From Self-Consistency to MedPrompt: Improving Product Classification Results by Ensembling LLMs

Florian Bauer

University of Mannheim, Germany
florian.bauer@students.uni-mannheim.de

Abstract. Large language models such as GPT-3.5-Turbo show remarkable capabilities in a variety of tasks, two of which are reasoning and product classification. Self-Consistency and MedPrompt are two prompting techniques that enhance the reasoning capabilities of language models without the need to alter the learned parameters. In this paper, I build on the findings of Self-Consistency and MedPrompt and apply them to a product classification task to test their generalizability and enhance product classification performance of large language models. However, my results fail to show a statistically significant performance improvement compared to the baseline. Two possible explanations for this are that either the sample size is not large enough to show significance, or that Self-Consistency and MedPrompt indeed do not improve production classification results, as reasoning and product classification rely on different abilities and possibly might not be transferable to each other.

1 Introduction

Large language models recently showed outstanding performance in various natural language processing (NLP) tasks, such as text generation and answering complex questions.

One important type of tasks for language models are reasoning tasks. However, these models traditionally underperform on reasoning tasks such as arithmetical and logical reasoning, compared to models fine-tuned for this specific use case [7]. Self-Consistency [22] and MedPrompt [17] are two techniques designed to mitigate this under-performance and improve reasoning capabilities of LLMs.

Another category of LLM tasks just as important are classification tasks. Classification tasks rely much more on knowing and correctly assigning factual knowledge to different classes, than applying multiple connected logical steps as reasoning tasks require. From text classification to product classification, LLMs achieve remarkable results on this type of tasks, if the correct prompting techniques are applied [2, 20]. An especially important classification task is product classification. Since e-commerce providers need to integrate huge sets of external data into their category hierarchy, it is crucial for the business that product classification can be performed automatically and provides reliable results.

In this paper, I present two reasoning techniques, Self-Consistency and Choice Shuffling, which is part of MedPrompt. Although being designed for a substantially different type of task, I apply them on product classification to test their generalizability to this important tasks. The task consists of classifying products into a corresponding multi-level category hierarchy. After collecting them, I evaluate the classification results based on various statistical measures, and discuss whether Self-Consistency and Choice Shuffling are appropriate techniques to enhance product classification performance of LLMs. Finally, I present various extension points to further verify and improve the results documented in this paper.

2 Background

2.1 Large Language Models

Large language models (LLM) refer to a specific kind of *pre-trained language models (PLM)* [13, 26]. PLMs are generative language models for natural language processing [16] that take a textual input and generate a corresponding textual output [21]. Research shows that scaling PLMs' model size leads to an improved model capacity with improved performance on various tasks, leading to the introduction of LLMs [8]. LLMs are transformer-based language models containing billions of parameters[1] that are trained on vast text data. During the training process, the model's parameters are fine-tuned to arrive closer at a desired output [5]. Researches find that model performance rises significantly when its parameter scale rises above a certain level. This effect is called *emergent abilities of LLMs* [28]. Hence, if large enough, LLMs show strong capacities to understand and generate natural language, and to solve complex tasks such as question answering.

2.2 Large Language Models on Product Classification

Besides processing and generation of natural language, LLMs also show remarkable performance in classification tasks. Research shows that in tasks of text classification, LLMs are able to outperform models specifically fine-tuned with supervised learning [20].

Examinations on product classification show that LLMs as transformer-based models also outperform traditional text-based classification techniques [2]. Product classification refers to a task of systematically categorizing products into specific groups based on shared features. This process aids in better organizing a product catalogue, which enhances the user experience for online shopping [9]. As this task is highly reliant on factual knowledge about product features and on correctly assigning those features to different categories, LLMs can make use of the extensive knowledge implicitly stored in the model's parameters which they learn from being trained on vast databases [18].

[1] In existing literature, there is no formal consensus on the minimum parameter scale for LLMs, since the model capacity is also related to data size and total compute

2.3 Fundamental Prompting Techniques

Prompting refers to text given as input to a LLM to guide the LLM to generating a certain desired textual output. Various studies have shown that performance of models can be heavily improved by using a prompt suitable for the specific task, without having to alter (i.e., fine-tuning) the model's learned parameters [24]. The process of creating such a suitable prompt is called *Prompt engineering*. Here, I briefly introduce some of the most relevant and fundamental prompt engineering concepts.

Few-Shot Prompting is a widely used and one of the most effective methods of *In-Context-Learning (ICL)*. ICL refers to the ability of language models to solve new tasks from just a few task demonstrations [3]. While standard zero-shot prompts only consist of natural language task descriptions [11], few-shot prompts consist of a test questions preceded by one or multiple different example questions and their expected outputs. The model then learns from the given examples and applies the learnings to the presented test question. The examples can have a similar effect on the model as parameter fine-tuning without having to update the parameters' weights. Hence, the choice of examples used as the few-shot context has a substantial influence on the model performance, introducing an important role for the example selection process.

Chain-of-Thought-Prompting (CoT) is a prompting technique that incorporates intermediate reasoning steps. By decomposing complex problems into smaller steps, CoT can solve these easier problems before assembling the answers and giving the final answer [23]. This decomposition of the problem and composition of the answers should help the model to generate a more accurate answer for tasks like math word problems and commonsense reasoning. CoT can be applied in zero-shot prompting, by adding a sentence like "Let's think step by step" to the prompt [11], or in few-shot prompting, by directly integrating the intermediate reasoning steps of CoT into the few-shot demonstrations [23].

Ensembling serves as a technique for combining the outputs of multiple model runs, enhancing robustness and accuracy of the answers [25]. The output of each run is stored and the final answer is selected via combining the separate outputs with a function like majority vote, averaging, or consensus. The diversity of outputs can be regulated by adjusting the temperature parameter in the model's generation process. However, while ensembling can enhance the robustness and accuracy of a model's outputs, it also increases the computational costs. As several runs need to be performed to arrive at a conclusion, the prompt's complexity must be considered carefully to avoid unnecessary and excessive costs.

2.4 Self-Consistency

Self-Consistency is a prompting technique based on CoT and ensembling. In tasks requiring deliberate thinking, there are likely several possible ways to ap-

proach the problem, and different humans apply different approaches to solve the problem. The idea behind Self-Consistency is to imitate this diversity in the human thinking process with LLMs via sampling from the language model's decoder.

To do so, multiple pairs of reasoning path and candidate outputs are sampled from the language model's decoder. The diversity of the generated outputs can be regulated by adjusting the model's temperature. Then, the different candidate outputs are aggregated by choosing the most consistent answer via majority vote, i.e., via $arg\ max_a \sum_{i=1}^{m} (a_i = a)$.

Given that LLMs are not perfect reasoners, they might make mistakes in their reasoning, leading to incorrect answers. However, such reasoning mistakes are not likely to occur identically multiple times. Hence, incorrect reasoning paths are less likely to arrive at the same incorrect answer and thus, it can be hypothesized that the correct reasoning paths have greater agreement than incorrect paths [22].

2.5 MedPrompt

MedPrompt is a similar approach as Self-Consistency, designed to more accurately answer multiple-choice questions [17]. It consists of preprocessing and inference steps that can be broadly categorized into the two parts Dynamic Few-shot and Choice Shuffling Ensemble:

Dynamic Few-shot. Dynamic Few-shot refers to an ICL-technique that automatically generates context examples and dynamically chooses suitable examples from the generated example pool. Traditionally, few-shot exemplars have to be handcrafted by domain experts [19], which is very costly and can be influenced by subjective perception. Dynamic few-shot makes uses of the large training dataset and selects automatically pre-generated exemplars by their similarity to the test question [14].

Preprocessing. First, an embedding vector is calculated for each training question. Then, a LLM (GPT-4 in the paper) generates a CoT rationale for each training question. To mitigate the risk of hallucinated or incorrect reasoning paths, the LLM also generates an answer following the generated rationale. If this answer does not match the ground truth answer, the sample entity is discarded, assuming that the reasoning too is incorrect. If the generated answer matches the ground truth answer, the rationale is assumed to be correct and the tuple of (Embedding vector, CoT rationale) is stored for the respective training question.

Inference. During each inference step, an embedding vector is calculated for the test question at hand. Then, the k closest training questions are selected using a k-NN clustering in the embedding space, and using the cosine similarity as similarity measure. For each of the k example questions, the CoT rationales generated during preprocessing are given to the prompt as few-shot context.

Choice Shuffling Ensemble. Language models can show position bias [10, 27], i.e., a bias to favor labels at certain positions in multiple choice questions over others, regardless of the label content [1]. To mitigate this bias, MedPrompt generates multiple answers for a single test question, with shuffled answer choices during each run, i.e., a random permutation $\pi = (\pi(1), \pi(2), ..., \pi(n))$ for n answer choices. This introduces diversity to the generated reasoning paths, which supports the model more robustly and more accurately selecting the correct answer. Subsequently, the most consistent answer, i.e., the answer that is least sensitive to choice shuffling, is selected via majority vote.

3 Experimental Design

3.1 Task

While both Self-Consistency and MedPrompt are designed to enhance the reasoning ability of language models, I apply them to product classification to test the generalizability of both methods on other tasks. Product classification generally relies much more on factual knowledge and less on the reasoning ability. The classification task involves classifying a specified product into its multi-level product category hierarchy.

3.2 Dataset

I evaluate Self-Consistency and MedPrompt on a dataset composed from the Icecat Product Catalogue[2]. I selected 25 different category hierarchy paths that obey to the format '*Computers & Electronics>Second-level Category>Third-level Category*', resulting in 10 different second-level and 25 different third-level categories. As the first-level category is fixed to 'Computers & Electronics', the task is to predict the second-level and the third-level categories. For each of the 25 category paths, I randomly sampled 4 sample products to a total sample size of n = 100, and stored their names, brands, and ground truth category paths[3].

3.3 Prompts

For evaluating Self-Consistency and MedPrompt on the product classification task, I use four different prompting methods. Fig. 1 shows the basic prompt that each of the methods uses. Here, I explain the four different prompting methods.

- **Baseline.** Greedy chain-of-thought decoding, as it has been previously used for decoding in large language models [4]. It generates one single answer which is output as the final answer.

[2] Icecat Product Catalogue available at `https://icecat.biz/`
[3] Dataset and code available at `https://github.com/floribau/ClassifyGPT/`

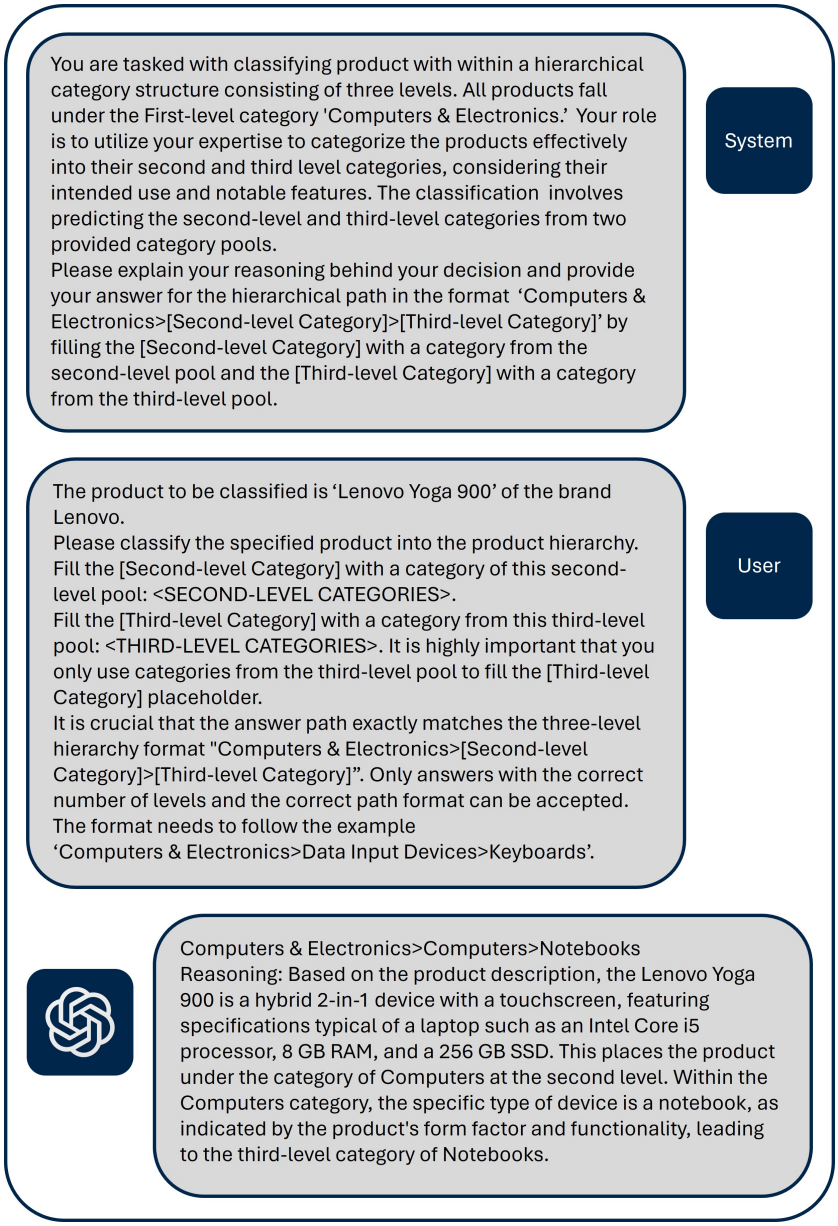

Fig. 1. Basic prompt to generate a single category path. The prompt consists of a system prompt explaining the general setting, and a user prompt introducing the specific product. <SECOND-LEVEL CATEGORIES> and <THIRD-LEVEL CATEGORIES> are filled with the respective lists of choices. The last text shows an example output generated by the language model.

- **Self-Consistency.** Applies the Self-Consistency method on the presented prompt. It generates an output for each of k different model runs and then aggregates the k outputs to one single answer via majority vote. In this experiment, I use k = 3, i.e., Self-Consistency generates three outputs which are then aggregated to one final answer. A way to introduce diversity to the generated output is via the model's temperature. In order to imitate different levels of creativity, I applied temperature sampling with the temperatures T = 0, T = 0.5, and T = 1.
- **Choice Shuffling.** Applies the Choice Shuffling method presented in MedPrompt on the prompt. Since my work focuses on Prompt Engineering and to not unfairly distort the comparison with Self-Consistency, I don't incorporate the Dynamic Few-shot prompting, but restricte the MedPrompt examination on Choice Shuffling. It generates an output for each of m different model runs and then aggregates the m outputs to one single answer via majority vote. In this experiment, I use m = 3. For each run, the order of the answer choices is shuffled by a different permutation π_i.
- **Combination.** Combines Self-Consistency and Choice Shuffling methods. It therefore generates an output for each of $k * m$ different model runs with k = 3, m = 3: Each temperature $T_j \in \{0, 0.5, 1\}$ used in Self-Consistency is paired with each permutation π_i used in Choice Shuffling. Thereafter, the $k * m$ outputs are aggregated to one single answer via majority vote.

3.4 Models

I evaluate the four methods on the following two transformer-based language models[4]. All experiments are performed without training or fine-tuning the language models.

- **GPT-3.5-Turbo-0125**[5]**.** The latest GPT-3.5-Turbo model developed by OpenAI with higher accuracy at responding in requested formats.
- **GPT-3.5-Turbo-0613**[6]**.** An older GPT-3.5-Turbo model which will be deprecated on June 13, 2024. This older model has been selected to resemble the testing environment used in the Self-Consistency paper, which was an even older model with GPT-3 [22].

4 Results

4.1 Evaluation Methods

For evaluating the performances of the different prompting methods, I make use of the hierarchy format *Computers & Electronics>Second-level Category>Third-level Category* of the Icecat product catalogue. This hierarchy allows to separately evaluate the performances on the whole path, the second-level categories,

[4] Public API available at `https://openai.com/api/`

[5] Documentation for GPT-3.5-Turbo-0125 and GPT-3.5-Turbo-0613 available at `https://platform.openai.com/docs/models/gpt-3-5-turbo/`

[6] See footnote 5.

and the third-level categories. Since the first-level category is always 'Computers & Electronics', it is not necessary to evaluate the performance on first-level categories individually.

As evaluation measure, I use the F-score, which is calculated from the precision and recall. Since working with multi-class classification, I evaluate the micro- and macro-values for the F-score, where the micro-value is an instance-pivoted measure giving equal weight to every instance, and the macro-value is an category-pivoted measure giving equal weight to every category. Micro-precision and micro-recall are defined as

$$P_{micro} = \frac{\sum_{i=1}^{|C|} tp_i}{\sum_{i=1}^{|C|} tp_i + fp_i}, \; R_{micro} = \frac{\sum_{i=1}^{|C|} tp_i}{\sum_{i=1}^{|C|} tp_i + fn_i}, \tag{1}$$

and macro-precision and macro-recall are defined as

$$P_{macro} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{tp_i}{tp_i + fp_i}, \; R_{macro} = \frac{1}{|C|} \sum_{i=1}^{|C|} \frac{tp_i}{tp_i + fn_i}, \tag{2}$$

with tp_i being the true positives for class i, fp_i being the false positives for class i, and fn_i being the false negatives for class i. The F-score is then composed from both precision and recall and is defined as

$$F_{1,micro} = \frac{2 \cdot P_{micro} \cdot R_{micro}}{P_{micro} + R_{micro}}, \; F_{1,macro} = \frac{2 \cdot P_{macro} \cdot R_{macro}}{P_{macro} + R_{macro}} \tag{3}$$

I then compare the results achieved by Self-Consistency, Choice Shuffling, and Combination to those achieved by the Baseline[7].

4.2 Results with GPT-3.5-Turbo-0125

Micro F-scores. The results on GPT-3.5-Turbo-0125 evaluated with the micro F-score are shown in Table 1. The values at 'Δ <Method>, Baseline' show the comparison in performance between each prompting method and the Baseline as $F_{1,<Method>} - F_{1,Baseline}$.

As can be seen from the Δs in Table 1, Choice Shuffling and Combination perform equally or worse than the Baseline on every evaluated level regarding the instance-pivoted evaluation. Self-Consistency performs worse on second-level categories, but slightly better on third-level categories.

Macro F-scores. The results evaluated with the macro F-score are shown in Table 2. Regarding this category-pivoted evaluation, Choice-Shuffling and Combination again perform equally or worse than the Baseline on every level. Particularly, Combination performs considerably worse than the Baseline on second-level categories (Δ = -0.13). Here, Self-Consistency performs better on second-level categories, but worse on third-level categories.

[7] Results available at `https://github.com/floribau/classify-gpt/tree/main/Results/`

Table 1. Performance of different prompting methods on GPT-3.5-Turbo-0125. Micro F-score as scoring measure.

Prompt Method	Path Micro-F1	Second-level Micro-F1	Third-level Micro-F1
Baseline	**0.79**	**0.87**	0.89
Self-Consistency	**0.79**	0.85	**0.90**
Δ Self-Consistency, Baseline	+0.0	-0.02	+0.01
Choice Shuffling	0.74	0.84	0.89
Δ Choice Shuffling, Baseline	-0.05	-0.03	+0.0
Combination	0.74	0.83	0.88
Δ Combination, Baseline	-0.05	-0.04	-0.01

Table 2. Performance of different prompting methods on GPT-3.5-Turbo-0125. Macro F-score as scoring measure.

Prompt Method	Path Macro-F1	Second-level Macro-F1	Third-level Macro-F1
Baseline	**0.58**	0.87	**0.88**
Self-Consistency	0.56	**0.88**	0.87
Δ Self-Consistency, Baseline	-0.02	+0.01	-0.01
Choice Shuffling	0.55	0.84	**0.88**
Δ Choice Shuffling, Baseline	-0.03	-0.03	+0.0
Combination	**0.58**	0.75	0.83
Δ Combination, Baseline	+0.0	-0.13	-0.05

4.3 Results with GPT-3.5-Turbo-0613

Micro F-scores. The results on GPT-3.5-Turbo-0613 evaluated with the micro F-score are shown in Table 3. Now, Self-Consistency performs better than the Baseline on the whole paths and second-level categories, but worse on third-level categories. Choice Shuffling performs better than the Baseline on all evaluated levels. Combination performs worse than the Baseline on the whole paths and second-level categories, but performs better on third-level categories.

Macro F-scores. The results evaluated with the macro F-score are shown in Table 4. Regarding this category-pivoted evaluation, Self-Consistency performs better than the Baseline on the whole paths, equally on second-level categories, and worse on third-level categories. Choice Shuffling performs better on all evaluated levels, while particularly performing considerably better on second-level categories and on third-level categories ($\Delta = +0.08$ for both levels). Combination performs worse than the Baseline on the whole paths, but performs better on second-level and on third-level categories.

4.4 Error Analysis

In this section, I analyze the errors made by the classifier. I count the related false positives fp and false negatives fn on second-level and third-level categories and

Table 3. Performance of different prompting methods on GPT-3.5-Turbo-0613. Micro F-score as scoring measure.

Prompting Method	Path Micro-F1	Second-level Micro-F1	Third-level Micro-F1
Baseline	0.81	0.88	0.90
Self-Consistency	0.82	0.89	0.89
Δ Self-Consistency, Baseline	+0.01	+0.01	-0.01
Choice Shuffling	**0.85**	**0.90**	**0.95**
Δ Choice Shuffling, Baseline	+0.04	+0.02	+0.05
Combination	0.78	0.86	0.91
Δ Combination, Baseline	-0.03	-0.02	+0.01

Table 4. Performance of different prompting methods on GPT-3.5-Turbo-0613. Macro F-score as scoring measure.

Prompting Method	Path Macro-F1	Second-level Macro-F1	Third-level Macro-F1
Baseline	0.64	0.81	0.87
Self-Consistency	**0.67**	0.81	0.85
Δ Self-Consistency, Baseline	+0.03	+0.0	-0.02
Choice Shuffling	0.65	**0.89**	**0.95**
Δ Choice Shuffling, Baseline	+0.01	+0.08	+0.08
Combination	0.62	0.86	0.90
Δ Combination, Baseline	-0.02	+0.05	+0.03

present the most common mistakes. I then discuss implications resulting from these mistakes.

False Classifications on Second-level. On second-level categories, the category 'TVs & Monitors' is mistaken for 'Computers' most often with 27 times overall. Other common faults are mistaking 'Telecom & Navigation' for 'TVs & Monitors' (13 times), 'Projectors' for 'TVs & Monitors' (13 times), 'Computer Cables' for 'Networking' (12 times), and 'Computers' for 'Telecom & Navigation' (11 times).

As can be seen in Table 5, the errors are spread evenly across all combinations of prompting method and language model, hence no combination showing significantly worse performances than the others on on any category. Furthermore, all of the 10 second-level categories are selected at least once as either false negative or false positive. However, it is remarkable that the categories 'TVs & Monitors' (31x false negative, 26x false positive), 'Computers' (14x false negative, 33x false positive), and 'Telecom & Navigation' (20x false negative, 11x false positive) are often selected as false negatives as well as often selected as false positives. Hence, the language model seems to be uncertain of the definitions of these categories, and adding a definition to resolve this uncertainty might improve the performance on second-level categories.

Table 5. Mistakes made on second-level categories. The most common mistakes include the categories 'TVs & Monitors', 'Computers', and 'Telecom & Navigation'.

Language Model	Prompting Method	Highest			Second-highest		
		Gold Standard	Prediction	Freq.	Gold Standard	Prediction	Freq.
GPT-3.5-Turbo-0125	Baseline	TVs & Monitors	Computers	3x	Computers	Telecom & Navigation	3x
	Self-Consistency	TVs & Monitors	Computers	3x	Telecom & Navigation	TVs & Monitors	2x
	Choice Shuffling	TVs & Monitors	Computers	4x	Projectors	TVs & Monitors	3x
	Combination	TVs & Monitors	Computers	4x	Projectors	TVs & Monitors	4x
GPT-3.5-Turbo-0613	Baseline	TVs & Monitors	Computers	3x	Computers	Telecom & Navigation	3x
	Self-Consistency	Computers	Telecom & Navigation	3x	Telecom & Navigation	TVs & Monitors	2x
	Choice Shuffling	TVs & Monitors	Computers	4x	Computer Cables	Networking	1x
	Combination	TVs & Monitors	Computers	4x	Computer Cables	Networking	3x

Table 6. Mistakes made on third-level categories. The most common mistakes confuse 'Software Licenses/Upgrades' for 'Antivirus Security Software', 'Tablets' for 'Notebooks', and 'TVs' for 'Flat Panel Spare Parts'.

Language Model	Prompting Method	Highest Freq.			Second-highest Freq.		
		Gold Standard	Prediction	Freq.	Gold Standard	Prediction	Freq.
GPT-3.5-Turbo-0125	Baseline	Tablet Cases	Mobile Phone Cases	2x	Software Lic./Upgr.	Antivirus Sec. Software	2x
	Self-Consistency	Software Lic./Upgr.	Antivirus Sec. Software	3x	Tablets	Notebooks	1x
	Choice Shuffling	Software Lic./Upgr.	Antivirus Sec. Software	3x	TVs	Flat Panel Spare Parts	2x
	Combination	TVs	Flat Panel Spare Parts	2x	Software Lic./Upgr.	Antivirus Sec. Software	2x
GPT-3.5-Turbo-0613	Baseline	Software Lic./Upgr.	Antivirus Sec. Software	2x	Tablets	Notebooks	1x
	Self-Consistency	Tablet Cases	Mobile Phone Cases	2x	Software Lic./Upgr.	Antivirus Sec. Software	2x
	Choice Shuffling	Software Lic./Upgr.	Antivirus Sec. Software	2x	Tablets	Notebooks	1x
	Combination	TVs	Flat Panel Spare Parts	3x	Software Lic./Upgr.	Antivirus Sec. Software	2x

False Classifications on Third-level. On third-level categories, the classifier most often mistakes 'Software Licenses/Upgrades' for 'Antivirus Security Software' (18 times overall). Other common errors are mistaking 'Tablets' for 'Notebooks' (8 times), 'TVs' for 'Flat Panel Spare Parts' (7 times).

As can be seen in Table 6, the errors again are spread across the different language model and prompting method combinations. However, the classifier seems to systematically confuse category pairs, as the categories mistaken for each other are always identical. Thus, adding descriptions specifically clarifying the differences between these categories might enhance performance on third-level categories.

4.5 Statistical Analysis

Considering the results discussed in Chapter 4.2 and 4.3, it is striking that most performances are highly similar across all prompting methods. The only prompting methods to apparently not hold an about equal performance, are Combination on GPT-3.5-Turbo-0125 which performs considerably worse than the Baseline on named model, and Choice Shuffling on GPT-3.5-Turbo-0613, which performs considerably better than the Baseline on the same model. Since the error analysis in Chapter 4.4 does not provide any pattern explaining the performance differences of the different prompting methods, a statistical analysis is to be performed to examine the observations.

Cohen's kappa. To analyze the agreement between a given prompting method and the Baseline, I evaluate Cohen's kappa. Cohen's kappa allows to assess the agreement between two methods for multi-class classification. The kappa is calculated as

$$\kappa = \frac{p_0 - p_e}{1 - p_e} \ , \tag{4}$$

where p_0 is the relative observed agreement, and p_e is the hypothetical probability of chance agreement. The value of κ ranges from -1 to 1, with $\kappa < 0$ being less-than-chance agreement, $\kappa = 0$ being chance agreement, $\kappa > 0$ being better-than-chance agreement, and $\kappa = 1$ being perfect agreement.

Table 7 shows the Cohen's kappas for each model compared with the Baseline on both language models. A commonly used scale for interpreting Cohen's kappa categorizes $0.61 \leq \kappa \leq 0.80$ as 'Substantial Agreement', and $0.81 \leq \kappa \leq 1.00$ as '(Almost) Perfect Agreement' [12]. When looking at the results in Table 7, it can be observed that almost all values for kappa are $\kappa > 0.80$ and thus, represent (almost) perfect agreement. The only value $\kappa < 0.80$ is for the kappa between Combination and Baseline on path-level on GPT-3.5-Turbo-0125 with $\kappa = 0.76$. Hence, the kappa here still represents substantial agreement.

McNemar's test. From the analysis of the agreement with Cohen's kappa between the different prompting methods and the Baseline, it can be observed that the different prompting methods possess a high agreement. To examine

Table 7. Cohen's kappa between each prompting method and the Baseline on both models.

Language Model	Prompting Methods	Path-level	Second-level	Third-level
GPT-3.5-Turbo-0125	Self-Consistency, Baseline	0.88	0.91	0.91
	Choice Shuffling, Baseline	0.83	0.90	0.90
	Combination, Baseline	0.76	0.84	0.86
GPT-3.5-Turbo-0613	Self-Consistency, Baseline	0.92	0.94	0.95
	Choice Shuffling , Baseline	0.89	0.92	0.94
	Combination, Baseline	0.84	0.87	0.93

whether the relatively small performance differences are statistically significant, a McNemar's test is applied to the results [15]. The McNemar's test assesses the statistical significance of changes from one prompting method to another. The null hypothesis of the McNemar's test states that the two marginal probabilities for each outcome are the same, i.e., the null and alternative hypotheses are $H_0 : p_b = p_c$ and $H_1 : p_b \neq p_c$. Here, b represents the case that the first method correctly classifies a product but the second method miclassifies it, and c represents the case that the first method misclassifies the product but the second method correctly classifies it. The test statistic is chosen based on the number of samples that fall into the b and c categories: If $b + c \geq 25$, the approximated χ^2-distribution is used, and otherwise, i.e., if $b + c < 25$, an exact binomial distribution [6] is applied. Since $b + c < 25$ for all combinations (see Table 8), the exact binomial test must be used, for which the p-value is calculated as

$$p = 2 \cdot \sum_{i=b}^{n} \binom{n}{i} \cdot 0.5^i \cdot (1 - 0.5)^{n-i} \tag{5}$$

Table 8. The values of $b+c$ between each prompting method and the Baseline on both language models. It holds $b + c < 25$ for every examined combination.

Language Model	Prompting Methods	Path-level	Second-level	Third-level
GPT-3.5-Turbo-0125	Self-Consistency, Baseline	8	6	7
	Choice Shuffling, Baseline	11	7	10
	Combination, Baseline	19	14	13
GPT-3.5-Turbo-0613	Self-Consistency, Baseline	7	5	5
	Choice Shuffling , Baseline	8	6	5
	Combination, Baseline	11	10	5

I set $\alpha = 0.05$ as the significance level. Table 9 shows the p-values as results of the evaluation with the exact McNemar's test. When evaluating the calculated p-values, it can be observed that all except one p-value are considerably higher than the significance level $\alpha = 0.05$ needed to reject the zero hypothesis. The only value that comes close to the significance level is the p-value between Choice

Shuffling and the Baseline on GPT-3.5-Turbo-0613 on the third-level categories with $p = 0.0625$. Since it is also widely accepted to set a more liberal significance level $\alpha = 0.1$, one could assume that this observation is significant. However, considering that all other performances, including the other performances of Choice Shuffling on the same language model and the performances of Choice Shuffling on third-level categories on GPT-3.5-Turbo-0125, are insignificant as well, it is still highly improbable that the performance difference between Choice Shuffling and the Baseline on third-level categories on GPT-3.5-Turbo-0613 is indeed statistically significant.

Table 9. P-values of McNemar's test for statistical significance in performance differences. The only difference that is in the range of potential statistical significance is between Choice Shuffling and the Baseline on third-level categories with GPT-3.5-Turbo-0613.

Language Model	Prompting Methods	Path-level	Second-level	Third-level
GPT-3.5-Turbo-0125	Self-Consistency, Baseline	1.0	0.6875	1.0
	Choice Shuffling, Baseline	0.2266	0.4531	1.0
	Combination, Baseline	0.3593	0.4240	1.0
GPT-3.5-Turbo-0613	Self-Consistency, Baseline	1.0	1.0	1.0
	Choice Shuffling , Baseline	0.2891	0.6875	**0.0625**
	Combination, Baseline	0.5488	0.7539	1.0

5 Conclusion

I presented background and methods for improving the ability to reason of large language models without relying on special parameter fine-tuning. I then tested the generalizability of the presented methods by applying them on the curious case of product classification into a multi-level category hierarchy using different GPT-3.5-Turbo models.

I described the four prompting methods used to measure the performances and the statistical tools to evaluate the measured results. However, the three more complex prompting methods Self-Consistency, Choice Shuffling, and Combination did not show any significant improvement in performance compared to the greedy-decoder Baseline.

A possible reason to this is that the sample size of $n = 100$ is not large enough to show statistically significant improvements. Another explanation could be that the more complex prompting methods in fact don't improve product classification results compared to the Baseline. As discussed previously in Chapter 3.1, the qualities needed for reasoning tasks and for product classification tasks differ substantially, since reasoning relies on applying and connecting logical reasoning steps, while product classification mainly relies on knowing and correctly identifying factual feature knowledge. This circumstance could lead to Self-Consistency and Choice Shuffling (MedPrompt) successfully improving reasoning capabilities

of large language models, but failing to improve product classification capabilities, since the factual knowledge basis is the same for the Baseline and the more complex prompting Methods.

As discussed in Chapter 2.3, 2.4, and 2.5, Self-Consistency and Choice Shuffling both rely on ensembling. Since ensembling includes generating multiple model outputs instead of only one, the computational costs are substantially higher than with prompting methods that only create one single output. Thus, in case that the more complex, multi-run prompting methods in fact fail to improve performance on product classification compared to the single-run Baseline, it is not recommended to use the more multi-run prompting methods for product classification due to the higher computational costs. Hence, further work needs to be done in order to verify whether the more complex prompting methods are able to improve product classification results or not.

6 Future Work

In Chapter 5, I conclude that further work needs to be done in the future to verify whether the more complex prompting methods for Self-Consistency, Choice Shuffling, and Combination indeed fail to improve product classification performance compared to the Baseline, or whether the sample size is simply too small to show statistically significant performance improvements. Here, I present multiple approaches to extend the experiments and verify the discussed issues in the future.

- **Increased sample size.** In my experiments, I use a sample size of $n = 100$ to limit the scope and resulting costs of the experiments. Possibly, this sample size is too small to show significant results, and hence, an increased sample size would be more appropriate to show statistical significance.
- **Increased number of generated paths.** In my experiments, I generated $k = 3$ paths for both Self-Consistency and Choice Shuffling to limit the computational costs. The papers presented in Chapters 2.4 and 2.5 document significant performance improvements with $k \geq 5$ generated paths. Thus, an increased number of generated paths would introduce even higher computational costs, but potentially could help show statistically significant improvements.
- **Adjusted temperature sampling.** In my experiment, I used the temperatures $T = 0$, $T = 0.5$, $T = 1$ for temperature sampling. Possibly, these temperatures are not optimal to perform temperature sampling, and the use of other temperatures might achieve better classification results.
- **Included Dynamic Few-shot.** MedPrompt, as presented in Chapter 2.5, introduces dynamic few-shot context as one substantial part of the method. However, since I focus on prompt engineering and to not unfairly distort the competition with Self-Consistency, I only use Choice Shuffling in my experiments. Likely, this harms the ability of MedPrompt, and better results could be achieved when also including the dynamic few-shot context for product classification.

References

1. Blunch, Niels J.: Position bias in multiple-choice questions. Journal of Marketing Research 21.2 (1984): 216-220.
2. Brinkmann, Alexander, and Bizer, Christian: Improving Hierarchical Product Classification using Domain-specific Language Modelling. IEEE Data Eng. Bull. 44.2 (2021): 14-25.
3. Brown, Tom, et al.: Language models are few-shot learners. Advances in neural information processing systems 33 (2020): 1877-1901.
4. Chowdhery, Aakanksha, et al.: Palm: Scaling language modeling with pathways. Journal of Machine Learning Research 24.240 (2023): 1-113.
5. Ding, Ning, et al.: Parameter-efficient fine-tuning of large-scale pre-trained language models. Nature Machine Intelligence 5.3 (2023): 220-235. https://doi.org/10.1038/s42256-023-00626-4
6. Fay, Michael P.: Exact McNemar's test and matching confidence intervals. (2011).
7. Huang, Jie, and Kevin Chen-Chuan Chang: Towards reasoning in large language models: A survey. arXiv preprint arXiv:2212.10403 (2022).
8. Kaplan, Jared, et al.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020).
9. Kiang, Melody Y., et al.: A service-oriented analysis of online product classification methods. Decision support systems 52.1 (2011): 28-39.
10. Ko, Miyoung, et al.: Look at the first sentence: Position bias in question answering. arXiv preprint arXiv:2004.14602 (2020).
11. Kojima, Takeshi, et al.: Large language models are zero-shot reasoners. Advances in neural information processing systems 35 (2022): 22199-22213.
12. Landis, J. Richard, and Koch, Gary G.: The measurement of observer agreement for categorical data. biometrics (1977): 159-174.
13. Li, Junyi, et al.: Pre-trained language models for text generation: A survey. ACM Computing Surveys 56.9 (2024): 1-39.
14. Liu, Jiachang, et al.: What Makes Good In-Context Examples for GPT-3?. arXiv preprint arXiv:2101.06804 (2021).
15. McNemar, Quinn: Note on the sampling error of the difference between correlated proportions or percentages. Psychometrika 12.2 (1947): 153-157.
16. Nadkarni, Prakash M., Ohno-Machado, Lucila, and Chapman, Wendy W.: Natural language processing: an introduction. Journal of the American Medical Informatics Association 18.5 (2011): 544-551.
17. Nori, Harsha, et al.: Can generalist foundation models outcompete special-purpose tuning? case study in medicine. arXiv preprint arXiv:2311.16452 (2023).
18. Petroni, Fabio, et al.: Language models as knowledge bases?. arXiv preprint arXiv:1909.01066 (2019).
19. Singhal, Karan, et al.: Large language models encode clinical knowledge. Nature 620.7972 (2023): 172-180.
20. Sun, Xiaofei, et al.: Text classification via large language models. arXiv preprint arXiv:2305.08377 (2023).
21. Wang, Haifeng, et al.: Pre-trained language models and their applications. Engineering (2022).
22. Wang, Xuezhi, et al.: Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171 (2022).
23. Wei, Jason, et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems 35 (2022): 24824-24837.

24. Yang, Chengrun, et al.: Large language models as optimizers. arXiv preprint arXiv:2309.03409 (2023).
25. Yang, Han, et al.: One LLM is not Enough: Harnessing the Power of Ensemble Learning for Medical Question Answering. medRxiv (2023).
26. Zhao, Wayne Xin, et al.: A survey of large language models. arXiv preprint arXiv:2303.18223 (2023).
27. Zheng, Lianmin, et al.: Judging llm-as-a-judge with mt-bench and chatbot arena. Advances in Neural Information Processing Systems 36 (2024).
28. Zoph, Barret, et al.: Emergent abilities of large language models. TMLR (2022).

7 Statement about Usage of AI Tools

Apart from conducting the experiment with OpenAI's API, I also used AI tools, especially large language models like ChatGPT, to help with various tasks related to the paper. Here, I discuss which tools I used for which tasks, and how useful they are.

7.1 AI for Summarizing Literature

For my literature research, I used ChatGPT to summarize parts of the related papers. This is highly useful, since ChatGPT shows remarkable capabilities to extract and reproduce the main statements of a given text. Letting ChatGPT summarize parts of the background papers helped me reading and understanding the texts better and faster, leading to more efficient literature research.

7.2 AI for Writing Prompts

I described the functionality needed for the basic prompt to ChatGPT and let it engineer the actual prompt. This was really useful, since ChatGPT knows prompt engineering techniques that I didn't know and hence could write a more effective prompt. Although the prompt was already highly effective, it resulted in a few unwanted effects, e.g., the response not always respecting the format. Thus, I needed to make some adjustments to the prompt, e.g., adding the demonstration example specifying the hierarchy format.

7.3 AI for Writing Code

I used ChatGPT to aid with writing code for the experiment. This included generating new code, which was only useful for getting a first idea of the code, as I had to adapt the generated code to fit the specific use case. A second, much more useful use case was fixing errors in the existing code. ChatGPT finds many errors faster than I do, leading to a much faster development process.

7.4 AI for Choosing Statistical Measures

For evaluating the results of my experiment, I made use of ChatGPT's knowledge for choosing appropriate statistical measures. It helped me understanding which statistical test is suitable to which use case and getting a deeper knowledge of how they work internally, hence being very useful.

7.5 AI for Writing Text

Lastly, I tried enhancing the writing process for this paper by letting ChatGPT write some paragraphs of the paper. This wasn't very useful, as ChatGPT's wording was too general and not specific enough to my use case, and hence I had to rephrase the sentences to apply them to this paper. Nonetheless, ChatGPT helped getting an idea of how to structure the paragraphs.