Automated Dialogue Order Processing using Small Large Language Models

1st Adithya Sudhan
Faculty of Computer Science
Memorial University of Newfoundland
St. Johns, Canada
asudhan@mun.ca

2nd Dr. Reza Shahidi
Faculty of Engineering
Memorial University of Newfoundland
St. Johns, Canada

3rd Dr. Adrian Fiech
Faculty of Computer Science
Memorial University of Newfoundland
St. Johns, Canada

***Abstract*—This paper addresses the underexplored intersection of prompt engineering techniques and system design in the context of small Large Language Models (LLMs) aimed at generating structured outputs. While prior studies have largely focused on larger LLMs, the potential of smaller models, which operate with significantly fewer parameters, remains largely untapped, particularly concerning entity extraction in real-world task-oriented dialogues. We propose a framework for deploying small LLMs in private, resource-efficient environments to enhance task-oriented workflows. Utilizing the TaskMaster-1 (TM-1-2019) dataset, our research demonstrates how structured outputs can be generated effectively in the absence of any fine-tuning. We evaluate various small LLMs and analyze critical metrics such as validity to identify the key factors and components necessary for transforming dialogue into actionable entities within programming environments.**

***Index Terms*—Large Language Models, small-scale LLMs, automated dialogue processing, order placement, prompt engineering**

I. Introduction

While several studies have explored the use of prompt engineering techniques with Large Language Models (LLMs), few, if any, have focused on how prompt engineering and system design can best be used to create structured outputs with small LLMs, which use far fewer parameters. Moreover, entity extraction has not been explored within the context of real world task oriented dialogues using such small LLMs. This paper tries to address this gap and outlines an approach that could lead to private, small-scale, resource-wise deployments of small LLMs to augment task-oriented order taking workflows in a real world setting. This study, using the TaskMaster-1 (TM-1-2019) dataset, focuses on pizza-ordering as a domain as a proxy for all such order taking workflows where there is somewhat predictable structure to the contents of the conversation. This is typical of the dialogue that occurs at drivethroughs, for example. Through the use of various small LLMs, we walk through metrics like validity to point out the limiting factors and essential components of a system that converts dialogue into entities (represented by objects in any programming language).

II. Data preparation

The dataset used in the study is the TaskMaster-1 (TM-1-2019) [2] dataset that contains real-world 2-person spoken dialogue collected using the Wizard of Oz (WoZ) methodology in various domains. Of this large collection, only the subset pertaining to pizza food orders was used. Once acquired, the dialogues are concatenated into a single string block so that the LLM may act on it in the appropriate stage. The following is an portion of a processed dialogue:

```
ASSISTANT: hi, how can i help you?
USER: I want to order pizza.
ASSISTANT: ok, from where?
USER: Bella Luna.
ASSISTANT: what would you like to order?
USER: chicken barbecue.
ASSISTANT: ok, 1 pizza?
```

III. System Design

The proposed approach follows a three-stage process for converting natural language input into structured data. It is designed to handle dialogue-based input across various domains by summarizing the relevant information, converting it into function calls, and generating structured outputs. To illustrate this process, we use the example of pizza ordering, but the approach can be generalized to other applications.

A. Summarization

The first stage involves summarizing the dialogue to extract relevant information. This step focuses on identifying key details from the dialogue that are necessary for fulfilling the user's request while filtering out irrelevant parts. For instance, in the pizza-ordering example, the goal is to capture details such as the number of pizzas, sizes, crust types, toppings, and any special instructions. The summarization is performed by generating a prompt for a language model that instructs it to provide a concise representation of the essential elements of the conversation. This summary forms the basis for subsequent stages and can be easily adapted to other domains by modifying the prompt to focus on different kinds of information.

B. Function Calling

The second stage translates the summarized information into function calls that represent the user's request in a structured format. This involves using the output from the summarization step to generate specific function calls. For example, the pizza order summary can be used to create a string presentation of a function call like `create_pizza(quantity, size, crust, toppings, special_instructions)`. This structured representation facilitates automation by mapping natural language input to pre-defined functions. In other domains, the function calls can be adapted to suit different types of actions or entities, making this stage highly flexible.

C. Structured Output Generation

In the final stage, the generated function calls are used to create structured objects based on pre-defined classes. This is a crucial stage because the somewhat structured response of the LLM needs to be parsed using regular expressions to make it possible to capture the requisite parameters. Naturally, the efficacy of the regular expression can be a huge factor in correctly producing objects.
The regular expression used in this study is

```
create_pizza(\s*quantity=(\d+),\s*size
    =[’”]([\w\s\-]+)[’”],\s*crust=[’”]([\w
    \s\-]+)[’”],\s*toppings=\[([^\]]*)\],\
    s*special_instructions=\s*(?:\[\s
    *[’”]?([^’”]*)[’”]?\s*\]|[’”]([^’”]*)
    [’”])\s*\)
```

.Once parsed, for the pizza example, each order is represented as an instance of a `Pizza` class, encapsulating attributes such as quantity, size, crust type, toppings, and special instructions. This allows for further validation, post-processing, or integration with other systems.

IV. Results

4 models, as mentioned in Table I, were tested across 3 different few-shot prompts, and therefore their results are presented in Fig. 1 for the purpose of comparison. Only 100 dialogues were used for testing for the sake of lowering run times on the GPU, and the metrics focused on were all centered around validity.

V. LLMs tested

TABLE I
Models Used with Annotations

Model	**Annotation**
mlx-community/Meta-Llama-3.1-8B-Instruct-4bit	L8
mlx-community/Mistral-7B-Instruct-v0.3-4bit	M7
mlx-community/Llama-3.2-3B-Instruct-4bit	L3
mlx-community/Qwen2-7B-Instruct-4bit	Q7

A. Criteria for validity

- **Size Validation:** A pizza size is considered valid if it matches any of the following predefined sizes: *small*, *medium*, or *large*. The validation uses a fuzzy matching algorithm that requires at least 80% similarity, based on Levenshtein distance, between the provided size and the valid options.
- **Crust Validation:** A pizza crust type is deemed valid if it is one of the following: *thin*, *thick*, *gluten-free*, *deep*, or *stuffed*. If the provided crust type does not exactly match any valid options, it is further evaluated by splitting the crust string into components (replacing hyphens with spaces) and checking each component for at least 80% similarity, based on Levenshtein distance, against the valid crust types.
- **Quantity Validation:** The quantity of pizzas ordered must be a positive integer. This ensures that the order is valid only if the quantity is greater than zero.
- **Toppings Validation:** The toppings must be provided as a list. The list is valid if it is either empty (represented as `['']`) or contains only non-empty strings. This criterion ensures that each topping must be a valid string representation.
- **Special Instructions Validation:** Special instructions for the pizza order may be provided as either a string or a list. This flexibility allows customers to include additional requests or details regarding their order.

Accuracy measurements, which would cater to whether the well formed pizza objects actually conform to the corresponding spoken dialogue, were not taken since this would either require truth values from either manual annotation or a high parameter-LLM.

VI. Conclusions

This study shows that small LLMs can reliably act on highly unstructured spoken dialogue and turn them into structured, actionable and deterministic software entities that can be consumed by another software system. This opens up the possibilities of using private small LLMs for sites that have limited or no access to the internet, and workflows where sending data off site is not acceptable. It must be noted however, that the hybrid-probabilistic system described in this paper, must be thoroughly tested and validated before using it on sensitive applications where behavior is required to be deterministic. LLMs are leaky abstractions and excel at summarization and following guidance, as demonstrated via function calling, but they still must be treated as systems that create probabilistic outputs whose integrity must be questioned perpetually. Such hybrid software systems therefore must be designed to be fault-tolerant and a greater emphasis must be placed on which failure scenarios since these failures may stem from imbalances in the training data that the base LLM was trained on.

VII. Limitations and next steps

The test dataset in this study only covers 100 pizza ordering dialogues. This could be expanded to cover many more domains as well as a greater number of dialogues to gain a better understanding of whether the validity holds for large amounts of data. Furthermore, a greater number of small LLMs maybe tested to understand the strengths and weaknesses of each model and perhaps conclude upon a recommended small LLM that works best across various task-domains. Lastly, there is ongoing work on structured generation using context-free grammars and regex-structured generation via the Outlines library [1] which is worth exploring for small LLMs. These work by constraining the probability distribution of the next token to predetermined regular expressions or known patterns, so as to guarantee that the LLM always produces predictable structured output. The core idea in this approach is that instead of using regular expressions after an LLM generates output, as is explained in this paper, regex could be used to guide the generation of tokens themselves. This study could be enhanced by exploring these techniques, especially in the context of checking accuracy of such regex guided LLM generation.

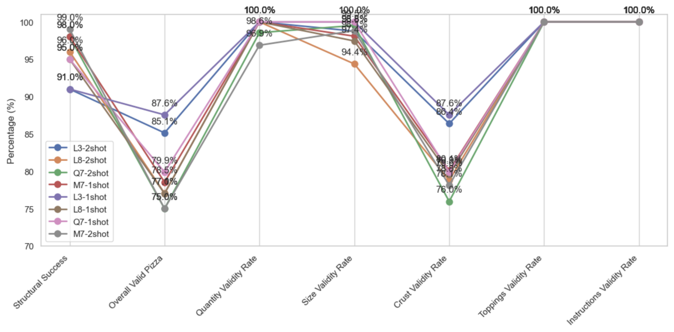

Fig. 1. Validity metrics are high across models

References

[1] Willard, Brandon T and Louf, "Efficient Guided Generation for LLMs," *arXiv preprint arXiv:2307.09702*, 2023.

[2] Byrne, Bill and Krishnamoorthi, Karthik and Sankar, Chinnadhurai and Neelakantan, Arvind and Duckworth, Daniel and Yavuz, Semih and Goodrich, Ben and Dubey, Amit and Kim, Kyu-Young and Cedilnik, Andy, "Taskmaster-1: Toward a Realistic and Diverse Dialog Dataset," *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2019.