters for the GPT models. For AC-PLT, we obtained the best performance with k=550 and k=1750 for CPN27 and CPN120, respectively. In the case of one-shot prompting (GPT-3.5-turbo-0125 and GPT-4o models), we use the models' default settings, without training or hyperparameter tuning. We only provided the models with a property codification example. Therefore, we divided the data into five training folds to compare the GPT models with the AC-PLT framework. In the GPT-3.5-turbo-0125 fine-tuning process, we keep the default hyperparameters (batch_size = 1 and learning_rate_multiplier= 2). However, we set n_epoch = 10 for CPN27 and n_epoch = 5 for CPN120, as a lower n_epoch reduces the number of tokens used for training, reducing the training monetary cost.

In Figure 4, we compare all models' accuracies for both datasets (top: CPN27 and bottom: CPN120). For the CPN27 dataset, we can see that the one-shot methods have a statistically lower accuracy than the AC-PLT method in both datasets (t-test with p-values $\leq$ 0.004, corrected for multiple comparisons using Bonferroni´s method), with GPT-4o being better than GPT-3.5-turbo-0125. Contrarily, the fine-tuning approach achieves better results than the AC-PLT top-5 accuracy (p-value $\leq$ 0.004). However, fine-tuning has a train accuracy of 0.913 and a test accuracy of 0.670, showing a possible overfitting of the method.

The CPN120 results are shown at the bottom plot on Figure 4. We can see that the one-shot prompting methods have lower accuracy than AC-PLT, having an accuracy of 0.073 for GPT-3.5-turbo-0125 and 0.178 for GPT-4o (p-values $\leq$ 0.004, corrected for multiple comparisons using Bonferroni´s method). Also, the fine-tuning method has a similar performance to AC-PLT Top 5. GPT-3.5-turbo-0125 achieves an accuracy of 0.554 and 0.582 for training and test, respectively. However, those accuracies are lower than AC-PLT top 5 accuracy (p-value = 0.01, corrected for multiple comparisons using Bonferroni´s method), and better than the AC-PLT top 1 (p-value $\leq$ 0.004).

In addition to the previous results, the fine-tuning of the model takes around 1 hour for training, while the testing process is even faster. On the contrary, the AC-PLT paper mentions that it took months for human coders to codify the smaller dataset (CPN27).

Unfortunately, the reported results have a monetary cost associated with the fine-tuning process for GPT-3.5-turbo-0125. The cost of training the model was US$25.26 and US$56.15 for CPN27 and CPN120, respectively. Correspondingly, we also needed to classify the test data, where we also spent US$0.48 and US$5.88 for CPN27 and CPN120, respectively. In total, the cost of fine-tuning GPT3.5-turbo-0125 was US$87.77, without hyperparameter search. A simple search for one of the hyperparameters, such as learning_rate_multiplier, with values from 1 to 10, could increase the cost to US$ 800.

Even though the GPT associated monetary costs, these results show the potential of the GPT models for the codification task. Even with no hyperparameter search and with a low amount of coded properties, the fine-tuned model has better results for one

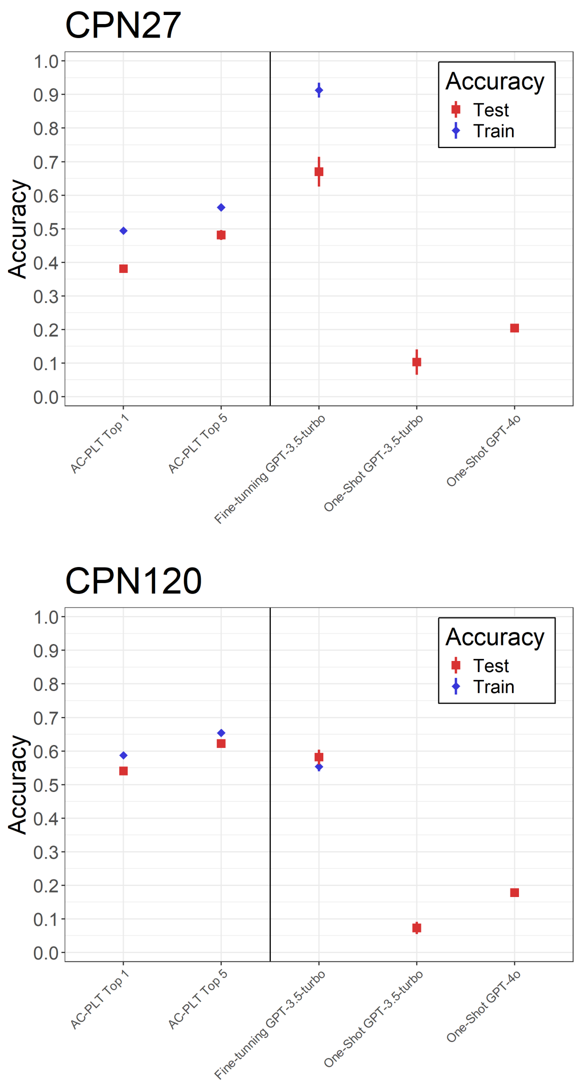


Figure 4: Experiment results of the LLMs models (right) compared to the AC-PLT model (left), using datasets CPN27 (top) and CPN120 (bottom).

dataset (CPN27) and no significant difference for the second dataset (CPN120), showing promising results for future work.

5 Conclusion

This paper proposed Chat-GPT for codifying Property Listing Tasks using one-shot prompting and fine-tuning. One-shot prompting with GPT-3.5-turbo-0125 and GPT-4o shows that a direct application of these models cannot replicate human coding. On the contrary, the fine-tuned model GPT-3.5-turbo-0125 obtained similar or even better performance than AC-PLT, the current state-of-the-art tool for this problem. Even though this is an initial study, these promising results can be a step forward in automating the codification process.

Acknowledgments

This work was supported by the James S. McDonnell Foundation grant, "A method for studying shared meaning in cultural groups" (https: //doi.org/10.37717/2022-3930) and by ANID, FONDECYT grant 1240075.

References

[1] Meysam Alizadeh, Maël Kubli, Zeynab Samei, Shirin Dehghani, Mohammad-masiha Zahedivafa, Juan D. Bermeo, Maria Korobeynikova, and Fabrizio Gilardi. 2024. Open-source llms for text annotation: a practical guide for model setting and fine-tuning. *Journal of Computational Social Science*, 8, 1, (Dec. 2024), 17. doi:10.1007/s42001-024-00345-9.

[2] Julia Amunts, Julia A. Camilleri, Simon B. Eickhoff, Kaustubh R. Patil, Stefan Heim, Georg G. von Polier, and Susanne Weis. 2021. Comprehensive verbal fluency features predict executive function performance. *Scientific Reports*, 11, 1, (Mar. 2021), 6929. doi:10.1038/s41598-021-85981-1.

[3] Marianna Bolognesi, Roosmaryn Pilgram, and Romy van den Heerik. 2017. Reliability in content analysis: the case of semantic feature norms classification. *Behavior Research Methods*, 49, 1984–2001, 6. doi:10.3758/s13428-016-0838-6.

[4] Tom B. Brown et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*. Vol. 2020-December. doi:10.48550/arXiv.2005.14165.

[5] Erin M. Buchanan, Simon De Deyne, and Maria Montefinese. 2020. A practical primer on processing semantic property norm data. *Cognitive Processing*, 21, 4, (Nov. 2020), 587–599. doi:10.1007/s10339-019-00939-6.

[6] Enrique Canessa, Sergio E Chaigneau, Rodrigo Lagos, and Felipe A Medina. 2021. How to carry out conceptual properties norming studies as parameter estimation studies: lessons from ecology. *Behavior Research Methods*, 53, 1, 354–370.

[7] Erminio Capitani, Marcella Laiacona, and Riccardo Barbarotto. 1999. Gender affects word retrieval of certain categories in semantic fluency tasks. *Cortex*, 35, 2, 273–278. doi:https://doi.org/10.1016/S0010-9452(08)70800-1.

[8] Cristian Cardellino. 2019. Spanish billion words corpus and embeddings. (Mar. 2019). https://crscardellino.github.io/SBWCE/.

[9] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20, 1, 37–46. doi:10.1177/001316446002000104.

[10] Barry J Devereux, Lorraine K Tyler, Jeroen Geertzen, and Billi Randall. 2014. The centre for speech, language and the brain (cslb) concept property norms. *Behavior Research Methods*, 46, 1119–1127, 4. doi:10.3758/s13428-013-0420-4.

[11] Luoma Ke, Song Tong, Peng Cheng, and Kaiping Peng. 2025. Exploring the frontiers of llms in psychological applications: a comprehensive review. (2025). https://arxiv.org/abs/2401.01519 arXiv: 2401.01519 [cs.LG].

[12] Arina Kostina, Marios D. Dikaiakos, Dimosthenis Stefanidis, and George Pallis. 2025. Large language models for text classification: case study and comprehensive review. (2025). https://arxiv.org/abs/2501.08457 arXiv: 2501.08457 [cs.CL].

[13] Gerhard Kremer and Marco Baroni. 2011. A set of semantic norms for german and italian. *Behavior Research Methods*, 43, 97–109, 1. doi:10.3758/s13428-010-0028-x.

[14] Klaus Krippendorff. 2004. Reliability in content analysis: some common misconceptions and recommendations. *Human Communication Research*, 30, (July 2004), 411–433, 3, (July 2004). doi:10.1111/j.1468-2958.2004.tb00738.x.

[15] Abhilasha A. Kumar. 2021. Semantic memory: a review of methods, models, and current challenges. *Psychonomic Bulletin & Review*, 28, 1, (Feb. 2021), 40–80. doi:10.3758/s13423-020-01792-x.

[16] Julieta Laurino, Simon De Deyne, Álvaro Cabana, and Laura Kaczer. 2023. The pandemic in words: tracking fast semantic changes via a large-scale word association task. *Open Mind*, 7, (June 2023), 221–239. eprint: https://direct.mit.edu/opmi/article-pdf/doi/10.1162/opmi_a_00081/2133848/opmi_a_00081.pdf. doi:10.1162/opmi_a_00081.

[17] Alessandro Lenci, Marco Baroni, Giulia Cazzolli, and Giovanna Marotta. 2013. Blind: a set of semantic feature norms from the congenitally blind. *Behavior Research Methods*, 45, 1218–1233, 4. doi:10.3758/s13428-013-0323-4.

[18] Chandreen R. Liyanage, Ravi Gokani, and Vijay Mago. 2024. Gpt-4 as an x data annotator: unraveling its performance on a stance classification task. *PLOS ONE*, 19, 8, (Aug. 2024), 1–21. doi:10.1371/journal.pone.0307741.

[19] Ken McRae, George S Cree, Mark S Seidenberg, and Chris Mcnorgan. 2005. Semantic feature production norms for a large set of living and nonliving things. *Behavior Research Methods*, 37, 547–559, 4. doi:10.3758/BF03192726.

[20] [SW] Ines Montani, Matthew Honnibal, Matthew Honnibal, Adriane Boyd, Sofie Van Landeghem, and Henning Peters, explosion/spaCy: v3.7.2: Fixes for APIs and requirements version v3.7.2, Oct. 2023. doi:10.5281/zenodo.10009823, URL: https://doi.org/10.5281/zenodo.10009823.

[21] Diego Ramos, Sebastián Moreno, Enrique Canessa, Sergio E Chaigneau, and Nicolás Marchant. 2023. Ac-plt: an algorithm for computer-assisted coding of semantic property listing data. *Behavior Research Methods*, 1–14. doi:10.3758/s13428-023-02260-9.

[22] J. Nick Reid and Albert Katz. 2022. The rk processor: a program for analysing metaphor and word feature-listing data. *Behavior Research Methods*, 54, 1, (Feb. 2022), 174–195. doi:10.3758/s13428-021-01564-y.

[23] R Selva Birunda S. and Kanniga Devi. 2021. A review on word embedding techniques for text classification. In *Innovative Data Communication Technologies and Application*. Springer Singapore, (Feb. 2021), 267–281. ISBN: 978-981-15-9651-3. doi:10.1007/978-981-15-9651-3_23.

[24] Xiao-Kun Wu et al. 2025. Llm fine-tuning: concepts, opportunities, and challenges. *Big Data and Cognitive Computing*, 9, 4. doi:10.3390/bdcc9040087.