Natural language understanding from traditional methods to large language models

Nicolas BOCK
xbockni00@fit.vut.cz

Santosh KESIRAJU
kesiraju@fit.vut.cz

Abstract

Natural Language Understanding (NLU) plays a key role in modern dialogue systems, enabling tasks such as Intent Classification (IC) and Slot-Filling (SL). These tasks are closely linked, as the type of user intent (e.g. "play music") often determines the relevant slots to be filled (e.g. "song" or "artist"). Traditional NLU systems rely on cascaded or naive joint approaches, which can propagate errors between tasks, reducing overall system performance. In contrast, instruction-tuned large language models (LLMs) offer a unified framework for addressing IC and SL jointly, potentially enhancing accuracy and efficiency.

This project evaluates three approaches to IC and SL: (1) traditional methods using word embeddings like Word2Vec, (2) fine-tuning pre-trained models such as BERT, and (3) in-context learning using instruction-tuned LLMs like Flan-T5-base from Google. Experiments were conducted on NLU-Evaluation Benchmark dataset.

Our findings reveal that while traditional methods provide a strong baseline, fine-tuned models significantly improve performance. Furthermore, in-context learning with LLMs achieves comparable results without the need for task-specific fine-tuning, demonstrating their versatility. This study highlights the evolution of NLU methodologies and the advantages of leveraging LLMs for joint IC and SL tasks in dialogue systems.

1 Introduction

Natural Language Understanding (NLU) powers dialogue systems by enabling tasks like Intent Classification (IC) and Slot-Filling (SL). IC identifies the purpose of a user's query, while SL extracts specific details to fulfill it. Traditional methods often rely on cascaded approaches, which can propagate errors between tasks, reducing performance.

With the rise of large language models (LLMs), such as instruction-tuned models, joint IC and SL tasks can be addressed more effectively. These models reduce dependency errors and improve accuracy without extensive task-specific fine-tuning.

This project explores three approaches: traditional word embeddings, fine-tuning pre-trained models, and in-context learning with LLMs. Experiments on standard NLU datasets aim to compare their performance and assess the strengths of each method. The findings highlight the evolution of NLU techniques and the advantages of LLMs in building robust dialogue systems.

2 Task Definition

In this project, we worked on two key tasks in Natural Language Understanding (NLU):

- **Intent Classification (IC)**: This involves identifying the purpose of the user's input in a dialogue system. For example, the sentence "Turn on the light in the living room at 7 PM" corresponds to the intent **hue_lighton**.
- **Slot-Filling/Slot-Labeling (SL)**: This involves extracting key details or entities from the user's input. For the same example, the slots would be: **[switch_light**: *Turn on*, **place**: *living room*, **time**: *7 PM***]**

The interdependence between IC and SL makes it important to address these tasks jointly. For instance, knowing the intent helps identify relevant slots. For example:

- Flow of Input Processing:
  - User Input → IC → SL → System Response

– Example:
  * INPUT: "*Book a flight to New York on January 5th.*"
  * IC: **book_flight**
  * SL: **destination**: "*New York*", **date**: "*January 5th*"
  * OUTPUT: "*Your flight to New York on January 5th is booked.*"

This project explores three different approaches:

1. Traditional methods based on word-embeddings like Word2Vec.
2. Fine-tuning pre-trained models like BERT.
3. In-context learning without any fine-tuning like Flan-T5-base from Google.

We aim to compare these methods in terms of performance and efficiency, focusing on the NLU-Evaluation Benchmark dataset.

3 Method

3.1 Traditional Methods: Word Embeddins and Classifiers

3.1.1 Intent Classification

Intent classification involves predicting the underlying intent of a given sentence. The approach relies on a combination of **Word2Vec embeddings** and traditional machine learning classifiers.

- **Word2Vec Embedding:** Each input sentence is first tokenized into words and converted into an **embedding**. Word2Vec is an unsupervised model that maps each word in the corpus to a high-dimensional vector. The vector representation of a word captures semantic relationships between words, enabling the model to learn contextual meanings. For each sentence, a fixed-length vector is obtained by averaging the embeddings of the individual words. The embedding for a sentence is denoted as $\mathbf{x}_i \in \mathbb{R}^d$, where d is the dimension of the embedding, typically set to 100 in this case.

  Let the sentence S_i consist of n words $w_1, w_2, \ldots, w_n$ and the corresponding word embeddings $\mathbf{w}_j \in \mathbb{R}^d$. The sentence embedding $\mathbf{x}_i$ is calculated as:

$$\mathbf{x}_i = \frac{1}{n} \sum_{j=1}^{n} \mathbf{w}_j$$

  This process generates a vector that represents the semantic meaning of the entire sentence.

- **Classifier:** After obtaining sentence embeddings, **Logistic Regression** and **Random Forest** classifiers are trained on the embeddings for intent prediction. These models are trained to predict a categorical intent for each input sentence, where each class represents a specific intent. Let the set of intents be denoted as $Y = \{y_1, y_2, \ldots, y_k\}$, where k is the number of possible intents. The objective is to predict the intent y_i for each sentence S_i, given the embedding $\mathbf{x}_i$ as input.

  The Logistic Regression model computes the probability of each intent y_i using a softmax function, which is trained by minimizing the cross-entropy loss:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{N} \sum_{j=1}^{k} y_{ij} \log(p(y_j \mid \mathbf{x}_i))$$

  where N is the number of sentences, y_{ij} is the true label (one-hot encoded), and $p(y_j \mid \mathbf{x}_i)$ is the predicted probability of intent y_j.

3.1.2 Slot-Filling / Slot-Labeling

Slot-filling is the task of extracting specific pieces of information from a sentence, often in the form of predefined **slots** (such as *dates*, *times*, and *locations*). For this task, we employ the **BIO tagging scheme** (*Begin, Inside, Outside*) to label tokens in the sentence based on wether they correspond to a slot.

- **Slot Extraction:** Given the annotated sentence, slots are extracted using a regular expression that identifies entities enclosed in square brackets. The annotation for each sentence is assumed to contain entities in the format **[slot_type: slot_value]**. The slot types and their corresponding values are then stored as pairs (s_j, v_j), where s_j represents the slot type, and v_j represents the value for that slot.

- **BIO Labeling:** The BIO (*Begin, Inside, Outside*) scheme is applied to the tokens in the sentence, where "**B-**" marks the beginning of a slot, "**I-**" marks subsequent tokens inside the same slot, and "**O**" indicates tokens outside any slots. For a sentence S_i with tokens

$\{t_1, t_2, \ldots, t_n\}$, we assign a label l_i for each token:

$$l_i = \begin{cases} \textbf{B-}s_j & \text{if } t_i \text{ is the first token of slot } s_j, \\ \textbf{I-}s_j & \text{if } t_i \text{ is a token inside slot } s_j, \\ \textbf{O} & \text{if } t_i \text{ is outside any slot.} \end{cases}$$

- **Modeling Slot-Filling:** Similar to intent classification, Word2Vec embeddings are used to represent the tokens in a sentence. For each sentence, the embeddings are computed by averaging the embeddings of the tokens in that sentence. A **One-vs-Rest** classification approach is employed, where a classifier is trained for each possible slot type. The model predicts a binary classification label for each token, indicating whether the token belongs to a specific slot type. The **alignment of BIO labels** is then performed to ensure that each token's predicted label corresponds to the correct slot. Let $\mathbf{y}_i$ represent the set of predicted BIO labels for sentence S_i, and $\mathbf{y}_i^{\text{aligned}}$ denote the final aligned labels based on the predicted slot types.

3.1.3 Training Procedure

The training process for both tasks follows a similar approach:

1. **Data Preprocessing:** The sentences are first tokenized into words using the *word_tokenize* function. For slot-filling, annotations are parsed to extract slot types and values.

2. **Embedding Calculation:** Word2Vec is trained on the tokenized training data to generate word embeddings. Each sentence is converted into a fixed-size embedding vector by averaging the embeddings of the individual tokens.

3. **Model Training:** For intent classification, the embeddings are used as input to the classifiers (Logistic Regression and Random Forest), where the objective is to predict the correct intent label. For slot-filling, a separate classifier is trained for each slot type using a One-vs-Rest approach.

4. **Evaluation:** The models are evaluated using accuracy and othe classification metrics, such as precision, recall, and F1 score. Confusion matrices are also plotted to visualize the model's performance.

Both models are trained using standard cross-validation procedures, with data split into training, validation, and test sets (70/15/15).

3.2 Fine-tuning pre-trained models: BERT

BERT (Bidirectional Encoder Representations from Transformers) is designed as a **transformer encoder model** that reads text bidirectionally, capturing the context from both left and right sides of each token. This makes it highly effective for understanding nuanced relationships in the input text.

- The input X is tokenized into subwords $\{x_1, x_2, \ldots x_n\}$ using **WordPiece tokenization**. Each token is embedded as:

  $$\mathbf{E}(x_i) = \mathbf{T}(x_i) + \mathbf{P}(x_i)$$

  where $\mathbf{T}(x_i)$ is the token embedding, and $\mathbf{P}(x_i)$ is the positional embedding that encodes the position i of the token in the sequence.

  The final input sequence becomes:

  $$\mathbf{E} = [\mathbf{E}(\texttt{[CLS]}), \mathbf{E}(x_1), \ldots, \mathbf{E}(x_n), \mathbf{E}(\texttt{[SEP]})]$$

  Transformer Layers: BERT applies multiple transformer layers, where each layer consists of:

  - **Multi-Head Self-Attention:**

    $$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

    Here, $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are projections of the input embeddings $\mathbf{E}$, and d_k is the dimension of the keys.
  - **Feedforward Layers:** Fully connected layers applied to the attention outputs. These layers produce contextualized token embeddings:

    $$\mathbf{H} = [\mathbf{h}_{\texttt{[CLS]}}, \mathbf{h}_1, \ldots, \mathbf{h}_n]$$

- **Fine-Tuning for IC and SL:**
  - **Intent Classification:** The embedding for the special token $\mathbf{h}_{\texttt{[CLS]}}$ is passed through a softmax layer to predict the intent:

    $$\hat{y}_{\text{intent}} = \text{softmax}(\mathbf{W}_{\text{intent}}\mathbf{h}_{\texttt{[CLS]}} + \mathbf{b}_{\text{intent}})$$
  - **Slot-Filling:** Each token embedding $\mathbf{h}_i$ is passed through another softmax layer to predict the slot label:

    $$\hat{y}_{\text{slot},i} = \text{softmax}(\mathbf{W}_{\text{slot}}\mathbf{h}_i + \mathbf{b}_{\text{slot}})$$

3.3 In-context learning without any fine-tuning: Google's Flan-T5-base

Flan-T5-base is a text-to-text transformer model, meaning it treats every NLP task as a sequence-to-sequence problem. Unlike BERT, Flan-T5 includes both an encoder and a decoder, enabling it to generate structured text outputs.

Input Representation: The input sequence

$$X = \{x_1, x_2, \ldots, x_n\}$$

is tokenized similarly and embedded as:

$$\mathbf{E}(x_i) = \mathbf{T}(x_i) + \mathbf{P}(x_i)$$

The task-specific input prompt, e.g., *"Intent and slots for: Book a flight to Paris tomorrow"*, is encoded into embeddings $\mathbf{H}_{\text{enc}}$ by the encoder.

Encoder-Decoder Architecture:

- **Encoder:** Similar to BERT, the encoder applies self-attention and feedforward layers to produce contextualized embeddings: $\mathbf{H}_{\text{enc}}$
- **Decoder:** The decoder generates the output sequence $Y = \{y_1, y_2, \ldots, y_m\}$ autoregressively. For each timestep t, the decoder computes:

  $$\mathbf{h}_{\text{dec},t} = \text{Attention}(\mathbf{Q}_t, \mathbf{H}_{\text{enc}}, \mathbf{H}_{\text{enc}})$$

  where $\mathbf{Q}_t$ is the query projection of the decoder's embeddings.

Output Generation: The decoder predicts each token y_t by applying a softmax over the vocabulary:

$$\hat{y}_t = \text{softmax}(\mathbf{W}_{\text{vocab}}\mathbf{h}_{\text{dec},t} + \mathbf{b}_{\text{vocab}})$$

Joint Modeling of IC and SL: The model generates a structured output combining both tasks. For example:

`Output:` *"**Intent:** BookFlight | **Slots:** destination=Paris, date=tomorrow"*

By training on this format, the model learns to jointly predict intents and slot labels in a single forward pass.

4 Experimental Setup

4.1 Dataset

We use the **NLU Evaluation Data Home Domain Annotated Dataset** for this project. The dataset was build by collecting real user data via Amazon Mechanical Turk (AMT). The robot had to answer questions about how people would interact with it, in a wide range of scenarios designed in advance (*alarm, audio, music, calendar, cooking, datetime, email, game, ...*. For more informations about the build and use of this dataset, please refer to the work of **Xingkun Liu** (cf. Benchmarking Natural Language Understanding Services for building Conversational Agents). The dataset contains annotated data with intents and corresponding slots ('*suggested_entities*') for various user utterances. Below is a summary of key attributes in the dataset:

userid, answerid, scenario, intent, status, answer_annotation, notes, suggested_entities, answer_normalised, answer, question.

Some columns like '*status*' or '*notes*' have a lot of missing values. Other columns such as '*userid*' and '*answer_normalised*' also contain missing values, which are handled during preprocessing. The dataset includes examples of valid utterances, as well as records labeled with statuses like '**IRR**' (*irrelevant*) and '**MOD**' (*modified*), which were filtered to focus on high-quality annotations.

4.2 Data Preprocessing

The following preprocessing steps were applied to clean and prepare the dataset for training:

- Rows with statuses starting with **IRR_** were removed.
- Missing values in critical columns were imputed as follows:
  - ***userid***: Filled with a default value of `1.0`.
  - ***scenario***: Defaulted to `audio`.
  - ***intent***: Defaulted to `volume_mute`.
  - ***answer_normalised*** and ***answer***: Defaulted to `stop`.
  - ***question***: Filled with a generic prompt: `Write what you would tell your PDA in the following scenario`.
- The dataset index was reset to ensure a clean and consistent structure.

All these steps ensured that the dataset was suitable for both intent classification and slot-filling tasks.

4.3 Model and Evaluation Metrics

As a pre-trained sequence-to-sequence transformer model, we used the Flan-T5-base from Google. The input formats were prompt-based setup where each sentence was embedded into a pre-defined template for intent classification, and a different prompt designed for extracting BIO-style slot labels for slot-filling.

We evaluate the model on its ability to accurately classify intents and extract slots using the following metrics:

- **Accuracy:** The proportion of utterances where the predicted intent matches the ground truth.
- **F1-Score:** Measures the harmonic mean of precision and recall for slot prediction.
- **Exact Match Accuracy:** Percentage of sentences where all slots are correctly predicted.

4.4 Implementation Details

- **Pre-trained Models:** The Flan-T5-base model was downloaded from the Hugging Face Model Hub.
- **Data Loading and Preprocessing:** Managed using Pandas for data manipulation and NLTK for natural language processing utilities.
- **Reproducibility:** Seeds were set for both NumPy and PyTorch to ensure consistent results across runs.

5 Results and Analysis

We evaluated the intent classification task using the accuracy metric. The results for each experimental configuration are summarized in Table 1.

Table 1: Intent Classification Results

Model	Accuracy (%)	Time
Logistic Regression	46.63	1min
Random Forest	50.98	8min
BERT	75.45	30min
Flan-T5-base	19.11	5h

Observations: The Flan-T5-base model should outperforme the baseline in terms of accuracy, but the results were really, if not extremely bad. The baseline model required significantly less training time, highlighting the trade-off between accuracy and computational cost. The inference time per utterance for Flan-T5-base was significantly higher, likely due to the autoregressive nature of the model.

Table 2: BERT Results for IC

Epoch	Training Loss	Accuracy (%)
1/3	1.3756	70.97
2/3	0.7467	73.25
3/3	0.6061	74.01

We also tried to predict **scenarios** and using a Logistic Regression we obtained an accuracy of 69.19%, pointing out that considering all the words in the sentence, the general scenario is easier to predict than the intents.

2. Slot-Filling Results For the slot-filling task, we evaluated the performance using exact match accuracy. The results are summarized in Table 3.

Table 3: Slot-Filling Results

Model	Accuracy (%)	Time
Logistic Regression	48.89	1min
Random Forest	53.03	8min
BERT	92.99	30min
Flan-T5-base	00.00	5h

Table 4: BERT Results for SL

Epoch	Training Loss	Accuracy (%)
1/3	0.6222	91.25
2/3	0.2636	92.17
3/3	0.1618	92.73

Observations: Flan-T5-base achieved the lowest accuracy, suggesting that the model occasionally struggles with complex utterances containing multiple slots, or that we made a mistake somewhere. But the BERT model showed us the best results, with the highest accuracy between all of them.

3. Error Analysis To better understand the model's weaknesses, we conducted a qualitative error analysis. Some of the key insights are:

Ambiguity in User Utterances: Utterances with ambiguous phrasing or unclear context were often misclassified. Rare Intents and Slots: The model underperformed on intents and slots that had fewer training examples, highlighting the need for more balanced datasets. Correlation Between Slots and Intents: In some cases, incorrect slot predictions led to incorrect intent classifications, indicating that the model might not be fully leveraging the relationship between these tasks

4. Hypotheses and Discussion Based on the results and error analysis, we hypothesize the following:

Hypothesis 1: The model's performance could be further improved by fine-tuning on a larger dataset or augmenting the existing dataset with synthetic examples for rare intents and slots (at least for the classifiers) Hypothesis 2: Flan-T5-base may require a more specialized prompt design to better capture complex relationships between intents and slots. Hypothesis 3: The observed inference latency may be mitigated by optimizing the model using techniques like quantization or distillation.

6 Conclusion

In this report, we explored the performance of various models for intent classification (IC) and slot-filling (SF) tasks. We evaluated Logistic Regression, Random Forest, BERT, and Flan-T5-base models, each with distinct strengths and weaknesses, in terms of accuracy, training time, and inference latency.

For the intent classification task, we observed that while BERT achieved the highest accuracy (75.45%), the computational cost, particularly for the Flan-T5-base model, was considerably high. The Flan-T5-base model, despite its promising architecture, underperformed significantly in terms of accuracy (19.11%), with high inference latency. This highlights the trade-off between accuracy and computational efficiency, a common challenge when working with large language models. In contrast, baseline models like Logistic Regression performed quicker but with significantly lower accuracy, demonstrating that accuracy gains often come at the cost of higher computational demands.

Similarly, for the slot-filling task, BERT again showed the best performance with an accuracy of 92.99%, while Flan-T5-base achieved the lowest accuracy (0.00%). This reinforces the importance of model choice and tuning, as some models may struggle with tasks involving complex relationships or multiple slots, as seen in the Flan-T5-base results. The low accuracy of Flan-T5-base in both tasks suggests a need for further fine-tuning or prompt engineering to better capture the intricate connections between intents and slots.

Our error analysis also revealed that ambiguity in user utterances and a lack of sufficient training data for rare intents and slots were key factors limiting model performance. Furthermore, some issues arose from misalignments between intent and slot predictions, suggesting that a more integrated approach could be beneficial for improving both tasks simultaneously.

Overall, we hypothesize that the performance of the models could be enhanced through strategies such as dataset augmentation, fine-tuning on larger datasets, and optimizing the inference process. In particular, exploring quantization and distillation techniques could help mitigate the high computational costs of large models like Flan-T5-base, making them more feasible for practical use in real-world applications.