

LLM-Based NLU for Explanatory Dialogue

alexander.berman@gu.se

February 28, 2024

1 Introduction

This report presents and discusses the integration of a large language model (LLM) based natural-language understanding (NLU) component into an explanatory dialogue system. The dialogue system is a chatbot that assists users in a game that revolves around assessing whether people are introverted or extraverted based on their music preferences. This chatbot is embedded in an interactive web page together with other elements of the game such as visualizations of music preferences, audio controls and navigation buttons. (The game will be further elaborated in the course project report.)

2 Motivation

One of the reasons for opting for an LLM-based NLU component is to be able to interpret user utterances in a more accurate and robust way than with a simple keyword- or rule-based component. Furthermore, it was hypothesized that an LLM-based solution would require less effort in terms of development time than a more conventional machine-learning based approach (such as logistic regression or support vector machine for intent classification) due to the ability to use zero- or few-shot learning, which in principle can drastically reduce the amount of required training data. Finally, it was deemed that the fine-grained semantics underpinning the dialogue modeling in the project would not be supported well by a more conventional approach based on intent classification and entity extraction. In contrast, with an LLM the problem can be approached as a sequence-to-sequence task, i.e. as a transformation from natural language to an expression in formal language whose syntax and semantics is fed to the LLM in the prompt.

3 Problem formulation

In conventional approaches, NLU for dialogue systems is typically divided into two “tasks”: intent classification and entity extraction. For example, the user utterance “I need a ticket to Paris” can be understood as expressing the intent `BookTicket` and the entity `Paris` (possibly assigned to the role `Destination`). In contrast, here the task is framed as a matter of expressing the meaning of a given natural-language utterance in a formal language. For example, the meaning of the utterance above might be expressed as a sequence of dialogue moves such as `[request(BookTicket), answer(dest_city(Paris))]`.

In general formal terms, given a user utterance U and a description L of a formal language, the NLU component should return an expression E in L that conveys the meaning of U . The value of L in the context of the specific domain is provided in Listing 1 (for source code, see `project/dialog/music_personality/nlu_openai.config.yml`).

```

The output can be one of the following:
- Ask(Question): the user asks a question
- ICM(understanding, negative): the user signals negative
understanding
- ICM(acceptance, positive): the user acknowledges
- None: the input from the user does not match any of the
above

Question can be one of:
- Why(): bare why question
- Why(Proposition): explicit why question concerning a
proposition
- BooleanQuestion(Proposition): question concerning whether a
proposition is true
- WhQuestion(FactorsConsidered): question concerning which
factors the system considers

Proposition can be one of:
- Extraverted(): the person is considered extraverted
- Not(Extraverted()): the person is considered introverted
- HighValue(X): the person likes music with high X, where X
is an audio feature
- Not(HighValue(X)): the person likes music with low X, where
X is an audio feature
- Explains(Explanans, Explanandum): Explanans explains
Explanandum (both being propositions)
- Supports(Antecedent, Consequent): Antecedent supports
Consequent (both being propositions)

Audio features:
- energy_mean
- mode_0_percentage (low values indicate a preference for
music with minor mode, while high values indicate a
preference for major mode)
- loudness_mean
- speechiness_mean
- instrumentalness_mean
- valence_mean
- danceability_mean

```

Listing 1: Description of formal language for conveying meaning of user utterances in the domain at hand.

4 Prompt design

The prompt contains a description of the task and of the formal language as well as examples of inputs and outputs, and has been iterated in tandem with testing (see section 6). The most recent version of the prompt can be found in `project/dialog/music_personality/nlu_openai_config.yml`.

5 Implementation

OpenAI’s chat completion API is used as follows: Given the user utterance U , a prompt describing the task is provided as an initial system message, with L embedded into the prompt. U is then fed as a user message. As completion, an assistant message M is obtained as output (see `project/dialog/music_personality/nlu_openai.py`).

In order for the output M to be used as input to the dialogue manager, it also needs to be parsed from a string (semantic expression) into a dialogue move. The dialogue system at hand is implemented in Python and uses custom types and classes to represent dialogue moves and their content. For example, instances of the class `Ask` denote ask-moves, and contain an instance of the class `Question`. To streamline the process of parsing semantic expressions, L is formulated in way that literally reflects the syntax of the internal semantics of the dialogue system. Hence, output from the LLM can be treated as executable Python code and can in principle be parsed by passing it to the built-in Python function `eval`. However, such a solution would not be secure since the output from LLM cannot be guaranteed to abide to the instructed form and content. Instead, a secure parser has been implemented. This parser uses Python’s `ast` module, which enables Python code to be parsed into an abstract syntax tree. The parser iterates this tree recursively to construct a semantic structure and raises an exception for unsupported (and potentially unsafe) expressions (see `project/dialog/semantic_serialization.py`).

6 Evaluation

At this point, the integration has only been evaluated unsystematically by manually testing the dialogue system and purposely trying user utterances other than the ones exemplified in the prompt. Initially, it was noted that the LLM (GPT 3.5) generated output not only in the formal language, but also accompanying elaborations in natural language. This problem was solved by adding an instruction to the prompt to “(o)nly output the semantic representation without any surrounding content.” However, the model would occasionally generate syntactically invalid output (mismatched parentheses). After switching to GPT 4, no occurrences of invalid syntax have been encountered.

As for accuracy, the impression so far is that GPT 4 performs very well. One interesting observation concerns deixis: when the utterance “why does that make it more likely that the person is extraverted?” was tried, in response to the system asserting that the person likes silent music, GPT produced the following interpretation: `Ask(Why(Supports(Antecedent, Extraverted())))`. Worth noting is that the first argument to `Supports`, semantically corresponding to the deictic demonstrative “that”, is `Antecedent`, i.e. underspecified. In other words, GPT seems to capture the deictic function of “that” in a relevant way, without having seen any examples of this kind. (At this point, however, the dialogue manager does not

support underspecified semantic content. Alternatively, one can consider letting the LLM perform deictic resolution itself by feeding dialogue history as input.)

In future work it would be useful to evaluate performance more systematically, e.g. by collecting dialogue data, including user utterances and GPT’s interpretations and annotating expected (correct) interpretations. This way, accuracy can be measured and potential misinterpretations be studied.

7 Conclusions