Universiteit Leiden
The Netherlands

Data Science and Artificial Intelligence

Comparison of LLM-based SotA approaches to solving data scarce intent detection in the few-shot setting

Matiss Dimins

First supervisor: I-Fan Lin (LIACS)
Second supervisor: Dr. S. Verberne (LIACS)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

29/06/2025

Abstract

As Internet users interact with online chatbots more and more, the task of intent detection becomes increasingly relevant. The recent rise of Large Language Models (LLMs) has resulted in a growing trend of LLM-powered chatbots. For many commercial applications, collecting a lot of training examples and then finetuning an LLM on the collected data for intent detection is not particularly feasible. Consequently, novel approaches have been developed that, given a few-shot setting, can solve intent detection well using LLMs for inference only. The three distinct approaches are: 1) *prompting LLM for the label directly*, 2) *training a small transformer encoder on LLM-augmented data*, 3) *extracting LLM sentence embeddings as predictions*. In this research, I compare and analyze the performance of the three approaches on the CLINC150 dataset using Llama-3.2-3B-Instruct. I found that the *extracting LLM sentence embeddings as predictions* approach notably outperformed the other two approaches on accuracy and OOS detection rate, while *prompting LLM for the label directly* consistently performed the worst. I performed approach-specific ablations that helped examine, which factors and metrics significantly contribute to the performance of each approach. Notable findings from the ablations include that more diversity does not equate to higher accuracy for the *training a small transformer encoder on LLM-augmented data* approach, and that cosine similarity is superior to the dot product as a classification metric for *extracting LLM sentence embeddings as predictions*.

Contents

1 Introduction

Natural Language Processing (NLP) is a field residing at the intersection between computer science, artificial intelligence, linguistics, statistics, and logic. The field is most notably concerned with instilling language skills in computers. This is by no means an easy task – NLP experts have to work around hardware limitations, the complexity of human language, the expressive power of current State-of-the-Art (SotA) architecture, and, depending on the task, the abundance or scarcity of human text. Regardless, significant progress has been made towards general language understanding and generation, in particular thanks to the most recent and most ground-breaking paradigm shift in NLP – the rise of Large Language Models (LLMs).

LLMs showcase both a broader and deeper understanding of human language than previous methods by leveraging the transformer architecture [VSP+17], as well as their enormous size. These key features allow LLMs to capture more unique relationships and interdependencies in human-generated text than previous solutions. The natural outcome of the wider applicability of LLMs is their introduction in various interactive and social contexts. Both in commercial and non-commercial settings, chatbots are now often the first point of contact for most online users. In a survey by Tidio[1], 88% of online users had at least one conversation with a chatbot in 2022. The advent of LLMs has consequently lead to a rising interest in LLM-powered chatbots [DHQZ24].

In order to deliver a serviceable chatbot experience and satisfy users, the underlying task-based chatbot must be capable of adequately performing the task of intent detection. At its core, intent detection is about determining the goal of the user, based on the user's query. In the context of this task, the goal of the user is defined as the user's *intent*, while the user's query is their *utterance*. Intent detection is a type of text classification task, where a model must accurately predict the intent of a given utterance. For example, when interacting with a chatbot on the website of an airline, a user might ask: *could you tell me the status of flight dl123*. The chatbot must then correctly predict that the intent is `flight_status` to be able to respond appropriately. The range of possible intents can be very diverse. A user may want to correspond with the chatbot in English, prompting the chatbot with *please respond to me in english from now on*, which the chatbot must correctly identify with the intent `change_language`. Some intents can be extremely fine-grained and specific, for example, *Hey, I thought my topup was all done but now the money is gone again – what's up with that?* should be classified with the intent `Reverted Top-up`, while *Tell me why my topup wouldn't go through?* has the intent `Failed Top-up`.

It is possible to fine-tune some of the largest and most powerful LLMs on the intent detection task to achieve strong results. However, for most purposes, this is not feasible. Fine-tuning in itself is a computationally expensive process. Moreover, the available data for fine-tuning is usually limited – it is an expensive and slow process to create human annotated (utterance, intent) pairs, and LLMs are known to be extremely data-hungry.

In response to the limitations outlined above, novel approaches have been developed for solving intent detection. These approaches are more data-efficient, requiring significantly less samples to

[1]The Future of Chatbots: 80+ Chatbot Statistics for 2025 - https://www.tidio.com/blog/chatbot-statistics/

achieve comparable results, and do not require fine-tuning on the LLM that is used for inference. These approaches are: 1) *prompting LLM for the label directly* (also, "Prompting LLM"), 2) *training a small transformer encoder on LLM-augmented data* (also, "Data augmentation"), 3) *extracting LLM sentence embeddings as predictions* (also, "Sentence embeddings"). However, there has been no research focused on systematically comparing these three approaches as a whole.

In this thesis, the performance of each approach will be systematically evaluated on an intent detection dataset. The primary goal is to determine which approach, given resource constraints, is most effective for solving the intent detection task. All three are implemented using the same LLM – Llama-3.2-3B-Instruct[2] [TLI+23] – and tested on the same dataset - CLINC150[3] [LMP+19]. The Llama model is used because it is open-source and has shown solid performance on standard NLP benchmarks[4]. Despite its comparatively small size, it has been pretrained on an enormous amount of data. CLINC150 is a staple benchmark in intent detection as it covers a wide range of domains and intents. The few-shot setting is used to adequately challenge each method and simulate data scarcity. The performance of each approach is evaluated on standard NLP metrics – accuracy, per-class F1 score, precision and recall, as well as confusion matrices. Qualitative analysis is performed by considering individual examples. Several ablation studies are performed to provide a more in-depth analysis of the performance of each approach. The research question naturally follows the discussion presented in this section:

How effective are LLM-based approaches at solving data scarce intent detection in the few-shot setting?

1.1 Thesis overview

This thesis begins with a brief discussion on the topic, presenting reasons for investigating and comparing the three LLM-based approaches in intent detection. Section 2 explores previous work on intent detection, each approach, as well as alternative solutions. Section 3 details, how each individual approach is implemented and tested. Section 4 communicates the results of the experiments performed. Section 5 investigates and explores the outcomes of Section 4. Finally, the thesis is wrapped up in Section 6 with adequate discussion of limitations and potential for further research presented. This paper is a bachelor thesis at Leiden University, supervised by I-Fan Lin and Dr. Suzan Verberne from the Leiden Institute of Advanced Computer Science (LIACS). This work was performed using the compute resources from the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University.

2 Background and related work

Intent detection as a standalone text classification task has been receiving increasing attention in recent years, with the publication of three different intent detection datasets. These are

[2]meta-llama/Llama-3.2-3B-Instruct - https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

[3]clinc/oos-eval - https://github.com/clinc/oos-eval

[4]meta-llama/Llama-3.2-3B-Instruct: Instruction-tuned models - https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct#instruction-tuned-models

CLINC150 [LMP+19] and HWU64[5] [LESR19] in 2019, as well as Banking77[6] [CTG+20] in 2020. The creation of these datasets was motivated by the shifting focus towards conversational agents (or, more simply, 'chatbots') made possible by the emergence of transformer models in 2017 [VSP+17], closely followed by the first generative models in 2018 [RN18]. The three aforementioned datasets have established themselves as essential intent detection benchmarks. As explained in Section 1, this thesis focuses on benchmarking intent detection in the few-shot setting due to constraints in available data. Section 2.1 briefly covers, why LLMs are the architecture of choice for intent detection, and 2.2 introduces the LLM that will be used in this paper. Section 2.3 details previous work on intent detection; with the three approaches; and relevant alternative solutions.

2.1 Decoder-only models and LLMs

This paper uses a decoder model – Llama-3.2-3B-Instruct [TLI+23]. A transformer model typically consists of an encoder and a decoder. A decoder model only uses the decoder of the transformer. Decoder models are uni-directional – a sequence of tokens is always processed from left to right. Consequently, a decoder model is auto-regressive – the output of the decoder depends on previous input. Moreover, decoder models use masked self-attention by replacing tokens on the right hand side of the current token being attended to with `[MASK]`. Example decoder-only model architecture is shown in Figure 1[7].

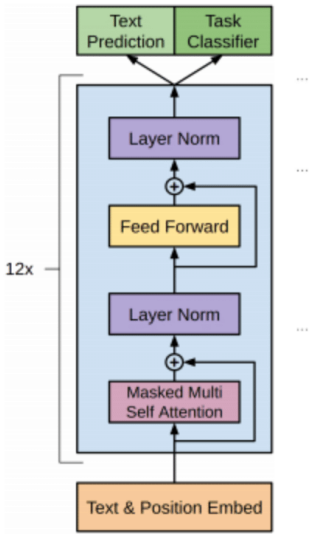

Figure 1: The decoder-only model architecture used by GPT-2 [POV21]. "Masked multi self-attention" refers to multiple self-attention heads with the right side tokens replaced with `[MASK]`.

[5]DeepPavlov/hwu64 - https://huggingface.co/datasets/DeepPavlov/hwu64
[6]Banking77 - https://www.unitxt.ai/en/1.9.0/catalog/catalog.cards.banking77.html
[7]Decoder-Only Architecture used by GPT-2. - https://www.researchgate.net/figure/Decoder-Only-Architecture-used-by-GPT-2_fig1_349521999

Decoder-only models are the backbone of most modern LLMs, particularly the largest models. This is because decoder-only models are much more suitable for pretraining and fine-tuning on very large-scale unlabeled data than encoder-decoder architectures (standard transformer models) [DXWL25]. Moreover, due to their autoregressive nature, they are much more suited for generating text, which makes them desirable in LLMs and, consequently, chatbots, compared to encoder-only architectures such as BERT [DCLT19]. Because of these properties, decoder-only models can be trained on an enormous amount of data, instilling them with unprecedented language capabilities.

2.2 Llama 3.2

The lightweight Llama 3.2 models are derived from the original Llama 3 family of models, which were pretrained on over 15T tokens, collected from publicly available sources [TM24]. Llama-3.2-3B and Llama-3.2-1B are structurally pruned versions of Llama-3.1-8B, pretrained on up to 9T tokens. As Meta has outlined in their blog on Llama 3.2[8], the use of pruning, teacher models, and knowledge distillation, can be used to create smaller improved models that retain a lot of the capabilities of the larger ones. Knowledge distillation uses a larger teacher model to supply more knowledge to a smaller model than it would otherwise be able to learn from the ground up. To achieve knowledge distillation, the output logits of Llama-3.1-8B and Llama-3.1-70B were 'used as token-level targets'. Both lightweight models have been scaled to support context length of 128K tokens. In practice, using a 128K token context window out-of-the-box requires $\sim$ 20GB of VRAM. The 'Instruct' versions of the lightweight models are attained through further fine-tuning to produce a variant that is optimized to follow instructions and complete NLP tasks. However, as a tradeoff, its intrinsic natural language knowledge is less general. As previously stated, this thesis uses the 'Instruct' version of Llama 3.2.

2.3 Related work

Initial work on intent detection using LLMs set pre-trained and/or fine-tuned benchmarks such as ConveRT [HCM+20], SetFit [TRJ+22] and ConvFit [VSC+21]. ConveRT is a small and fast pretrained model geared towards conversational applications that was shown to transfer effectively to the intent detection task, achieving strong results on datasets such as AmazonQA [GKC+19]. Meanwhile, the SetFit framework first applies contrastive fine-tuning to a Sentence Transformer that is then used to generate text embeddings to train a classification head for few-shot learning. Similarly, ConvFit includes two stages of fine-tuning that allows intent detection to be formulated as a semantic similarity task and solved using nearest neighbour retrieval. Other work [ZBY+21] applies self-supervised contrastive pre-training, and then performs fine-tuning using supervised contrastive learning [KTW+21] for few-shot intent detection. Following the seminal paper *Language Models are Few-Shot Learners* [BMR+20], which proposed that LLMs can effectively leverage in-context learning (ICL) via prompting, recent work has profoundly shifted to prompting-style approaches as opposed to pre-training or fine-tuning. These approaches, as previously stated, are much more data efficient, requiring fewer task-specific data for good results.

[8] Llama 3.2: Revolutionizing edge AI and vision with open, customizable models - https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/

To tap into the potential of the *prompting LLM for the label directly* approach, previous work has sought to create a prompting framework that maximizes in-context learning by presenting relevant and useful examples. As described in a survey on in-context learning with retrieved demonstrations [LXL+24], there are four distinct approaches that address this: few-shot ICL for LLM, ICL with Demonstration Retrieval, Off-the-Shelf Demonstration Retriever, and Fine-tuned Demonstration Retriever. Few-shot ICL for LLM seeks to optimize aspects of the prompt, such as number of demonstrations, the demonstration format, or the order of the demonstrations. Meanwhile, ICL with Demonstration Retrieval focuses on optimal retrieval corpora and retrieval objectives such as similarity or diversity. However, Off-the-Shelf Demonstration Retrievers allow this process to be done automatically, mainly focusing on the retriever's type: term-based similarity, sentence-based similarity, or dual encoder. For text classification with many labels, like in CLINC-150, using an off-the-shelf retriever in tandem with modern LLMs has resulted in State-of-the-Art performance in few-shot settings [MRB23]. Finally, Fine-tuned Demonstration Retrievers are trained to retrieve the optimal examples for the relevant task, where the focus is on choosing optimal training data and training objective. A successful proposed framework for this approach is RetICL [SL24], which trains a demonstration retriever using reinforcement learning.

The *training a small transformer encoder on LLM-augmented data* approach also uses prompting and in-context learning. The LLM, following a few demonstrations, is prompted to generate additional training examples. The original + synthetic data is used to train a smaller model than the LLM itself to perform the text classification task. Initial work on this approach proposed the framework ZeroGen [YGL+22], which created a synthetic dataset from the ground up using GPT-2 XL and then trained an LSTM on this dataset. Other work proposes using a decoder model, like GPT, for data augmentation, and a bidirectional encoder, such as BERT, for inference [MHZH22]. The first research on intent detection specifically used GPT-3 to generate more (utterance, intent) pairs, however, for semantically close intents, this data was less helpful [SRL+22]. Building on these findings, pointwise V-information (PVI) was used as a metric to filter out less relevant synthetic examples after performing data augmentation [LPK+23]. Very recently, the framework Generate then Refine [LHV24] proposes the introduction of a Refiner, a fine-tuned model that improves synthetic examples, in the data augmentation pipeline for zero-shot intent detection. Despite being zero-shot, this method has shown to be competitive with existing few-shot alternatives.

Finally, the *extracting LLM sentence embeddings as predictions* approach focuses on improving the quality of the model's sentence embeddings, so they can be extracted to perform NLP tasks. Initial work proposed the framework SimCSE [GYC22], which uses contrastive learning on sentence embeddings both supervised and unsupervised. Building on this framework, DiffCSE [CDL+22], another unsupervised contrastive learning framework, was developed. It learns sentence embeddings based on the difference between the original and edited sentence. Recently, PromptEOL [JHL+24] has shown to outperform the sentence embeddings of contrastive learning based methods. Using clever prompting, the framework forces the LLM to condense the meaning of a sentence into one word – its next predicted token. This method is further strengthened using a mini-framework for sentence embedding alignment, which can be used for finetuning. An alternative to PromptEOL, echo embeddings [SKF+24] are another solution to the approach. Echo embeddings, through repetition of input, seek to address the limitation that the token embeddings of autoregressive models cannot contain information from later input tokens. This allows LLMs to leverage higher

quality embeddings by allowing the model to 'see' the entire sequence before predicting the next token.

3 Methodology and Experimental Setup

In this section, the methodology that I employed in carrying out this research is described. I explain the methodology step-by-step, starting with the simulation of the few-shot setting and seeding, continuing with the technical implementation of each LLM-based approach, and ending with the testing, evaluation and analysis details. I wrote the code in Python, and I carried out all experiments in a Python *venv* virtual environment.

3.1 Dataset for intent detection

As described in the previous sections, I use CLINC150 [LMP+19] to benchmark intent detection for this research. The dataset contains 150 intent classes across 10 different domains. Each intent class has 100 train, 20 validation, and 30 test examples. Moreover, the dataset also contains the 'oos' (Out-Of-Scope) class with 100 train, 100 validation, and 1000 test examples. However, I did not consider OOS detection for the base comparison of the three approaches, so I removed it from the dataset prior to experimentation. A script for removing the 'oos' class and remapping the dataset is available in Appendix Section A. One of the ablation studies covered in Section 4.3 addresses adding the 'oos' class back in (see 4.3.1). The full CLINC150 dataset was imported as a Hugging Face Dataset object. The statistics of CLINC150 are summarized in Table 1.

CLINC150	
domains	10
intents	150
train samples	15,000
validation samples	3,000
test samples	4,500

Table 1: The full statistics of the CLINC150 dataset.

3.1.1 Making CLINC150 few-shot

To simulate data scarcity and to make the intent detection task more challenging for each approach, I used a few-shot setting ($K = 5$). This means that, for this experiment, there are only $K = 5$ utterances (train examples) available to the LLM per intent in CLINC150, constituting a total of $5 \times 150 = 750$ example utterances. A randomly sampled few-shot CLINC150 dictionary is obtained as shown in Algorithm 1.

Algorithm 1 Randomly sampling few-shot CLINC150 dictionary

```
Initialize dataset, intents
Initialize empty sampled_intent_ex dictionary
train_clinc = dataset['train']
intent_to_utterance = defaultdict(list)
for ex in train_clinc do
    intent_idx = ex['intent']
    intent_name = intents[intent_idx]
    intent_to_utterance[intent_name].append(ex['text'])
end for
for intent, examples in intent_to_utterance.items() do
    sampled_intent_ex[intent] = random.sample(examples, 5)
end for
return sampled_intent_ex
```

The dictionary was sufficient for the Data augmentation and Sentence embeddings approaches, while for the Prompting LLM approach I used a few-shot list to make conversion to a Dataset object easier. I did this by adding the following lines to the end of Algorithm 1:

```
few_shot_list = []
for intent, texts in sampled_intent_ex.items():
     for t in texts:
          few_shot_list.append('text':t, 'intent':reverse_map[intent])
```

3.1.2 Seeding and model settings

To ensure the reproducibility of the results of this research, I specified a random seed at the top of each script for all modules that use any form of random seeding. I used three different random seeds during the experiments to increase the reliability of the results and allow for averaging over the three seeds. The seeds I used were: 0, 42, and 1337. I implemented the assignment of a random seed value with the script below:

```
seed_val = 42
np.random.seed(seed_val)
random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed(seed_val)
set_seed(seed_val)
```

For this paper, I largely used Llama-3.2-3B-Instruct out of the box with respect to hyperparameters. For important settings, such as `temperature` or `top_p`, I used the default value. The only hyperparameters that I modified were: `max_new_tokens`, `num_return_sequences`, `do_sample`, and `eos_token_id`.

3.2 Prompting LLM approach

The first distinct approach to performing LLM-based few-shot intent detection is *prompting LLM for the label directly*. The specific implementation of this approach I used for this research was based on the method proposed by Milios et al. (2023) [MRB23]. The paper proposes using an off-the-shelf demonstration retriever to build a prompt containing train examples that have the highest cosine similarity with the test example provided during inference. The intuition here is that some of the in-prompt examples will have the same intent as the test example, so the chances of the LLM generating the correct intent are much higher after being primed with the prompt.

3.2.1 Demonstration retriever

The demonstration retriever I used is the same as the one used in Milios et al. (2023). The specific model is `all-mpnet-base-v2`[9], also called Sentence-BERT (or SBERT), from the SentenceTransformers library. It contains 110M parameters (just like base BERT), and it has been trained on over 1 billion sentence pairs with contrastive learning to create meaningful sentence embeddings. Consequently, it can be used to encode sentences into powerful embeddings. Using the demonstration retriever, I encoded the few-shot CLINC150 train set into a set of normalized 'demonstration' embeddings. Similarly, I encoded the original CLINC150 test set into a set of normalized 'query' embeddings.

3.2.2 Prompt engineering

Having used the demonstration retriever to encode the few-shot train set and the test set as embeddings, we can query the demonstration retriever with test examples ('queries') and it will retrieve the train examples ('demonstrations') with the highest cosine similarity. I built the prompts for the LLM following the same format described in Milios et al. (2023). Algorithm 2 details how the prompts are engineered for the prompting LLM approach.

Algorithm 2 Engineering prompts for prompting LLM approach

```
Initialize demonstrations, queries, demo_embeddings, query_embeddings
Initialize intent_classes, demo_intents, N
Initialize empty list_of_prompts
for j, q_embedding in enumerate(query_embeddings) do
    cosine_similarities = np.dot(demo_embeddings, q_embedding)
    top_n_index = np.argsort(cosine_similarities)[-N:][::-1]
    prompt = ""
    for i in top_n_index do
        prompt += f"Text: {demonstrations[i]}\nClass: {intent_classes[demo_intents[i]]}\n\n"
    end for
    prompt += f"Text: {queries[j]}\nClass: "
    list_of_prompts.append(prompt)
end for
return list_of_prompts
```

[9] sentence-transformers/all-mpnet-base-v2 - https://huggingface.co/sentence-transformers/all-mpnet-base-v2

In Milios et al. (2023) [MRB23], the number of in-prompt train examples used was varied, ranging from 10 up to fully saturated prompts (=number of in-prompt examples that fit in-context). This varied across datasets and LLMs, for example, when using the full 4K context length of the LlaMA-2 models on the GoEmotions dataset, this number was 140 in the paper. However, saturating the prompt of Llama 3.2 is infeasible and would go against the few-shot setting as both the 1B and 3B models have 128K context length out of the box. When comparing the proposed method against baselines, such as ConvFit and SetFit, the number of in-prompt train examples used in the paper was 20. As such, I also used 20 in-prompt train examples for this research ($N = 20$ in Algorithm 2). Since I use the few-shot setting ($K = 5$), the prompt can contain a maximum of 5 examples with the same intent as the test example. The other 15 in-prompt examples will be highly similar utterances corresponding to other intents in CLINC150. It is entirely possible that the 5 examples from the same intent as the test example are not retrieved first in the prompt or that not all 5 are retrieved in the prompt if the test example is dissimilar enough. An excerpt from an example prompt is shown in Table 2.

Text: tell me how to say, 'it is a beautiful morning' in italian Class: translate Text: please suggest a meal from italy to me Class: meal_suggestion Text: how do they say "i love you" in japanese Class: translate Text: tell me how handkerchief is spelled Class: spelling Text: is there a surcharge for using my card in italy while i'm there Class: international_fees Text: what is the surcharge for using my card in italy while i visit Class: international_fees ... Text: how would you say fly in italian Class:

Table 2: Example prompt for the test example *how would you say fly in italian* following the Prompting LLM approach.

3.2.3 Extracting predictions

As discovered during the implementation of the Prompting LLM approach and in Milios et al. (2023) [MRB23], the predicted class generated by the LLM does not always match an intent from

the set of intents in CLINC150. If this is the case, then I encode this class with the tokenizer of the demonstration retriever and query it against an encoded set of all intents in CLINC150. The intent from CLINC150 with the highest cosine similarity is selected as the model's prediction. In doing so, the model is not penalized for generating output that is not an exact match. Examples of generated classes and the corresponding intents in CLINC150 that returned the highest cosine similarity are shown in Table 3.

Generated	Predicted
`transfer_money`	`pay_bill`
`math`	`calculator`
`1_letter_words`	`spelling`
`401k`	`rollover_401k`
`10_minutes`	`cook_time`

Table 3: The model's out-of-the-box generated class often does not match any of the classes in the dataset, but the demonstration retriever can be used to retrieve the class that was most likely predicted.

3.3 Data augmentation approach

The second approach to performing LLM-based few-shot intent detection is *training a small transformer encoder on LLM-augmented data*. The implementation of this approach that I used for this paper was based on the method proposed by Sahu et al. (2022) [SRL+22]. The paper proposes using an LLM to extend the existing few-shot dataset with augmented train examples and then finetuning a transformer encoder on the resulting augmented dataset.

3.3.1 Prompt engineering

Engineering the prompts that are provided to the LLM to create an augmented dataset is relatively straightforward. I adopted the same prompt format as shown in Sahu et al. (2022) [SRL+22]. I used the few-shot CLINC150 dictionary obtained using Algorithm 1 to build prompts, formatted as in Table 4. Algorithm 3 describes the full prompt engineering process for Data augmentation.

The following sentences belong to the same intent restaurant_reviews: Example 1: tell me the reviews for bjs Example 2: how good are the ratings for outback Example 3: so does outback steakhouse have good reviews Example 4: at yakamoto how is their sushi Example 5: does mcdonalds have good reviews Example 6:

Table 4: Example Data augmentation engineered prompt for the intent *restaurant_reviews*.

Algorithm 3 Engineering prompts for the Data augmentation approach

```
Initialize intents, intent_dictionary
Initialize empty list_of_prompts
for i in intents do
    prompt = f"The following sentences belong to the same intent {i}:\n \n"
    for n, example in enumerate(intent_dictionary[i]) do
        prompt += f"Example {n+1}: {ex}\n"
    end for
    prompt += f"Example 6:"
    list_of_prompts.append(prompt)
end for
return list_of_prompts
```

3.3.2 Building an augmented few-shot dataset

I used the prompts created in Section 3.3.1 as input to the LLM, allowing it to complete "Example 6:" with another utterance. There are 150 prompts, corresponding to 150 intents, so `num_return_sequences=10` is used to allow the LLM to complete the same prompt 10 times, generating more augmented examples for the same intent. To ensure that the LLM does not always return the same, most-likely utterance, `do_sample=True` is used. Using `do_sample` ensures some diversity among the generated examples as the LLM's predictions are less deterministic. The generated output is decoded and cleaned. I stored this output in the few-shot CLINC150 dictionary that was made before using Algorithm 1. The resulting augmented dataset contains 15 train examples per intent (5 from original, 10 augmented by LLM), totaling to 2250 train examples.

3.3.3 Finetuning BERT-large

To complete the *training a small transformer encoder on LLM-augmented data* approach, naturally a small transformer encoder is required. `bert-large-uncased`[10] [DCLT18] is the encoder used in Sahu et al. (2022) [SRL+22], which I also used for this paper. It contains ∼330M parameters, making it a larger version of the base BERT model. In Sahu et al. (2022) [SRL+22], the authors design their own classification head for the BERT model to have more control over every step of the finetuning process.

For this paper, I retrieved *BertForSequenceClassification* from Hugging Face. It also contains a classification head with uninitialized weights. While this solution has less control, it is more reliable and easier to implement and deploy. To finetune the model on the augmented dataset, I initialized a Trainer instance with the following hyperparameters modified by TrainingArguments:

```
per_device_train_batch_size=32
per_device_eval_batch_size=64
num_train_epochs=10
learning_rate=2e-5
```

[10] google-bert/bert-large-uncased - https://huggingface.co/google-bert/bert-large-uncased

```
weight_decay=0.01
warmup_ratio=0.1
```

The model is finetuned with Trainer.train() and the predictions are extracted by performing Trainer.predict() on the CLINC150 test set.

3.4 LLM Sentence embeddings approach

The third distinct approach to performing LLM-based few-shot intent detection is *extracting LLM sentence embeddings as predictions*. For this research I used an adapted implementation variant of the method proposed by Jiang et al. (2024) [JHL+24]. This paper proposes the framework PromptEOL, intended to enhance the quality of the embeddings generated by the LLM by introducing a one word limitation.

3.4.1 Prompt engineering

I formatted the prompts according to the PromptEOL framework described in Jiang et al. (2024) [JHL+24], but I adapted them for intent detection since the original paper addresses a semantic textual similarity (STS) task. Algorithm 4 shows the prompt engineering process according to PromptEOL. The motivation to enforce a one word limitation to describe the meaning of a given sentence is based on the idea that the LLM will adhere to the prompt and try to condense as much of the meaning of the sentence as possible in its next predicted word/token. Consequently, this is reflected in the LLM's embeddings, which are of much higher quality after priming with the prompt. I extracted the final hidden state of the last token as the sentence representation.

Algorithm 4 Engineering prompts for the Sentence embeddings approach

```
Initialize few_shot_dict
for intent, examples in few_shot_dict.items() do
    temp_list = []
    for e in examples do
        prompt = f'The task is intent detection. The goal is to identify the purpose or goal behind a user input. The user intent of this sentence: "{e}" means in one word:'
        temp_list.append(prompt)
    end for
    few_shot[intent] = temp_list
end for
return few_shot_dict
```

The task is intent detection. The goal is to identify the purpose or goal behind a user input. The user intent of this sentence: "i would like to know how to say hello in french" means in one word:

Table 5: Example Sentence embeddings engineered prompt for the intent *translate*.

3.4.2 Classification with embeddings

I prompted the LLM with all few-shot examples from CLINC150 with the same prompt format, extracting 750 embeddings, 5 per intent. To adapt the PromptEOL framework to intent detection, I created a centroid from the 5 extracted embeddings of each intent (by performing mean pooling on the 5 embeddings). These centroids are normalized to unit vectors that are in the same direction as the original centroid, but with a magnitude of 1. As a result, there are now 150 unit centroids, representing 150 intents as embeddings. To extract the LLM's predictions on the test set, I performed the same prompt engineering as in Algorithm 4 on the test set, creating prompts that are passed to the LLM to create test set embeddings. These test embeddings are also normalized to unit vectors. I calculated a dot product of each test embedding against all centroids. This is effectively cosine similarity - the unit centroid, which is most similar to the test embedding in its direction, is chosen as the prediction. This process is shown in Algorithm 5 below.

Algorithm 5 Extracting test set predictions from sentence embeddings

```
Initialize centroid_keys, centroid_values as lists
Initialize test_prompts
for test_intent, test_prompt_list in test_prompts.items() do
    for test_prompt in test_prompt_list do
        tokenized_input = llama_tokenizer(test_prompt,
                                          padding=True,
                                          return_tensors="pt").to(device=device)
        test_embedding = llama_model(**tokenized_input,
                                     output_hidden_states=True,
                                     return_dict=True).hidden_states[-1][:, -1, :]
        norm_embedding = torch.nn.functional.normalize(test_embedding, p=2, dim=1)
        similarities = torch.stack(centroid_values) @ norm_embedding.T
        most_sim_idx = torch.argmax(similarities).item()
        predicted_intent = centroid_keys[most_sim_idx]
    end for
end for
return few_shot_dict
```

4 Results

The performance of the three approaches on 5-shot intent detection is covered in this section. Section 4.1 presents the quantitative results of the three experiments by addressing overall accuracy, per-class metrics, and confusion matrices. In Section 4.2, I examine individual misclassified examples to add more nuance to the analysis. Section 4.3 presents the different ablation studies I performed to further explore and understand the performance of the three approaches. In Section 4.3, the inclusion of the 'oos' class and, consequently, OOS detection rate are first considered, followed by one ablation specific to each approach.

4.1 Quantitative analysis

4.1.1 Accuracy

For text classification tasks, such as intent detection, accuracy is the standard metric with respect to performance. I evaluated the three approaches by their accuracy on the test set of CLINC150, which contains 4500 samples. The accuracy of each approach over 3 random seeds is summarized in Table 6. The bottom row displays the average accuracy and standard deviation.

Random Seed	**Prompting LLM**	**Data augmentation**	**Sentence embeddings**
42	0.783	0.852	0.887
0	0.794	0.864	0.878
1337	0.795	0.857	0.886
Average	0.791 ± 0.006	0.858 ± 0.006	0.884 ± 0.005

Table 6: Average accuracy on the test set of CLINC150 over three random seeds for each approach.

As observed in Table 6 above, the *extracting sentence embeddings as predictions* approach ('Sentence embeddings' in the table) consistently outperforms the other two approaches in terms of accuracy. On average, it is 0.026 more accurate than the *finetuning a small transformer encoder on LLM-augmented data* approach ('Data augmentation' in the table) and 0.093 more accurate than the *prompting LLM for the label directly* approach ('Prompting LLM' in the table). The difference in accuracy, especially between Sentence embeddings and Data augmentation, might appear insignificant, but this corresponds to, respectively, ~ 117 and ~ 419 more correctly labelled test samples by the Sentence embeddings approach.

4.1.2 Per-class metrics for the bottom 10 intents

A more complete picture of the three approaches can be formed by examining their performance floor. Tables 7, 8, 9 show the bottom 10 intents sorted by F1 score, precision, and recall respectively. In each table, the intents highlighted in **bold** are in the bottom 10 intents for all three approaches. Meanwhile, intents marked with * are the worst performing intents for a given approach that are not in the bottom 10 for the other two approaches. For example, `user_name` is the worst performing intent for the Prompting LLM approach by F1 score and recall, however, it also appears in the list of the bottom 10 intents for the Sentence embeddings approach, so it is not marked with *. This way it is possible to distinguish intents that are difficult to correctly predict for all approaches, and also specific intents that cause problems for a given approach. In Section B in the Appendix, subsections B.1, B.2, B.3 show the bottom 10 intents by F1 score, precision, and recall respectively for all three seeds.

Bottom 10 intents on average sorted by F1 score					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	F1	Intent	F1	Intent	F1
user_name	0.144	shopping_list *	0.192	**change_ai_name**	0.474
smart_home	0.293	order	0.481	user_name	0.477
change_ai_name	0.402	tire_pressure	0.590	change_user_name *	0.480
w2 *	0.445	**smart_home**	0.618	reminder	0.585
payday	0.446	apr	0.620	todo_list	0.616
what_is_your_name	0.480	translate	0.666	ingredients	0.634
cancel	0.488	**change_ai_name**	0.685	**smart_home**	0.634
bill_balance	0.489	calendar_update	0.699	report_lost_card	0.646
account_blocked	0.493	redeem_rewards	0.714	translate	0.681
calories	0.501	how_busy	0.727	apr	0.682

Table 7: Bottom 10 intents sorted by F1 score for all three approaches averaged over three seeds

Bottom 10 intents on average sorted by precision					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Prec	Intent	Prec	Intent	Prec
user_name *	0.260	shopping_list	0.195	change_ai_name	0.509
what_is_your_name	0.357	calendar_update *	0.680	**change_user_name**	0.512
flip_coin	0.383	how_busy	0.693	shopping_list	0.610
w2	0.412	bill_balance	0.712	recipe *	0.629
next_holiday	0.437	redeem_rewards	0.718	reminder	0.631
cancel	0.447	**change_user_name**	0.723	what_is_your_name	0.633
cook_time	0.505	change_accent	0.732	cancel	0.661
change_ai_name	0.521	smart_home	0.737	damaged_card	0.669
change_user_name	0.541	yes	0.748	todo_list	0.675
smart_home	0.545	find_phone	0.748	distance	0.687

Table 8: Bottom 10 intents sorted by precision for all three approaches averaged over three seeds

Bottom 10 intents on average sorted by recall					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Recall	Intent	Recall	Intent	Recall
user_name	0.100	shopping_list *	0.189	user_name	0.411
smart_home	0.200	order	0.356	**change_ai_name**	0.444
payday *	0.311	apr	0.467	change_user_name *	0.467
change_ai_name	0.333	**smart_home**	0.533	**smart_home**	0.500
calories	0.356	tire_pressure	0.533	ingredients	0.533
account_blocked	0.389	translate	0.589	report_lost_card	0.578
bill_balance	0.422	**change_ai_name**	0.611	translate	0.589
schedule_maintenance	0.433	meeting_schedule	0.667	reminder	0.589
how_busy	0.456	credit_limit	0.667	todo_list	0.600
w2	0.500	confirm_reservation	0.677	text	0.656

Table 9: Bottom 10 intents sorted by recall for all three approaches averaged over three seeds

In Table 7, there are two intents – `smart_home` and `change_ai_name` – that are difficult for all three approaches – the Prompting LLM approach achieves an F1 score of 0.293 and 0.402 respectively, while Data augmentation reaches 0.618 and 0.685, and Sentence embeddings manages 0.634 and 0.474. Notably, the Prompting LLM approach performs the worst on these intents out of the three approaches. Except for 2 intents – `shopping_list` and `order` – all other intents for the Data augmentation approach have an F1 score larger than 0.500. Using the same cutoff, Prompting LLM and Sentence embeddings have, respectively, 9 and 3 intents with F1 score below 0.500. Considering both this and that the bottom 7th intent of the Data augmentation approach has a higher F1 score than all bottom 10 intents of the Sentence embeddings approach, it is interesting that the Sentence embeddings approach noticeably outperforms it. It could be that the Sentence Embeddings approach has a lower floor, but a higher ceiling or that the rest of its intents consistently perform better. The intent `shopping_list` is predicted uniquely poorly by the Data augmentation approach, since it does not appear in the bottom 10 intents for the other two approaches and its F1 score is just 0.192. In comparison, all other 149 intents have an F1 score of at least 0.481. This especially perplexing, since the Prompting LLM and Sentence embeddings approaches achieve F1 scores of 0.842 and 0.748 respectively. Meanwhile, the Prompting LLM approach struggles significantly with the intent `user_name` with an F1 score of just 0.144. It is not marked with * because the Sentence embeddings approach also finds it difficult to predict with an F1 score of 0.477. Remarkably, besides `user_name`, `smart_home` and `change_ai_name`, the rest of the bottom 10 for the Prompting LLM approach are intents that only this approach particularly struggles with. In terms of F1 score, the Sentence Embeddings approach does not have outliers as clear as `user_name` and `shopping_list` for the other two approaches. Instead, the intents `change_ai_name`, `user_name` and `change_user_name` are all similarly challenging, since these are the only intents with an F1 score below .500. Out of these three intents, only `change_user_name` is an intent that the Sentence embeddings approach has problems with uniquely.

In Table 8, in terms of precision, the Prompting LLM and Data augmentation approaches continue to have difficulties with `user_name` and `shopping_list` respectively. Markedly, the three approaches only have one commonly difficult intent – `change_user_name` – with respect to precision. Not only

is it different from the two commonly difficult intents in Table 7, but its precision is also relatively high: 0.541, 0.723, and 0.512 for Prompting LLM, Data augmentation, and Sentence embeddings respectively. It is worth pointing out that the Sentence embeddings approach scores the lowest on this intent out of the three approaches. Outside of `shopping_list`, a noticeable trend emerges for the Data augmentation approach – precision is generally high even for the worst predicted intents. Table 8 shows that the other 149 intents in CLINC150 were predicted with a precision of at least 0.680 by the Data augmentation approach, and at least 142 intents were predicted with a precision of at least 0.748. Similarly, in terms of precision, the Sentence embeddings approach also performs relatively well. All 150 intents have a precision of at least .500 using the Sentence embeddings approach. Additionally, its uniquely most difficult intent in Table 8 – `recipe` – is still predicted with a precision of 0.629. On the other hand, the trend is less clear for the Prompting LLM approach. While its bottom 10 intent precision is generally higher than its bottom 10 intent F1 score, there are intents such as `flip_coin` and `next_holiday` that do not even appear among the bottom 10 intents in Table 7, but have poor precision here. Once again an interesting trend emerges when comparing the Data augmentation and Sentence embeddings approaches – the 3rd worst predicted intent by the Data augmentation approach – `how_busy` – has higher precision (0.693) than all bottom 10 intents of the Sentence embeddings approach.

In Table 9, it can be observed that all approaches have generally worse recall than precision or F1 score. This indicates that, when a given intent is predicted, it is generally predicted correctly (higher precision), but often it is missed when it should have been predicted (lower recall). For the Prompting LLM approach, `user_name` and `smart_home` have especially low recall of 0.100 and 0.200 respectively, which converts to just 3 and 6 correctly predicted test examples for these intents. At the same time, 90% and 80% cases respectively were missed. A uniquely difficult intent to recall for Prompting LLM was `payday` with a score of 0.311. Meanwhile, similar to Table 7, Data augmentation has poor recall for `shopping_list` and `order` with scores of 0.189 and 0.356 respectively. Uniquely, `apr` has also a quite poor recall of 0.467. However, comparing the approaches side by side, all bottom 10 intents for the Prompting LLM and Sentence embeddings approaches have poorer recall than the bottom 10 intents for Data augmentation starting from `meeting_schedule`. While generally the Data augmentation approach has worse recall than precision, it is still quite high for most intents. In fact, for the two intents that are difficult for all approaches – `smart_home` and `change_ai_name` – the Data augmentation approach has the highest recall on both among the three approaches. On the other hand, outside of the intents `user_name`, `change_ai_name`, and `change_user_name` – the same intents that were most difficult for the Sentence embeddings approach in terms of F1 score – no other intent has recall below .500. The intent `user_name` is interesting in particular – it has a recall of just 0.411, but a precision of 0.805.

4.1.3 Confusion matrices

To investigate further, why certain intents are uniquely difficult for some approaches but not others, confusion matrices were created that show the top 20 most confused intents in CLINC150 per approach. These can be found in Subsection C.1 under Section C in the Appendix. It should be noted that these matrices correspond to Seed: 42, but the general trends seen in these matrices remain consistent with the average results over the three seeds. Looking at Figure 2, it becomes apparent, why the Prompting LLM approach finds the intent `user_name` particularly difficult. Whenever the

true intent is user_name, what_is_your_name is overwhelmingly the predicted intent (22 times). Meanwhile, this intent also has noticeably low precision – it is predicted as the correct intent when the true intent is change_ai_name, change_user_name, or what_is_your_name. It appears that the approach is unable to distinguish well enough between these more fine-grained intents. Meanwhile, the difficult intent change_ai_name was predicted as change_user_name 14 times. Interestingly, the generally difficult intent smart_home is often confused with the intents cook_time and flip_coin (6 and 4 times respectively). Finally, the other difficult intent for the Prompting LLM approach - w2 - was confused with taxes 19 times.

Considering Figure 3, it appears that shopping_list was often incorrectly predicted by the Data augmentation approach, when the true intent was order and todo_list (6 and 5 times respectively). At the same time, change_ai_name and change_user_name were often confused - change_user_name was predicted 13 times, when the true intent was change_ai_name, and change_ai_name was predicted 11 times, when the true intent was change_user_name. Just based on this it can be hypothesized that the augmented examples might not be diverse enough to clearly distinguish the two intents. The intent order that the approach also struggled with was predicted as recipe and shopping_list 9 and 6 times respectively.

Shifting to Figure 4, it becomes obvious why the intents change_ai_name and change_user_name are consistently among the three worst performing intents for the Sentence embeddings approach: the intent change_ai_name was confused with change_user_name 7 times, while the intent change_user_name was confused with change_ai_name 12 times. It appears that cosine similarity as a metric to select an appropriate centroid might not be fine-grained enough to correctly distinguish the two intents. It could also be that the few-shot examples that were combined into a centroid for the two respective intents might be highly similar, causing less diverse centroids. Interestingly, both intents – change_user_name and change_ai_name – were confused with make_call 4 and 6 times respectively. The other bottom 3 intent for the Sentence embeddings approach – user_name – was often confused with what_is_your_name: what_is_your_name was predicted 5 times for the true intent user_name, while user_name was predicted 7 times for the true intent what_is_your_name. Among other interesting results, todo_list was confused with reminder 13 times and shopping_list was confused with shopping_list_update 10 times.

4.2 Qualitative analysis

To better understand the differences between the three approaches and their performance, it is important to investigate a few individual examples to gain a better overview. Additionally, this will help with understanding why certain intents are difficult for certain approaches, why certain intents are often confused with others, and why some intents are generally difficult regardless. The individual examples are selected based on the intents seen in Tables 7, 8, and 9.

4.2.1 Prompting LLM for the lable directly examples

	Test example	Generated	Predicted	Correct
1.	what's your designation	what_is_your_role	who_do_you_work_for	user_name
2.	how do you refer to me	what_is_your_name	what_is_your_name	user_name
3.	call my name	call_name	make_call	user_name
4.	what site can i get my w2	1040	taxes	w2
5.	set the ceiling fan to low	ceiling_fan	book_flight	smart_home
6.	lets start calling you allan	change_user_name	change_user_name	change_ai_name
7.	how about i call you sue	what_is_your_name	what_is_your_name	change_ai_name
8.	start the dishwasher	dishwasher	directions	smart_home
9.	can you tell me the date of my last check	last_check	last_maintenance	payday
10.	what's the date on my last pay stub	last_paystub	pay_bill	payday

Table 10: Test examples incorrectly classified by the Prompting LLM approach.

Table 10 shows 10 test examples from CLINC150 that were misclassified by the *prompting LLM for the label directly* approach. As discussed in Section 3.2.3 and shown in Table 3, the intents generated by Llama-3.2-3B-Instruct in response to a prompt do not always match existing intents in CLINC150. To better understand the misclassified examples, the model's generated output is included as the column 'Generated', while 'Predicted' is the intent produced by the demonstration retriever if the intent in 'Generated' does not match any of the intents in CLINC150.

Test example 1 in Table 10 is an interesting case of a fairly ambiguous utterance as arriving at the correct intent – `user_name` – is only possible by determining the correct meaning of the word 'designation'. In this case, the demonstration retriever associated the word with its use in professional contexts, when referring to assignments or appointments. As a result, the first 4 in-prompt examples retrieved in response to test example 1 were for the intent `who_do_you_work_for`. However, the intended meaning was 'a special, distinguishing name'. Interestingly, the demonstration retriever also retrieved more in-prompt examples earlier in the full prompt for the intent `what_is_your_name` than `user_name`. This was most likely due to the framing of the sentence as a question. Another interesting case is test example 3, where the model failed to generate an existing intent, and the demonstration retriever found `make_call` to be the intent predicted, most likely on the basis of the word 'call', which moved the prediction away from any of the intents with the word 'name' in them. Test example 4 shows what was a general trend with Llama-3.2-3B-Instruct – it would generate other typical tax forms, like `1040` and `1099`, instead of `w2`. Consequently, the demonstration retriever would return the more general intent `taxes`. Meanwhile, examples 6 and 7 highlight the difficulties this approach has with helping the model understand and correctly interpret grammatical persons as it fails to recognize that 'you' is directed towards the AI in `change_ai_name`. Finally, examples 9 and 10 once again show how the model generating non-existent intents causes the demonstration retriever to find the incorrect intent because it lacks the context of the test example, for example, `last_check` is associated with a different meaning of 'check', so `last_maintenance` is predicted.

4.2.2 Finetuning a small transformer encoder on LLM-augmented data examples

	Test example	Predicted	Correct
1.	i need milk on my shopping list	shopping_list	shopping_list_update
2.	iterate over the items on my grocery list	order	shopping_list
3.	do i have to buy milk	pay_bill	shopping_list
4.	"alexa, buy a new television"	smart_home	order
5.	can you order me some nacho chips	recipe	order
6.	get me an order of creatine powder	ingredient_substitution	order
7.	can someone look at my check engine light that's on	smart_home	schedule_maintenance
8.	how is the temperature at ac	weather	smart_home
9.	my car battery is dead what do i do	tire_pressure	jump_start
10.	recite the items i'm planning to purchase	order	shopping_list

Table 11: Test examples incorrectly classified by the Data augmentation approach.

Table 11 shows 10 test examples from CLINC150 that were misclassified by the *finetuning a small transformer encoder on LLM-augmented data* approach. The findings of these test examples are contextualized by considering the augmented train examples for the intents that the approach particularly struggled with. The two intents this approach struggled with in particular were `shopping_list` and `order`. Table 12 below shows the few-shot original train examples (in **bold**) and augmented examples (in *italics*) for both intents.

Looking at the few-shot examples for the intent `order`, it becomes clear why it is often confused with `shopping_list` in the Data augmentation approach – 'shopping list' is overrepresented in the original examples. The model notices this pattern and reproduces it when generating the augmented examples. Consequently, 9 out 10 generated examples for the intent `order` contain 'shopping list'. Meanwhile, the generations for `shopping_list` are too similar and uniform - 4 generations start with the phrase 'does my shopping list ...' and only 1 generation does not directly mention 'shopping list'. This causes the transformer encoder – `bert-large-uncased` – to learn shallow features during finetuning that are less effective and not representative of the two intents. This would explain, why, for test example 1 in Table 11, the predicted intent is `shopping_list` – BERT has learnt the incorrect feature that a phrase containing 'shopping list' belongs to the `shopping_list` intent. On the other hand, by overrepresenting the phrase 'shopping list' and underrepresenting the word 'order' in the train examples for the intent `order`, BERT misses fairly obvious utterances corresponding to the intent `order` that contain the word 'order', such as in examples 5 and 6 in Table 11, opting to predict based on 'nacho chips' and 'creatine powder' respectively. Simultaneously, BERT ends up predicting the intent `order` more often for utterances that correspond to `shopping_list` as in examples 2 and 10.

order	shopping_list
please get everything on my shopping list	**what does my shopping list entail**
i want to make a purchase	**does my shopping list have tomato written on it**
i'm ready to put in the order for everything on my shopping list	**can you show me my shopping list**
get my entire shopping list ordered	**do you want peanut butter for your lunches this week**
buy everything on my list	**what items are on my shopping list**
finalize my order for all the items on my shopping list	*do you want peanut butter for lunch*
complete my shopping list	*do you have a shopping list*
complete my shopping list	*does my shopping list have any leftovers*
can you finalize my shopping list	*does my shopping list include apples*
complete the purchase for all items on my shopping list	*does my shopping list include eggs*
i'd like to finalize the order for all of my items	*does your shopping list have eggs on it*
finalize my shopping list	*does my shopping list contain milk*
complete my shopping list	*can I see my shopping list*
i need you to complete my shopping list	*can you help me make a shopping list*
complete my shopping list	*does your shopping list include eggs*

Table 12: Original and augmented train set examples for the two intents `order` and `shopping_list`.

4.2.3 Extracting sentence embeddings as predictions examples

	Test example	Predicted	Correct
1.	how do you cooked eggs	cook_time	recipe
2.	what is needed to cook lasagna	recipe	ingredients_list
3.	bob is my name now	change_ai_name	change_user_name
4.	change my name to bob	change_ai_name	change_user_name
5.	call my name	change_user_name	user_name
6.	you know me by what right now	maybe	user_name
7.	the temperature at ac is what	weather	smart_home
8.	let me know when it's been 5 minutes	timer	reminder
9.	lets start calling you allan	make_call	change_ai_name
10.	what do i take home	shopping_list	income

Table 13: Test examples incorrectly classified by the Sentence embeddings approach.

Table 13 shows 10 test examples from CLINC150 that were misclassified by the *extracting sentence embeddings as predictions* approach. Given that the Sentence embeddings approach performs a dot product between a transposed test embedding unit vector and a stack of 150 centroid unit vectors

(corresponding to 150 intents), it is possible to also extract the cosine similarity values. For each misclassified test example, the top 5 highest cosine similarities and their corresponding intents are extracted to better understand, how close or how far off a prediction was.

For the Sentence embeddings approach, `recipe` is an intent that the approach particularly struggles with in terms of precision. Examples 1 and 2 in Table 13 show how `recipe` was missed or wrongly predicted as the true class respectively. The predicted intent `cook_time` in example 1 can indicate one of a few things: 1) the Llama-3.2-3B-Instruct still has noticeably limited Natural Language Understanding (NLU), embedding the sentence's meaning as semantically closer to `cook_time` than `recipe`; 2) the example's embeddings overemphasized the word 'cooked', leading to the test example having higher cosine similarity with `cook_time` than `recipe`; 3) the test example is grammatically incorrect, worsening the resulting embeddings. I am inclined to believe that the example's embedding was predisposed towards `cook_time` due to the inclusion of 'cooked', especially given that both intents had very high cosine similarity with the test example's embedding – 0.921 and 0.913; `cook_time` was 0.008 more similar. Example 2 is an interesting 'miss' by the Sentence embeddings approach as the two intents `recipe` and `ingredients_list` are only separated after 4 decimal places (0.9659 and 0.9657). Examples 3 and 4 show that the embeddings of the Sentence embeddings approach are not fine-grained enough to correctly use grammatical persons in order to distinguish between the two intents `change_ai_name` and `change_user_name`. Example 5 is another case, where the two intents had a very small difference in cosine similarity, only being separable after 4 decimal places (0.9021, 0.9020). Finally, an interesting outcome was example 10, where `shopping_list` had a cosine similarity of 0.887 with the test example embedding, while the correct intent – `income` – was not even among the top 5 most cosine similar intents for this test example.

4.3 Ablation studies

This section presents several ablations that I performed to further explore what contributes to the performance of each approach and whether my implementation of each approach is effective. Section 4.3.1 performs the same experiment from Section 4, but also includes the 'oos' class to investigate, how the inclusion of the class affects the performance of each approach and how good each approach is at OOS detection. Sections 4.3.2, 4.3.3, 4.3.4 present approach-specific ablations for Prompting LLM, Data augmentation, and Sentence embeddings approaches respectively. In each section I modify a component of the original implementation approach and then compare against the results of the original experiment.

4.3.1 OOS detection

For the first ablation, it would be insightful to investigate, how the performance of the three approaches changes when the 'oos' class is included back into the CLINC150 dataset. To do this, I simply took the full dataset out-of-the-box and did not perform any preprocessing on it. I then reran the experiment for all three approaches. The results of this ablation are covered in Table 14 below. It should be noted that this ablation was carried out on just random seed 42.

To preface the results seen in Table 14, as already mentioned in Section 3.1, it is worth restating that the 'oos' class, despite having the same amount of train examples as the other classes in

	w/o OOS	with OOS	OOS detection
Prompting LLM	0.783	0.642	0.002
Data Augmentation	0.852	0.711	0.011
Sentence embeddings	**0.887**	**0.759**	**0.184**

Table 14: OOS detection rate. Columns 'w/o OOS' and 'with OOS' show the overall accuracy on the CLINC150 test set when 'oos' is excluded and when 'oos' is included respectively. The 'OOS detection' column shows the accuracy of the three approaches on the 'oos' class.

the CLINC150 dataset (100), has 1000 test examples in CLINC150, compared to just 30 for the other classes. This means that the OOS detection rate can still be very poor, even if the raw number of correctly predicted 'oos' test examples is more than some of the other classes. In Table 14, the Sentence embeddings approach is clearly superior to the other two approaches when 'oos' is included. Not only is its overall accuracy 0.117 and 0.048 higher than the accuracy of the Prompting LLM and Data augmentation approaches respectively, but also its OOS detection rate is significantly better: the Sentence embeddings approach has an OOS detection rate of 0.184, which is 0.182 and 0.173 higher than the OOS detection rate of the Prompting LLM and Data augmentation approaches respectively. The OOS detection rate can be directly translated to raw numbers for all three approaches: the Sentence embeddings approach predicted 'oos' correctly 184 times, followed by the Data augmentation approach with 11 times, and the Prompting LLM approach with just 2 times. An OOS detection rate of 0.184 still indicates that, most of the time, the Sentence embeddings approach missed the 'oos' examples, but this is still a remarkable result, given the notorious difficulty of the OOS detection task and when comparing with the other two approaches. Its worth reminding that the approach was able to correctly predict 184 'oos' test examples using a centroid made up of just 5 'oos' train examples.

In Section B in the Appendix, subsections B.1, B.2, B.3 show the performance of the three approaches for the bottom 10 intents in terms of F1 score, precision, and recall respectively. Unsurprisingly, the 'oos' class was the worst predicted intent in terms of F1 score and recall for all approaches. However, in terms of precision, when an approach did predict 'oos' as the correct class, it was generally correct, since the 'oos' class did not appear among the bottom 10 intents for any of the approaches in terms of precision. Of course, for the Prompting LLM and Data augmentation approaches, this was due to the 'oos' class barely being predicted at all. For the Prompting LLM approach, it can be argued that the 5-shot setting severely inhibits the expressive capabilities of the demonstration retriever, since the 5 randomly sampled OOS examples that it can use cannot be representative enough of the rest of the class to be that useful. This is because the OOS examples can include a wide variety of sentences covering a diverse range of topics that do not exactly fit any of the other intents in CLINC150, but are somewhat related or adjacent. Consequently, the demonstration retriever would often retrieve examples from other classes as the first examples in the prompt before it retrieved any of the examples from the 'oos' class. Even worse, it could build an entire prompt without including an example from the 'oos' class. After all, the demonstration retriever uses cosine similarity, and, due to the diversity of the 'oos' class, it is easy to see how

it could retrieve examples from other intents over 'oos', since two 'oos' examples can be quite dissimilar with respect to cosine similarity. For the Data augmentation approach, the LLM most likely emulated the data distribution of the 5 'oos' examples it saw in the prompt during in-context learning. Consequently, its generated examples are more representative of the 5-shot subset of the 'oos' class than its entirety. These 5 randomly sampled 'oos' examples do not necessarily capture the same data distribution of the whole class.

In section C in the Appendix, subsection C.2 shows the confusion matrices of the three approaches when the 'oos' class is included in CLINC150. As seen in Figures 5, 6, 7 in the Appendix, the 'oos' class strongly distorts the confusion matrix as a lot of intents in CLINC150 were confused with 'oos' at one point or another, especially for the Prompting LLM and Data augmentation approaches. As particular outliers, the Prompting LLM approach confused 'oos' with 'car_rental' 49 times, while the Data augmentation approach confused 'oos' with 'travel_suggestion' 37 times and, interestingly, the Sentence embeddings approach confused 'oos' most often with 'definition', 'what_can_i_ask_you' and 'order' (85, 53, and 51 times respectively). It appears that for the Prompting LLM approach and the Data augmentation approach, the confusion with 'oos' is more uniform across intents, while the Sentence embeddings approach particularly confuses 'oos' with a few classes. Perhaps this pattern is visible due to the former two approaches largely guessing.

4.3.2 Prompting LLM for the label directly: random demonstrations

In Section 3.2, I described an implementation of the Prompting LLM approach that used an Off-the-Shelf demonstration retriever to fill the prompt by finding examples that, based on cosine similarity, are most relevant to the test example. This implementation of the approach almost guaranteed that few-shot train examples corresponding to the intent of the test example would be retrieved first in the prompt. Of course, with semantically close intents and examples, it was still possible to retrieve irrelevant few-shot train examples first. As we saw, this was especially true in Section 4.3.1, where the demonstration retriever was severely limited by the few-shot setting, unable to retrieve OOS examples that would allow the LLM to predict 'oos'. I decided that it would be helpful to investigate, whether the demonstration retriever is useful at all with helping the LLM make correct predictions. To do this, instead of filling the prompt with the demonstration retriever's provided examples, I filled the prompt using random sampling for each test example. In the case that the output of the LLM did not match any of the intents in CLINC150, I still used the demonstration retriever to retriever the most likely predicted intent. This ablation allowed me to investigate, whether using a demonstration retriever based on cosine similarity to fill the prompt is better than simply random prompt generation. The results are summarized in Table 15.

Random Seed	Demonstration retriever	Random demonstrations
42	0.783	0.544
0	0.794	0.548
1337	0.795	0.539
Average	0.791 ± 0.006	0.544 ± 0.005

Table 15: Average accuracy on the test set of CLINC150 over three random seeds for the original Prompting LLM approach (using the demonstration retriever) compared to the ablation (using random demonstrations).

As seen in Table 15, the demonstration retriever plays a significant role in making the Prompting LLM approach viable in terms of performance. Using randomly generated prompts, the Prompting LLM approach achieves an average accuracy of just 0.544. That is 0.247 less accurate than when using the demonstration retriever. It appears that when using the demonstration retriever approach there is some guarantee that most or all few-shot examples corresponding to the intent of the test example will be included in the prompt. Consequently, the LLM is able to use this over-representation of examples corresponding to the test intent to increase the possibility that it predicts the correct intent. Moreover, these provided examples are reliably useful enough that the LLM can process and use them during in-context learning. With random demonstrations, there is no such guarantee, and the in-prompt examples could be the least informative for the given test (utterance, intent) pair, leading to the LLM largely guessing based on what it saw in the prompt during in-context learning.

4.3.3 Finetuning a small transformer encoder on LLM-augmented data: allowing Llama to freely generate

In Section 3.3, I covered an implementation of the Data augmentation approach, in which I prompt the LLM with the prompt as seen in Table 4, and let the LLM generate 10 different return sequences, each of which completes "Example 6:". This made generating and extracting augmented examples very easy. However, I needed to use `do_sample=True` to ensure that there is some variety among the generated examples, such that it does not simply generate the same example 10 times. An alternative way to use this prompt to generate augmented examples would be to set the number of return sequences back to 1, but allow the LLM to generate freely, continuing the prompt with examples 7, 8, 9, ..., 15.

The motivation behind implementing the approach this way is as follows: when the model keeps generating examples, it will use its own output tokens as input to generate new ones. Each example generated by the LLM will essentially be primed by the model's own previously generated tokens, which will be more diverse than using the same tokens (that encoded the prompt) to complete "Example 6:" 10 times. The hope is that this diversity in tokens will lead to more diverse augmented examples, since the model would naturally respond differently to different tokens. I used a near-identical prompt as in Table 4, however, during testing I noticed that the LLM found it difficult to follow the prompt format, often wasting most of its tokens explaining the prompt or generating examples that do not adhere to the prompt format. To ensure that the LLM closely follows the

prompt format, the following was appended at the start of the prompt: "Strictly follow the prompt format and continue the list with 10 additional examples. Do not explain the prompt!\n\n".

The output of this version of the data augmentation approach was processed with regular expressions to extract the augmented examples for the augmented dataset. It should be noted that, despite the additional prompt engineering performed for this ablation, the LLM still failed to follow the prompt format in some rare instances. Consequently, the regular expression used to extract 10 augmented utterances for each intent would fail to do so occasionally. As mentioned in Section 3.3.2, the augmented few-shot dataset should total to 2250 train examples. Due to the issues mentioned above, the augmented few-shot dataset of the ablated version of the Data Augmentation approach totaled to 2238, 2233, and 2245 train examples for the seeds 42, 0, and 1337 respectively. Once again, BERT-large was finetuned on this augmented dataset and tested on CLINC150's test set. The results of this ablation are summarized in Table 16 below.

Random Seed	**10 return sequences**	**Generate freely**
42	0.852	0.852
0	0.864	0.863
1337	0.857	0.859
Average	0.858 ± 0.006	0.858 ± 0.006

Table 16: Average accuracy on the test set of CLINC150 over three random seeds for the original Data augmentation approach (generating 10 return sequences) compared to the ablation (using free generation).

Looking at Table 16, it appears that the the original and ablated version of the Data augmentation approach perform equally well in terms of accuracy. Both have the same accuracy for random seed 42 and on average overall. For seeds 0 and 1337 the two have very small differences in accuracy. These results suggest that the performance of this approach is most likely bottlenecked by BERT-large, which, seemingly regardless of the underlying augmented dataset it has been finetuned on, performs about the same. Clearly, accuracy alone is not enough to separate the original from the ablation.

To better compare the original Data augmentation approach against its ablation, it is worth considering the augmented datasets themselves. Building an augmented dataset in two different ways, which one generates higher quality augmented examples? In the context of train examples, a measure of quality is the diversity of these examples. To measure diversity, I used the Distinct-N metric, proposed by Jiwei Li et al. (2016) [LGB+16]. The Distinct-N metric can be divided into the distinct-1 and distinct-2 metrics, which calculate the number of distinct unigrams and distinct bigrams in sentences respectively. As stated in Jiwei Li et al. (2016) [LGB+16], the resulting value is scaled by the total number of generated tokens to avoid favoring long sentences. We introduce a modification to the method for calculating the distinct-1 and distinct-2 metrics by concatenating all augmented examples for a given intent into one large string. The metrics are then calculated on this string. This is done to penalize highly similar independent augmented examples, which, on their own, would get a high degree of diversity. Table 17 below shows the distinct-1 metric averaged over all 150 intents for the two versions of the Data Augmentation approach across the

three random seeds. The 'Real data' column refers to a 10-shot randomly sampled subset from the original CLINC150 dataset for all 150 intents. This column is included as a benchmark to compare against. Following Table 17, Table 18 shows the distinct-2 metric averaged over all 150 intents.

Random Seed	**10 return sequences**	**Generate freely**	**Real data**
42	0.337	0.384	0.480
0	0.326	0.375	0.470
1337	0.328	0.378	0.475
Average	0.330 ± 0.006	0.379 ± 0.005	0.475 ± 0.005

Table 17: Average overall distinct-1 metric for the datasets generated by both approaches and the original CLINC150 dataset. For CLINC150, 10 examples were randomly sampled per intent.

Random Seed	**10 return sequences**	**Generate freely**	**Real data**
42	0.545	0.611	0.770
0	0.529	0.592	0.764
1337	0.532	0.602	0.766
Average	0.535 ± 0.009	0.602 ± 0.01	0.767 ± 0.003

Table 18: Average overall distinct-2 metric for the datasets generated by both approaches and the original CLINC150 dataset. For CLINC150, 10 examples were randomly sampled per intent.

Looking at both tables, it comes as no surprise that the original CLINC150 dataset is more diverse by a significant margin, compared to the other two approaches. Using distinct-1, it is, on average, 0.145 and 0.096 more diverse than the original augmented and the ablation augmented datasets respectively. Using distinct-2, the gap in diversity grows: on average, it is 0.232 and 0.165 more diverse than the original and ablation respectively. Notably, however, the ablation dataset is more diverse than the original approach to Data augmentation. Using distinct-1, the ablation augmented dataset is 0.049 more diverse than the original. Meanwhile, using distinct-2, the difference in diversity increases to 0.067. Clearly, the augmented dataset created using the ablation is more diverse than the original augmented one. This suggests one of two things: 1) BERT-large is unable to take advantage of the more diverse and higher quality examples generated with the ablated approach; 2) While the examples generated by the ablated Data augmentation approach are more diverse, they fail to emulate the data distribution of the original CLINC150 dataset accurately enough. To test this, I ran a small experiment, where I randomly sampled a 15-shot dataset directly from CLINC150, which, again, totaled to 2250 train examples that were used to finetune BERT-large. The results of this experiment are shown in Table 19 below.

Random Seed	15-shot
42	0.910
0	0.902
1337	0.911
Average	0.908 ± 0.005

Table 19: Average accuracy on the test set of CLINC150 over three random seeds attained by BERT-large after finetuning on a randomly sampled 15-shot subset of CLINC150.

Looking at Table 19, it appears that the Data augmentation approach is not limited by BERT-large, since BERT-large was noticeably more accurate after being finetuned on a 15-shot subset of CLINC150 than any of the augmented datasets. Therefore, the ablation augmented dataset, while more diverse than the original augmented dataset, likely either is still not diverse enough or fails to emulate the distribution of CLINC150 accurately enough, and, consequently, does not lead to more accurate predictions.

4.3.4 Extracting sentence embeddings as predictions: dot product

In Section 3.4, I covered an implementation of the Sentence embeddings approach that normalizes both the centroids and test embeddings into unit vectors, so that a dot product between a test embedding and a stack of centroids would return cosine similarities. The centroid unit vector that had the highest cosine similarity with the unit vector representing the test embedding would be selected as the predicted intent. When using cosine similarity as the metric for making predictions, only the direction of a vector is important. However, it is also possible to take both the direction and magnitude of a given vector into account when making a prediction. This can be done by not performing any normalization on the test embeddings and centroids before calculating the dot product. This ablation is performed to investigate, whether considering both the magnitude and direction of a vector improves the performance of the Sentence embeddings approach as opposed to just considering direction. The results of this ablation are summarized in Table 20 below.

Random Seed	Cosine similarity	Dot product
42	0.887	0.846
0	0.878	0.845
1337	0.886	0.842
Average	0.884 ± 0.005	0.844 ± 0.002

Table 20: Average accuracy on the test set of CLINC150 over three random seeds for the original Sentence embeddings approach (using cosine similarity) and the ablation (using dot product on non-normalized embeddings).

As observed in Table 20, considering both magnitude and direction between a test embedding and a stack of centroids leads to worse performance on the CLINC150 test set in terms of accuracy. Using the dot product, the average accuracy is 0.844. On average, using just direction for predictions results

in 0.04 higher accuracy than using both direction and magnitude. This difference is remarkable enough that if I were to use the ablated version of the Sentence embeddings approach for comparison against the other approaches, it would be outperformed by the Data augmentation approach. It can be speculated that more often vectors with larger magnitude are 'outcompeting' vectors that are otherwise better aligned in terms of direction, leading to more incorrect predictions. For example, some key words encoded in the test embedding might trigger a high dot product with an intent that typically includes the key word, but might not be the correct one in this scenario. This ablation tells us that cosine similarity is a much more accurate metric for predicting intents in CLINC150 than the dot product.

5 Discussion

In this section, I discuss and further interpret the experimental results presented in Section 4, as well as address summarize the limitations that were encountered during this research.

5.1 Interpretation of results

As seen in Section 4.1, the best performing approach to few-shot intent detection on the CLINC150 dataset was *extracting LLM sentence embeddings as predictions*, closely followed by *finetuning a small transformer encoder on LLM-augmented data*. Both approaches noticeably eclipsed the *prompting LLM for the label directly* approach. While the ablation of the Prompting LLM approach in Section 4.3.2 confirmed that the off-the-shelf demonstration retriever used for this approach was incredibly useful, achieving a significantly higher accuracy than just randomly filled prompts, it was still likely the main limiting factor, why the approach underperformed in comparison. As seen with OOS detection in Section 4.3.1, the demonstration retriever was particularly limited by the 5-shot setting, since it was very unlikely that 5 randomly sampled 'oos' examples, when included in the prompt, could be representative and useful enough for predicting 1000 'oos' test examples. Moreover, most of the time, some or all 5 'oos' examples were excluded from the prompt because they were not similar enough to the test 'oos' examples. Furthermore, for semantically close intents, such as `user_name`, `what_is_your_name`, `change_ai_name`, and `change_user_name`, the demonstration retriever would often include examples from each of these intents, potentially misleading the LLM. While the prompt is clearly ordered from most to least similar based on cosine similarity, this relationship, especially after tokenizing the entire prompt, might not be obvious to a unidirectional decoder model. After all, the Prompting LLM approach is the only one of the three that directly uses the LLM's in-context learning capabilities to make predictions. This can be a drawback of using the Prompting LLM approach on a relatively small LLM such as Llama-3.2-3B-Instruct: in response to the prompt, the model generates output tokens during inference, which, after decoding, sometimes do not match any of the intents in CLINC150. Using the demonstration retriever to retrieve the intent the LLM was most likely predicting can be extremely helpful, but, as already discussed in Section 4.2, can also lead to misclassified test examples (such as examples 3, 5, and 9 in Table 10 in Section 4.2.1) since the intent retrieval is removed from the context of the prompt and test example.

It is worth pointing out that the Prompting LLM and Sentence embeddings approach are somewhat similar. Both calculate cosine similarity between vector embeddings of the original sentences. However, the expressive power of the embeddings of the demonstration retriever is unarguably smaller than that of the LLM: the demonstration retriever ("all-mpnet-base-v2") maps input to a 768 dimensional vector space, while the LLM ("Llama-3.2-3B-Instruct") maps input to a 3072 dimensional vector space. Additionally, a 5-shot centroid is most likely a better representative of its corresponding intent than five separate embeddings, although it is not possible to use it with in-context learning, since decoding it would likely produce word salad. Regardless, the effectiveness of the Sentence embeddings approach was highly impressive given the 5-shot setting, especially since it appeared to showcase the true NLU capabilities of the LLM by using its vector space directly. Moreover, it was arguably the most straightforward approach to implement and test in practice. On the other hand, in some aspects the approach had a lower performance floor than the Data augmentation approach as seen in Section 4.1.2 on the per-class metrics for the bottom 10 intents. At the same time, this indicates a very high performance ceiling, since it outperformed the latter.

The Data augmentation approach, while less accurate than the Sentence embeddings approach, provided unique insight through its ablation study in Section 4.3.3 on which factors are most significant for this approach. As it turned out, an increase in the diversity of the augmented dataset did not necessarily lead to measurable gains in accuracy. Of course, going from an augmented dataset to an actual subset of CLINC150 both led to more diversity and better performance as this subset was more representative of the type of examples in CLINC150. It can be speculated, whether using a larger LLM could match or outperform a subset of CLINC150 in diversity and whether this changes the performance of the approach. I leave this for future research. What should be noted is that BERT-large is powerful enough in NLU that the original (less diverse) augmentation method using Llama did not negatively impact performance. In fact, BERT-large outperforms base Llama-3.2-3B on the SQuAD 1.1 benchmark: its EM (Exact Match) score is 84.3[11] vs. Llama's score of 67.7[12], suggesting that a larger LLM is key for this approach. The paper on which my implementation of the Data augmentation approach is based on used GPT-3 to build the augmented dataset. GPT-3 has 175B parameters compared to just 3B for Llama 3.2. For a larger LLM like base Meta-Llama-3-70B[13] [TM24], the EM score on SQuAD 1.1 is 85.6[14], further suggesting that simply using an LLM with more parameters is important for data augmentation. A major drawback of the Data augmentation approach, however, was seen in OOS detection as the additional augmented 'oos' class examples did not help at all: just 11 'oos' test examples were classified correctly after BERT-large was finetuned on 15 'oos' train examples (of which 10 were augmented).

[11] google-bert/bert-large-uncased: Evaluation results - https://huggingface.co/google-bert/bert-large-uncased#evaluation-results

[12] meta-llama/Llama-3.2-3B-Instruct: Base Pretrained Models - https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct#base-pretrained-models

[13] meta-llama/Meta-Llama-3-70B - https://huggingface.co/meta-llama/Meta-Llama-3-70B

[14] meta-llama/Meta-Llama-3-70B: Base Pretrained Models - https://huggingface.co/meta-llama/Meta-Llama-3-70B#base-pretrained-models

5.2 Limitations

While this research is a step in the right direction with respect to robustness and generalisation of the results, there is still room for improvement. In carrying out these experiments, I used 3 different random seeds, which allowed me to present average results. To increase generality and robustness, it would be important to perform the same experiments over 5 or even 10 seeds. On the other hand, standard deviation was fairly small – differences between seeds started at 3 decimal places.

Another limitation of this research is the LLM used. It is difficult to claim, whether the same general conclusions hold true for larger LLMs. After all, in the context of LLMs, a 3B model is tiny. It may be the case that an LLM's in-context learning, text generation, and embedding quality do not scale equally for larger models. Perhaps a larger model's in-context learning capabilities catch up with its text generation and embedding quality.

A final limitation of this research is that its coverage of the three approaches is not exhaustive. As covered in Section 2 on Related work, there are several methods of implementing each approach, for example, one can finetune a demonstration retriever, instead of using it off-the-shelf for the Prompting LLM approach, or one can use metrics to filter out unhelpful augmented examples during the Data augmentation approach, or one can use echo embeddings for the Sentence embeddings approach. As such, it is possible that using a different method for the Prompting LLM or Data augmentation approaches could lead to different conclusions about the comparison between the approaches.

6 Conclusions

To conclude, the goal of my research was to determine the optimal approach to solving data scarce intent detection in the few-shot setting by comparing and analyzing the performance of three emerging paradigms: *prompting LLM for the label directly*, *finetuning a small transformer encoder on LLM-augmented data*, and *extracting LLM sentence embeddings as predictions*. To do this, I implemented a method representing each paradigm: an Off-the-Shelf Demonstration Retriever was used to fill prompts for the Prompting LLM approach, BERT-large was finetuned on a Llama-augmented 15-shot dataset for the Data augmentation approach, and, using cosine similarity, test embedding vectors were compared against a set of vector embedding centroids representing each intent in CLINC150 for the Sentence embeddings approach. The performance of each approach was evaluated quantitatively based on overall accuracy, by considering per-class metrics for the bottom 10 intents of each approach, and by examining the confusion matrices. Following that, qualitative analysis was carried out by investigating unique misclassified examples to gain a deeper understanding, where each approach succeeds or fails. To build a more complete picture, several ablation studies were carried out. Notably, the 'oos' class was added back into CLINC150 during one of the ablations to test the performance of each approach on OOS detection. The other ablations helped gain a better understanding of what factors contribute to the performance of each approach and what is their effect: for the Prompting LLM approach, an ablation using randomly sampled prompts helped quantify the importance of the Demonstration Retriever; for the Data augmentation

approach, an ablation that lead to a more diverse augmented dataset and comparisons against real data helped to understand, which component was bottlenecking the approach; for the Sentence embeddings approach, the ablation helped stress the importance of using the correct metric in predictions.

As discussed in Sections 4.1, 4.2, and 5.1, *extracting LLM sentence embeddings as predictions* was clearly the superior approach to solving few-shot intent detection on the CLINC150 dataset. Given a fairly simple implementation, it performed extremely well in the experiments carried out. It should be remarked that all three approaches were still very impressive in their ability to solve intent detection given the 5-shot setting: having seen just 750 train examples, each approach achieved an overall accuracy of at least 0.791 on a test set consisting of 4500 examples. However, as discussed in Section 5.2, the generalization of the results is limited: it is difficult to claim that the same results would hold true if components of my original experiment were modified.

The room for future research stems from the limitations discussed in Section 5.2. The experiment could be reran for more than 3 seeds to determine if the results still hold true. Furthermore, the experiment could be performed on a larger LLM to investigate, whether the conclusions drawn from my experiment are generalizable or only hold true for tiny LLMs such as Llama-3.2-3B-Instruct. Finally, this experiment could better represent each approach by being more exhaustive: including more different implementation methods of each approach in the comparison. This would create a much more detailed and general picture regarding the State-of-the-Art to solving data scarce intent detection in the few-shot setting.

References

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[CDL+22] Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljačić, Shang-Wen Li, Wen tau Yih, Yoon Kim, and James Glass. Diffcse: Difference-based contrastive learning for sentence embeddings, 2022.

[CTG+20] Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders, 2020.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[DHQZ24] Sumit Kumar Dam, Choong Seon Hong, Yu Qiao, and Chaoning Zhang. A complete survey on llm-based ai chatbots, 2024.

[DXWL25] Ziyi Dong, Yao Xiao, Pengxu Wei, and Liang Lin. Decoder-only llms are better controllers for diffusion models, 2025.

[GKC+19] Mansi Gupta, Nitish Kulkarni, Raghuveer Chanda, Anirudha Rayasam, and Zachary C Lipton. Amazonqa: A review-based question answering task, 2019.

[GYC22] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings, 2022.

[HCM+20] Matthew Henderson, Iñigo Casanueva, Nikola Mrkšić, Pei-Hao Su, Tsung-Hsien Wen, and Ivan Vulić. Convert: Efficient and accurate conversational representations from transformers, 2020.

[JHL+24] Ting Jiang, Shaohan Huang, Zhongzhi Luan, Deqing Wang, and Fuzhen Zhuang. Scaling sentence embeddings with large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 3182–3196, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[KTW+21] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2021.

[LESR19] Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. Benchmarking natural language understanding services for building conversational agents, 2019.

[LGB+16] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models, 2016.

[LHV24] I-Fan Lin, Faegheh Hasibi, and Suzan Verberne. Generate then refine: Data augmentation for zero-shot intent detection, 2024.

[LMP+19] Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[LPK+23] Yen-Ting Lin, Alexandros Papangelis, Seokhwan Kim, Sungjin Lee, Devamanyu Hazarika, Mahdi Namazifar, Di Jin, Yang Liu, and Dilek Hakkani-Tur. Selective in-context data augmentation for intent detection using pointwise V-information. In Andreas Vlachos and Isabelle Augenstein, editors, *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1463–1476, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics.

[LXL+24] Man Luo, Xin Xu, Yue Liu, Panupong Pasupat, and Mehran Kazemi. In-context learning with retrieved demonstrations for language models: A survey, 2024.

[MHZH22] Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. Generating training data with language models: Towards zero-shot language understanding, 2022.

[MRB23] Aristides Milios, Siva Reddy, and Dzmitry Bahdanau. In-context learning for text classification with many labels, 2023.

[POV21] Luis Perez, Lizi Ottens, and Sudharshan Viswanathan. Automatic code generation using pre-trained language models, 02 2021.

[RN18] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[SKF+24] Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. Repetition improves language model embeddings, 2024.

[SL24] Alexander Scarlatos and Andrew Lan. Reticl: Sequential retrieval of in-context examples with reinforcement learning, 2024.

[SRL+22] Gaurav Sahu, Pau Rodriguez, Issam Laradji, Parmida Atighehchian, David Vazquez, and Dzmitry Bahdanau. Data augmentation for intent classification with off-the-shelf large language models. In Bing Liu, Alexandros Papangelis, Stefan Ultes, Abhinav Rastogi, Yun-Nung Chen, Georgios Spithourakis, Elnaz Nouri, and Weiyan Shi, editors, *Proceedings of the 4th Workshop on NLP for Conversational AI*, pages 47–57, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[TM24] Llama Team and AI @ Meta. The llama 3 herd of models, 2024.

[TRJ+22] Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. Efficient few-shot learning without prompts, 2022.

[VSC+21] Ivan Vulić, Pei-Hao Su, Sam Coope, Daniela Gerz, Paweł Budzianowski, Iñigo Casanueva, Nikola Mrkšić, and Tsung-Hsien Wen. Convfit: Conversational fine-tuning of pretrained language models, 2021.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[YGL+22] Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. Zerogen: Efficient zero-shot learning via dataset generation, 2022.

[ZBY+21] Jianguo Zhang, Trung Bui, Seunghyun Yoon, Xiang Chen, Zhiwei Liu, Congying Xia, Quan Hung Tran, Walter Chang, and Philip Yu. Few-shot intent detection via contrastive pre-training and fine-tuning, 2021.

A Removing OOS from CLINC150

Algorithm 6 Filter Out-of-Scope (OOS) intent

```
clinc_150_data = load_dataset("clinc_oos", "plus")
intent_classes = clinc_150_data["train"].features["intent"].names
oos_idx = intent_classes.index("oos")
function FilterOOS(example)
    return example["intent"] ≠ oos_idx
end function
filtered_clinc150 = {split: clinc_150_data[split].filter(FilterOOS) for split in clinc_150_data}
```

Algorithm 7 Correctly remap intents after filtering OOS

```
new_labels = [label for label in intent_classes if label ≠ "oos"]
function RemapLabels(example)
    new_label = example["intent"]
    if new_label > oos_idx then
        new_label -= 1
    end if
    example["intent"] = new_label
    return example
end function
filtered_clinc150 = {split: ds.map(RemapLabels) for split, ds in filtered_clinc150}
```

B Bottom 10 intents sorted by metrics

B.1 Sorted by F1 score

Bottom 10 intents sorted by F1 score (Seed: 42)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	F1	Intent	F1	Intent	F1
user_name	0.150	order *	0.450	**change_ai_name**	0.429
smart_home *	0.341	**change_ai_name**	0.525	change_user_name	0.436
payday	0.359	shopping_list	0.576	todo_list	0.522
w2	0.364	todo_list	0.578	translate *	0.627
change_ai_name	0.385	change_user_name	0.613	reminder	0.649
cancel	0.387	meeting_schedule	0.638	apr	0.667
calories	0.410	apr	0.652	replacement_card_duration	0.680
flip_coin	0.431	change_speed	0.654	**user_name**	0.690
bill_balance	0.462	**user_name**	0.655	text	0.694
account_blocked	0.478	change_volume	0.655	goodbye	0.694

Table 21: Bottom 10 intents sorted by the F1 score metric for all three approaches (Seed: 42)

Bottom 10 intents sorted by F1 score (Seed: 0)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	F1	Intent	F1	Intent	F1
user_name	0.140	shopping_list *	0.000	user_name	0.182
smart_home	0.293	order	0.286	report_lost_card *	0.359
change_ai_name	0.333	tire_pressure	0.500	change_user_name	0.367
calories *	0.439	smart_home	0.509	change_ai_name	0.475
payday	0.455	apr	0.512	calendar_update	0.560
what_is_your_name	0.468	change_accent	0.571	ingredients	0.583
change_user_name	0.481	application_status	0.578	translate	0.636
w2	0.493	translate	0.588	reminder	0.638
schedule_maintenance	0.524	schedule_meeting	0.601	pto_used	0.638
schedule_meeting	0.538	play_music	0.621	what_is_your_name	0.651

Table 22: Bottom 10 intents sorted by the F1 score metric for all three approaches (Seed: 0)

Bottom 10 intents sorted by F1 score (Seed: 1337)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	F1	Intent	F1	Intent	F1
user_name	0.143	shopping_list	0.000	**smart_home**	0.421
smart_home	0.244	tire_pressure *	0.400	reminder *	0.467
account_blocked *	0.449	w2	0.462	ingredients	0.490
bill_balance	0.449	what_song	0.533	change_ai_name	0.475
what_is_your_name	0.449	**smart_home**	0.560	user_name	0.560
w2	0.480	change_language	0.604	todo_list	0.621
change_ai_name	0.489	translate	0.633	pto_balance	0.630
schedule_maintenance	0.511	credit_limit	0.640	change_user_name	0.636
cancel	0.516	schedule_maintenance	0.643	shopping_list	0.640
payday	0.524	calendar_update	0.647	sync_device	0.652

Table 23: Bottom 10 intents sorted by the F1 score metric for all three approaches (Seed: 1337)

Bottom 10 intents sorted by F1 score (OOS)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	F1	Intent	F1	Intent	F1
oos	0.004	**oos**	0.022	**oos**	0.311
user_name *	0.038	**change_ai_name**	0.390	definition	0.367
smart_home	0.132	order	0.412	order	0.381
w2	0.297	shopping_list *	0.489	**change_ai_name**	0.381
payday	0.341	todo_list	0.522	change_user_name	0.421
what_is_your_name	0.403	report_lost_card	0.533	todo_list	0.462
change_ai_name	0.407	travel_suggestion	0.560	what_can_i_ask_you *	0.466
cancel	0.413	definition	0.575	reminder	0.500
next_holiday	0.413	change_user_name	0.585	gas_type	0.578
account_blocked	0.417	ingredients	0.588	apr	0.627

Table 24: Bottom 10 intents sorted by the F1 score metric for all three approaches (OOS)

B.2 Sorted by precision

Bottom 10 intents sorted by precision (Seed: 42)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Prec	Intent	Prec	Intent	Prec
user_name *	0.300	**change_ai_name**	0.516	**change_ai_name**	0.462
flip_coin	0.306	whisper_mode	0.518	change_user_name	0.480
cancel	0.375	recipe *	0.571	reminder *	0.545
what_is_your_name	0.387	shopping_list	0.586	shopping_list	0.556
w2	0.400	bill_balance	0.590	cancel	0.634
next_holiday	0.404	change_user_name	0.594	apr	0.636
cook_time	0.439	min_payment	0.619	make_call	0.674
change_ai_name	0.455	account_blocked	0.632	whisper_mode	0.675
bill_due	0.541	schedule_meeting	0.652	next_holiday	0.683
bill_balance	0.545	what_is_your_name	0.667	restaurant_suggestion	0.707

Table 25: Bottom 10 intents sorted by precision for all three approaches (Seed: 42)

Bottom 10 intents sorted by precision (Seed: 0)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Prec	Intent	Prec	Intent	Prec
user_name *	0.231	shopping_list *	0.000	change_user_name	0.474
what_is_your_name	0.344	order	0.500	change_ai_name	0.483
change_ai_name	0.375	calendar_update	0.509	what_is_your_name	0.500
w2	0.436	smart_home	0.560	damaged_card *	0.537
flip_coin	0.439	how_busy	0.582	recipe	0.580
cancel	0.467	travel_notification	0.588	todo_list	0.632
next_holiday	0.478	bill_due	0.591	distance	0.643
bill_due	0.515	change_accent	0.615	pto_balance	0.657
cook_time	0.528	play_music	0.643	restaurant_suggestion	0.674
change_user_name	0.542	are_you_a_bot	0.659	last_maintenance	0.675

Table 26: Bottom 10 intents sorted by precision for all three approaches (Seed: 0)

Bottom 10 intents sorted by precision (Seed: 1337)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Prec	Intent	Prec	Intent	Prec
user_name *	0.250	shopping_list	0.000	reminder *	0.467
what_is_your_name	0.339	how_busy *	0.542	shopping_list	0.560
w2	0.400	yes	0.578	change_ai_name	0.583
flip_coin	0.406	calendar_update	0.579	recipe	0.583
next_holiday	0.429	no	0.585	change_user_name	0.583
smart_home	0.455	pto_balance	0.609	damaged_card	0.614
cancel	0.500	vaccines	0.610	what_is_your_name	0.622
change_user_name	0.523	find_phone	0.622	cancel	0.628
cook_time	0.547	translate	0.633	ingredients	0.632
account_blocked	0.579	bill_balance	0.657	todo_list	0.643

Table 27: Bottom 10 intents sorted by precision for all three approaches (Seed: 1337)

Bottom 10 intents sorted by precision (OOS)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Prec	Intent	Prec	Intent	Prec
user_name *	0.043	change_ai_name	0.319	definition	0.231
smart_home	0.109	shopping_list	0.367	order	0.267
w2	0.250	order	0.368	what_can_i_ask_you *	0.314
what_is_your_name	0.270	travel_suggestion *	0.400	change_ai_name	0.364
next_holiday	0.275	definition	0.422	reminder	0.364
car_rental	0.330	ingredients	0.455	change_user_name	0.444
flip_coin	0.354	directions	0.490	gas_type	0.453
cook_time	0.361	weather	0.500	shopping_list	0.517
food_last	0.362	schedule_maintenance	0.520	cancel	0.520
cancel	0.394	vaccines	0.536	oil_change	0.537

Table 28: Bottom 10 intents sorted by precision for all three approaches (OOS)

B.3 Sorted by recall

Bottom 10 intents sorted by recall (Seed: 42)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Recall	Intent	Recall	Intent	Recall
user_name	0.100	order *	0.300	todo_list	0.400
smart_home *	0.233	todo_list	0.433	**change_ai_name**	0.400
payday	0.233	meeting_schedule	0.500	change_user_name *	0.400
calories	0.267	apr	0.500	translate	0.533
change_ai_name	0.333	**change_ai_name**	0.533	text	0.567
w2	0.333	shopping_list	0.567	replacement_card_duration	0.567
account_blocked	0.367	change_speed	0.567	goodbye	0.567
bill_balance	0.400	**user_name**	0.600	**user_name**	0.667
how_busy	0.400	travel_alert	0.600	order	0.667
cancel	0.400	change_volume	0.600	shopping_list	0.667

Table 29: Bottom 10 intents sorted by recall for all three approaches (Seed: 42)

Bottom 10 intents sorted by recall (Seed: 0)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Recall	Intent	Recall	Intent	Recall
user_name	0.100	shopping_list *	0.000	user_name	0.100
smart_home	0.200	order	0.200	report_lost_card *	0.233
change_ai_name	0.300	tire_pressure	0.333	change_user_name	0.300
calories *	0.300	apr	0.367	change_ai_name	0.467
payday	0.333	application_status	0.433	translate	0.467
schedule_maintenance	0.367	**smart_home**	0.467	calendar_update	0.467
account_balance	0.433	schedule_meeting	0.467	ingredients	0.467
play_music	0.433	translate	0.500	**smart_home**	0.500
change_user_name	0.433	change_accent	0.533	reminder	0.500
schedule_meeting	0.467	confirm_reservation	0.567	pto_used	0.500

Table 30: Bottom 10 intents sorted by recall for all three approaches (Seed: 0)

Bottom 10 intents sorted by recall (Seed: 1337)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Recall	Intent	Recall	Intent	Recall
user_name	0.100	shopping_list	0.00	**smart_home**	0.267
smart_home	0.167	tire_pressure *	0.267	ingredients *	0.400
account_blocked *	0.367	w2	0.300	user_name	0.467
bill_balance	0.367	what_song	0.400	change_ai_name	0.467
change_ai_name	0.367	**smart_home**	0.467	reminder	0.467
payday	0.367	meeting_schedule	0.533	sync_device	0.500
schedule_maintenance	0.400	change_language	0.533	shopping_list	0.533
restaurant_reservation	0.467	credit_limit	0.533	card_declined	0.567
how_busy	0.467	apr	0.533	pto_balance	0.567
calories	0.500	order	0.567	todo_list	0.600

Table 31: Bottom 10 intents sorted by recall for all three approaches (Seed: 1337)

Bottom 10 intents sorted by recall (OOS)					
Prompting LLM		**Data augmentation**		**Sentence embeddings**	
Intent	Recall	Intent	Recall	Intent	Recall
oos	0.002	**oos**	0.011	**oos**	0.184
user_name	0.033	report_lost_card *	0.400	todo_list	0.400
smart_home *	0.167	todo_list	0.400	**change_ai_name**	0.400
payday	0.233	order	0.467	change_user_name *	0.400
calories	0.300	**change_ai_name**	0.500	translate	0.533
account_balance	0.333	bill_balance	0.567	text	0.567
change_ai_name	0.367	transactions	0.567	replacement_card_duration	0.567
w2	0.367	fun_fact	0.567	goodbye	0.567
bill_balance	0.400	what_is_your_name	0.600	user_name	0.667
how_busy	0.400	apr	0.600	order	0.667

Table 32: Bottom 10 intents sorted by recall for all three approaches (OOS)

C Confusion matrices

C.1 Top 20 most confused intents in CLINC150 with 150 intents

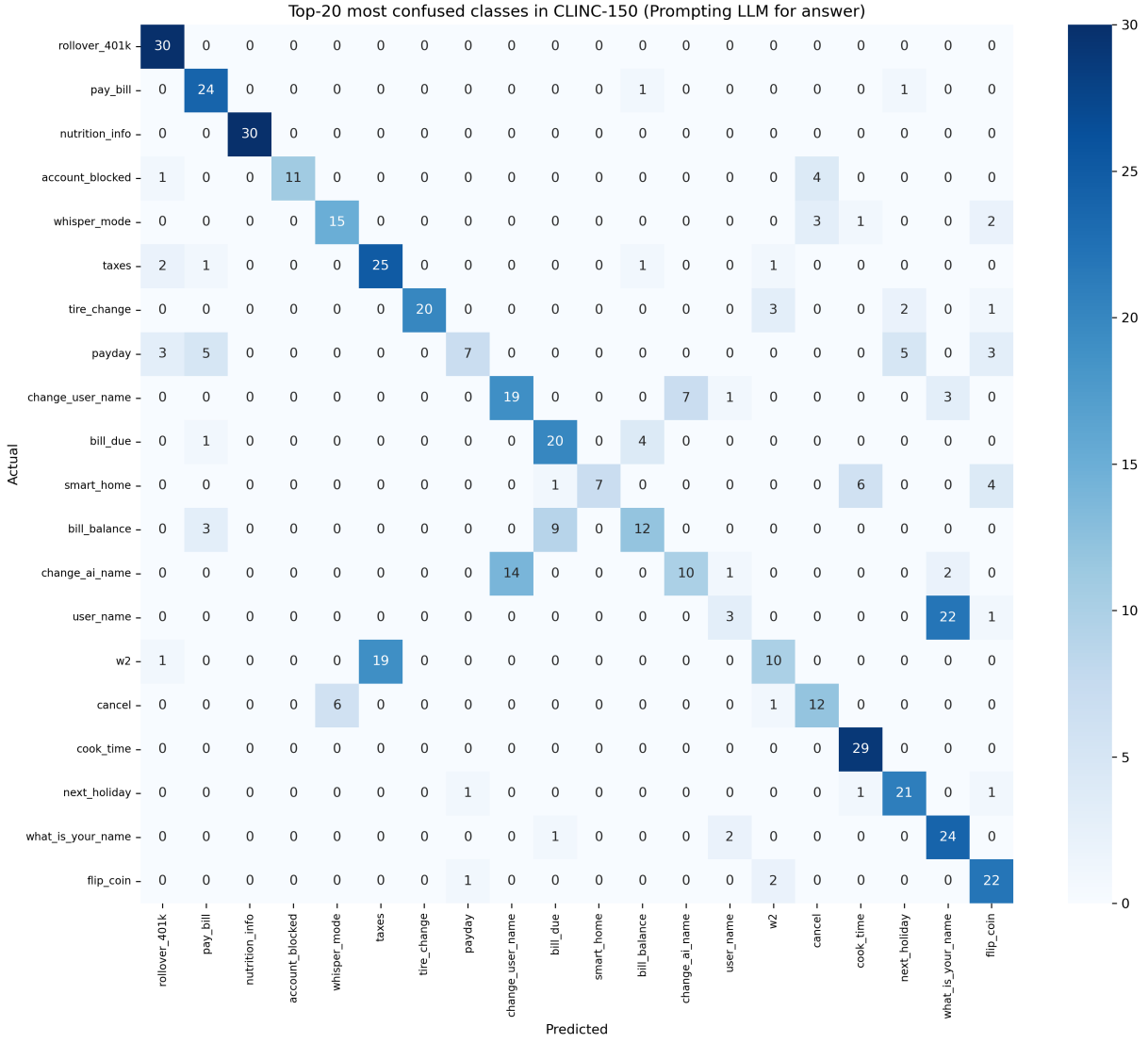

Figure 2: Top 20 most confused intents in CLINC150 (Prompting LLM for the label directly)

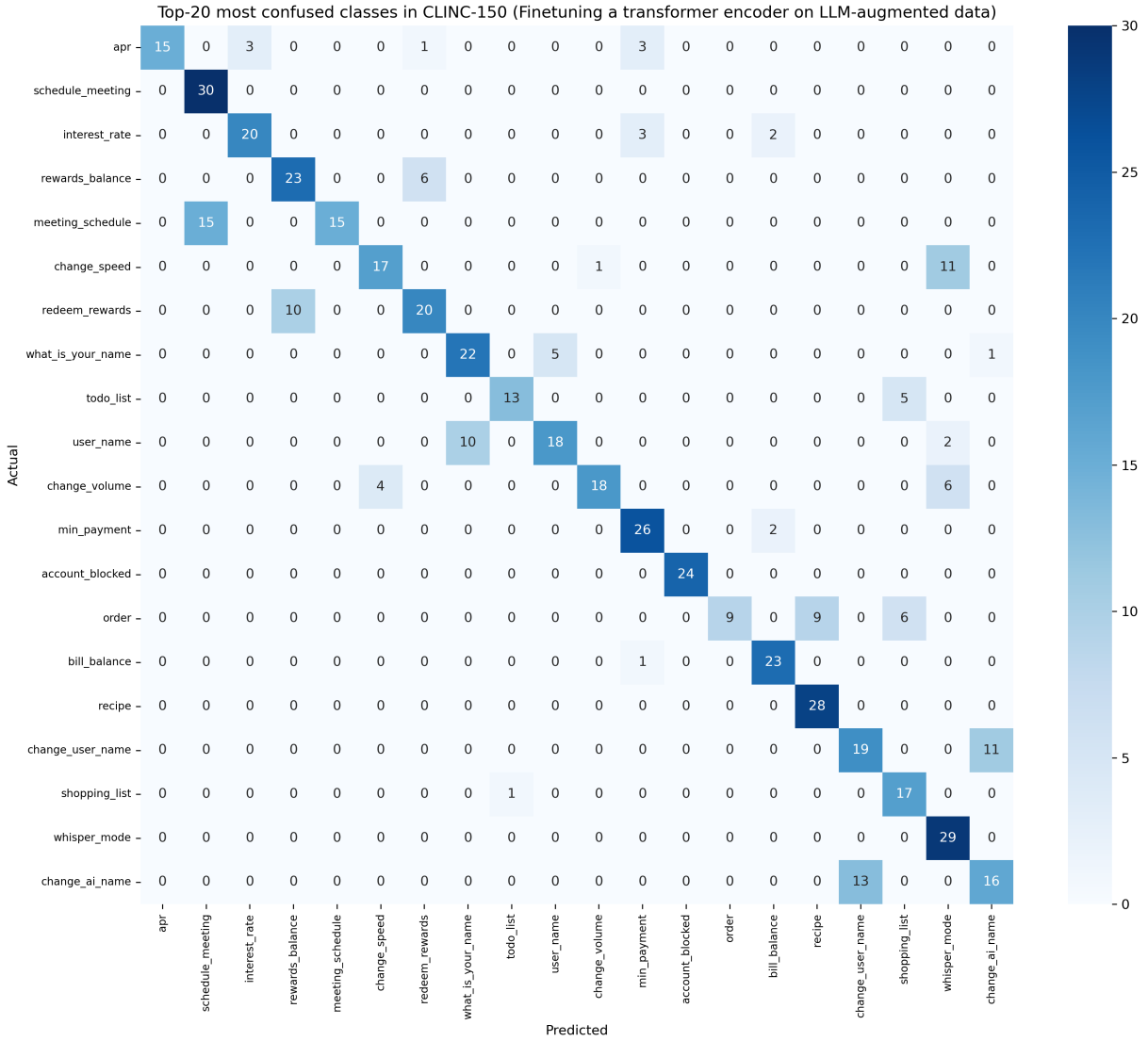

Figure 3: Top 20 most confused intents in CLINC150 (Finetuning a small transformer encoder on LLM-augmented data)

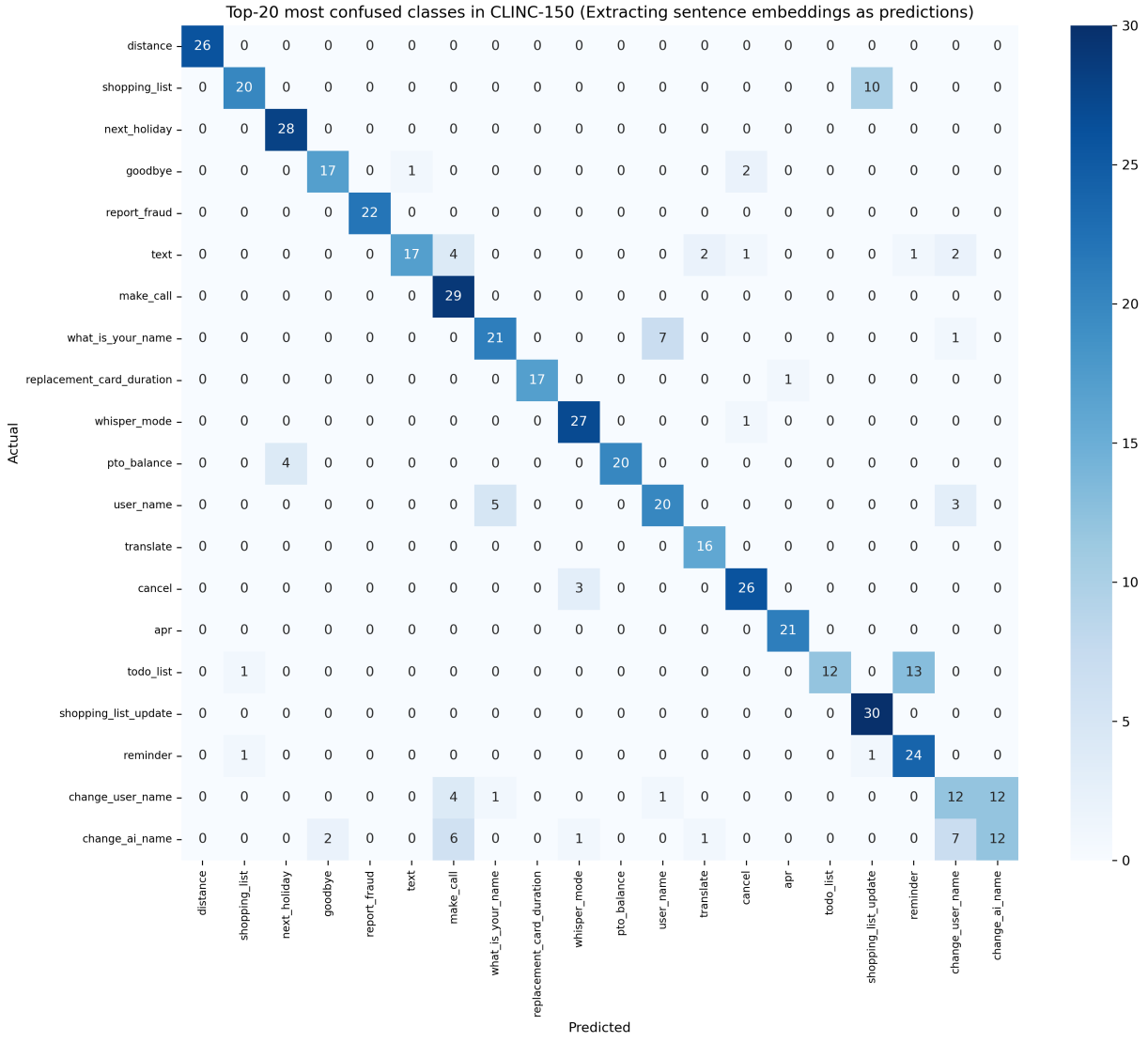

Figure 4: Top 20 most confused intents in CLINC150 (Extracting sentence embeddings as predictions)

C.2 Top 20 most confused intents in CLINC150 OOS

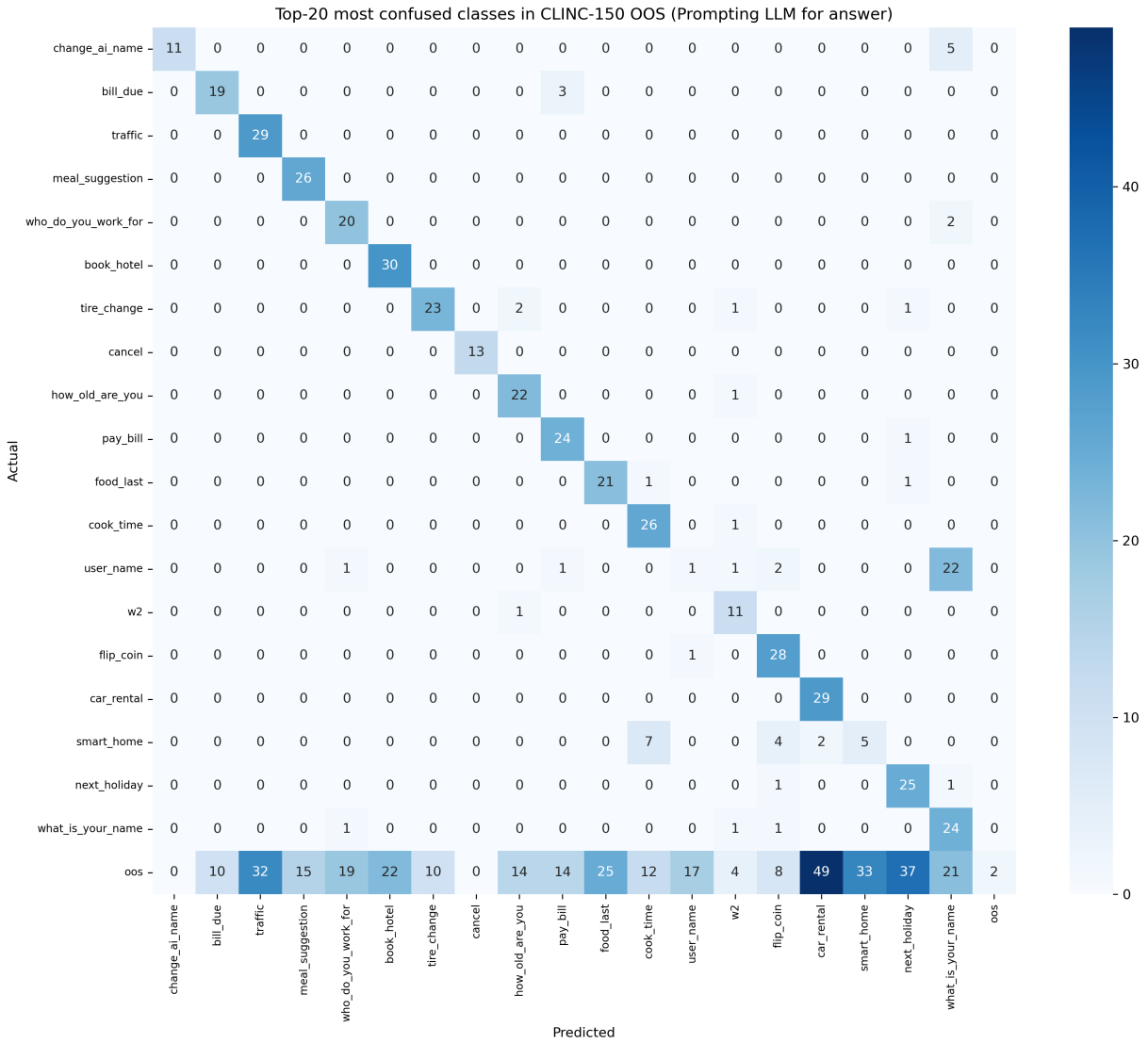

Figure 5: Top 20 most confused intents in CLINC150 with OOS (Prompting LLM for the label directly)

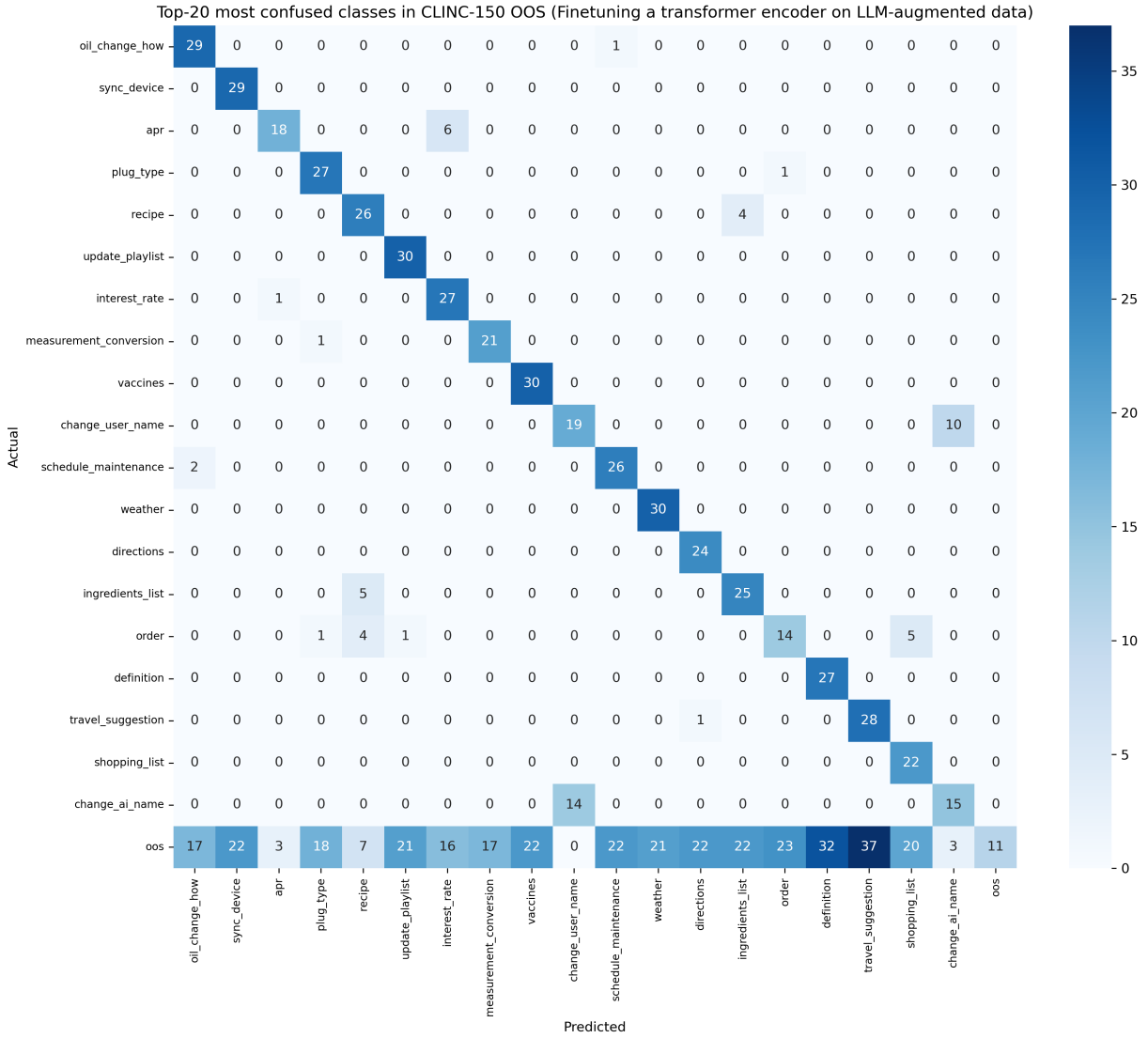

Figure 6: Top 20 most confused intents in CLINC150 with OOS (Finetuning a small transformer encoder on LLM-augmented data)

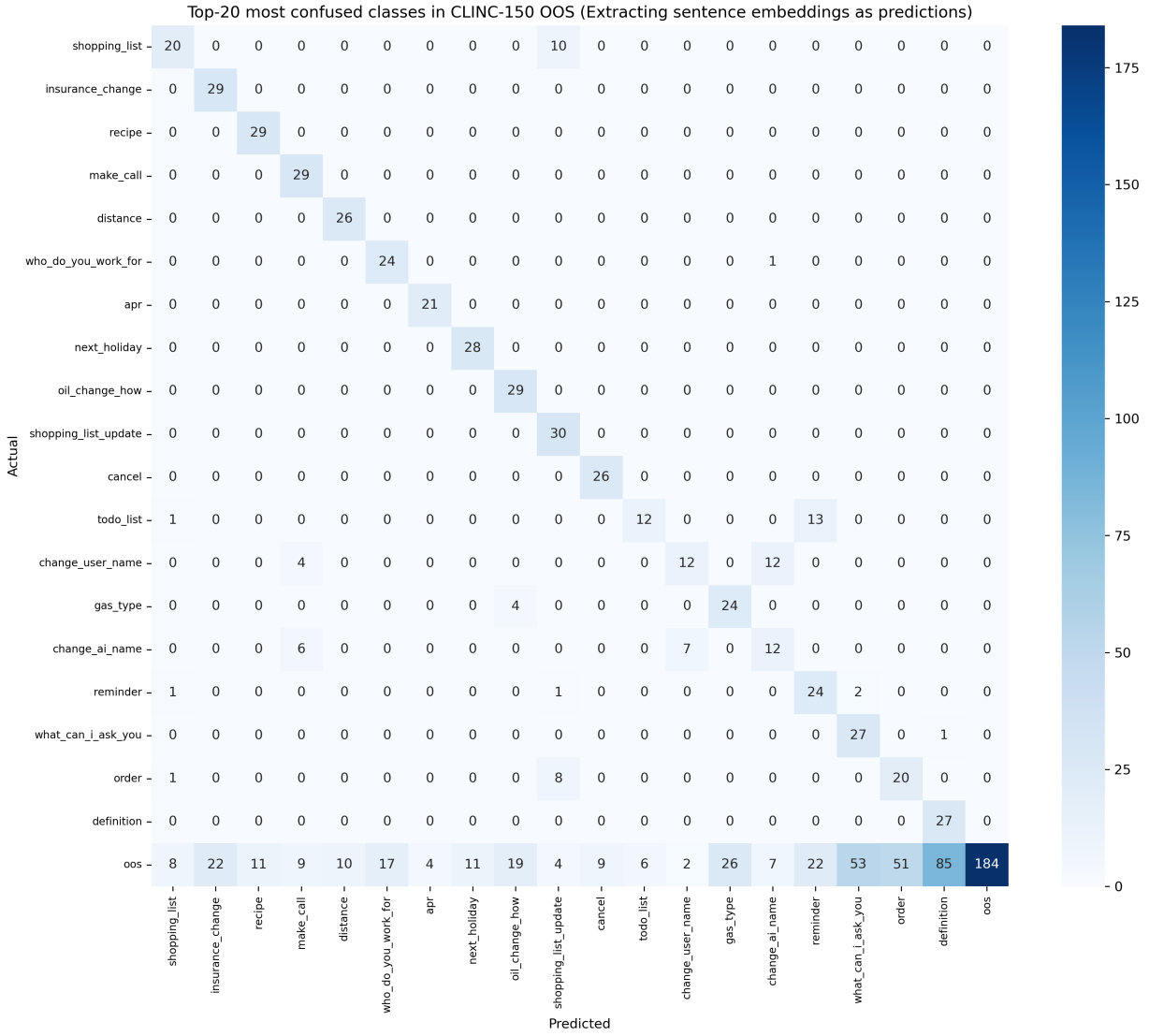

Figure 7: Top 20 most confused intents in CLINC150 with OOS (Extracting sentence embeddings as predictions)