Conversational agents for simulation applications and video games

Ciprian Paduraru, Marina Cernat, and Alin Stefanescu
Department of Computer Science, University of Bucharest, Romania
ciprian.paduraru@unibuc.ro, marina.cernat@unibuc.ro, alin.stefanescu@unibuc.ro

Keywords: Natural Language Processing, video games, active assistance, simulation applications

Abstract: Natural language processing (NLP) applications are becoming increasingly popular today, largely due to recent advances in theory (machine learning and knowledge representation) and the computational power required to train and store large language models and data. Since NLP applications such as Alexa, Google Assistant, Cortana, Siri, and chatGPT are widely used today, we assume that video games and simulation applications can successfully integrate NLP components into various use cases. The main goal of this paper is to show that natural language processing solutions can be used to improve user experience and make simulation more enjoyable. In this paper, we propose a set of methods along with a proven implemented framework that uses a hierarchical NLP model to create virtual characters (visible or invisible) in the environment that respond to and collaborate with the user to improve their experience. Our motivation stems from the observation that in many situations, feedback from a human user during the simulation can be used efficiently to help the user solve puzzles in real time, make suggestions, and adjust things like difficulty or even performance-related settings. Our implementation is open source, reusable, and built as a plugin in a publicly available game engine, the Unreal Engine. Our evaluation and demos, as well as feedback from industry partners, suggest that the proposed methods could be useful to the game development industry.

1 INTRODUCTION

NLP is one of the most important components in the development of human-machine interaction. It studies how computers can be programmed to analyze and process a large amount of natural language data. Some of the most challenging topics in natural language processing are speech recognition, natural language generation, and natural language understanding. According to literature and surveys, simulation applications and video games are one of the most valuable sectors in the entertainment industry [Baltezarevic et al., 2018]. Our main contribution to this area is to work with our industry partners to identify and develop methods based on modern NLP techniques that could potentially help the simulation software and video games industry. The main problem we are addressing is how to help the player and respond live to his feedback. The person or entity assisting the user can be rendered virtually in the simulation environment, i.e., as a non-playable character (NPC) or as a narrator/environmental listener. Together with industry partners, several specific use cases of NLP in simulation applications and game development industry were investigated. The first category investigated relates to classical *sentiment analysis* [Wankhade et al., 2022] problem. An example of this is when a user is playing with an NPC and indicates, either through speech or text, that the difficulty level of the simulation has some problems (e.g., too difficult or not challenging enough). In this case, one solution would be to dynamically adjust the difficulty level to provide an engaging experience for the user. Another concrete case: imagine the user is playing a soccer game such as FIFA[1], and is disappointed by a referee's decision. The decision could then attract different unusual noises made by the user. The in-game referee could then react accordingly and make the feedback more severe or even penalize the user's behavior.

Moreover, we found that NLP techniques can be used to create NPC companions that physically appear in the simulated environment and can be prompted by a voice or text command to help the user. Concrete situations could be that the user is blocked at an objective or cannot find the way to a needed location. In this case, the NPC companion could understand the user's request and guide them to a desired location or provide hints to solve various puzzles.

[1] https://www.ea.com/en-gb/games/fifa

Another source of frustration for users when interacting with simulated environments is that they cannot understand various designed mechanics. This problem is common in the industry and causes users to abandon the applications before developers can generate any revenue or enough game satisfaction. In this case, we see NLP as a potential solution where the user can ask questions that a chatbot can answer live to help them. Concrete examples could be questions about healing mechanisms, finding different items on a given area, things needed or missing to achieve certain goals, etc.

The novelty of our work is that it uses modern NLP techniques to address the above mentioned problems. We summarize our contributions below.

- Identify concrete problems in simulation applications and video games that can be solved with NLP techniques, together with industry partners.
- A reusable and extensible open source framework to help simulation and game developers add NLP support to their products. The solution we provide at https://github.com/AGAPIA/NLPForVideoGames is designed as a plugin for a game engine commonly used in both industry and academia, the Unreal Engine[2]. For the basic part, we use very recent deep learning methods from the NLP literature, which are suitable for real-time inference as needed for today's applications. User input can be either in the form of voice or text messages. For evaluation purposes, a demo is built on top of the framework. Along the repository, an example from the demo application can be found on voice commands at https://youtu.be/Z0JqyTO724M, while for text commands at https://youtu.be/v2Ls9pboXxc.
- Identify some best practices and specifics for applying NLP to simulation software and video games.

The rest of the paper is organized as follows. Section 2 presents use cases of NLP application in various other industries that provide similar solutions in a context different from ours. Section 3 presents the theoretical and technical methods we propose within our framework for using NLP in video games and simulation applications. The evaluation from a quantitative and qualitative perspective, along with details about our setup, observations, and datasets are presented in Section 4. The final section presents our conclusions and ideas for future work.

[2]https://www.unrealengine.com

2 RELATED WORK

As described in the review paper [Allouch et al., 2021], Conversational Agents (CA) are used in many fields such as medicine, military, online shopping, etc. These agents are usually virtual agents that attempt to engage in conversation with interested humans and answer their questions, at least until they receive information from them, which is then passed on to real human agents. To the authors' knowledge, however, there is no previous work that uses modern NLP techniques to achieve the goals we seek in the area of simulation applications and/or game development. While the literature for the use of NLP for the defined purpose of this project is new, we establish our foundations by following or referencing existing work in the literature and transferring, as much as possible, the methods used by CA from other domains.

An overview of the use of CAs in healthcare is presented in [Laranjo et al., 2018]. The work in [Dingler et al., 2021] explores CAs for digital health that are capable of delivering health care at home. Voice assistants are built with context-aware capabilities to provide health-specific services. In [Sezgin and D'Arcy, 2022], the authors explore how CAs can help collect data and improve individual well-being and healthy lifestyles. The use of CAs has also been applied to specific topics such as assessing and improving individuals' mental health [Sedlakova and Trachsel, 2022] and substance use disorders [Ogilvie et al., 2022].

Another interesting use case of CAs is in emergency situations, as explored in [Stefan et al., 2022], which describes the requirements and design of appropriate agents in different contexts. An adjacent remark is made by [Schuetzler et al., 2018], which shows that people are more willing to discuss with virtual CAs, especially sensitive information, than with real human personnel.

CAs have also been used for gamification purposes. In [Yunanto et al., 2019], the authors propose an educational game called *Turtle trainer* that uses an NLP approach for its non-playable characters (NPCs). In the game, NPCs can automatically answer questions posed by other users in English. Human users can compete against NPCs, and the winner of a round is the one who answered the most questions correctly. While their methods for understanding and answering questions are based on classical NLP methods, we follow their strategy of evaluation based on qualitative feedback from two perspectives: (a) How do human users feel about how well their competitors, i.e., the NPCs, understand and answer the questions, (b) Does the presence of NPCs in this form repre-

sent a greater interest for the learning game itself? A common platform for teaching different languages is *Duolingo*. With its support, various learning games, e.g., language learning through gamification [Munday, 2017], and machine learning-based methods for performance testing are developed. Using the platform itself and NLP methods, the authors proposed the use of automatically generated language tests that can be graded and psychometrically analyzed without human effort or supervision.

3 METHODS

3.1 Overview

The inference process starts with a user message and ends with an application-designated component that resolves the query made at game runtime, as shown in Figure 1. The first part is to detect the type of message and translate it into a text format. Input from the user side can be both voice and entered text messages. For speech recognition and synthesis, a locally instantiated model based on the DeepPavlov suite [Burtsev et al., 2018] is used. The output is then sent to a preprocessing component that uses classic NLP operations required for other modules, such as removing punctuation, making the text lowercase, adding beginning and ending markers for sentences, etc., as detailed in BERT model [Devlin et al., 2018] and explained below in this section. Spelling mistakes and abbreviations for the text are also taken into account for the BERT input [Hu et al., 2020].

3.2 Foundational NLP models

We first motivate BERT [Devlin et al., 2018] instead of GPT [Brown et al., 2020] model as the core of our methods. In short, both models are based on Transformer architectures [Vaswani et al., 2017] trained on large plaintext corpora with different pre-training strategies that the reader can explore in the recommended literature. Probably the fundamental architectural difference between the two is that in a transformer's mindset, BERT is the encoder part, while GPT-3 is the decoder. Therefore, BERT can be more easily used to refine its encoding and further learn by appending output layers to produce desired customized functions and categories. This architectural implication of BERT makes it suitable for domain specific problems, which is also our target case. Each application has its own data, text, locations, characters, etc., and the underlying NLP model has to adapt to these individual use cases. This point of view is also supported by the work in RoBERTa [Liu et al., 2019], which fine-tuned BERT to learn a new language data set, confirming that the model is better suited for downstream tasks. Since BERT is pretrained on large scale text using two strategies i.e. masked language model and next sentence, it can provide contextual features at sentence level. With this benefit and the fact that it can be easily customized for custom datasets, the base model chosen can be used for tasks like intent classification and slot filling, two of the main tasks we need for our goal.

3.3 Intent classification and slot filling

Intent classification is the assignment of a text to a particular purpose. A classifier analyzes the given text and categorizes it into intentions of the user. Filling slots is a common design pattern in language design, usually used to avoid asking too many or detailed questions to the user. These slots represent a goal, starting point, or other piece of information that needs to be extracted from a text. A shared model performs multiple tasks simultaneously. Because of its structure, the intent classification model can easily be extended to a shared model for classifying intentions and filling slots [Chen et al., 2019b].

To understand a user's message, a model is used that processes the data and provides two outputs: (a) the *intent* of the message, (b) the *slots* that represent parts of the text that can be semantically categorized and make connections with the application's knowledge base. To this end, the framework relies primarily on the methods described by [Chen et al., 2019b], which provide a model for computing the two required outcomes simultaneously. One feature we have identified as useful, at least from a real-time simulation perspective, is replacing the WordPiece tokenizer from the original model with a faster version with tokenization complexity O(n) [Song et al., 2021].

This task corresponds to the *SemanticExtraction* block from Figure 1 and is described in the following text with examples and the connection to our current implementation.

The intent. Describes the category of the message. This is formalized as a set, $I = \{I_1, I_1, \ldots, I_{ni}\}$. This can be customized depending on the needs of the developer. For illustration, three types of concrete intentions are used in our current demo attached to the repository.

- *SentimentAnalysis*
  The message sent by the user is considered as general live feedback to the application, e.g. too difficult or not challenging enough, graph too low. In return, the application should take appropriate

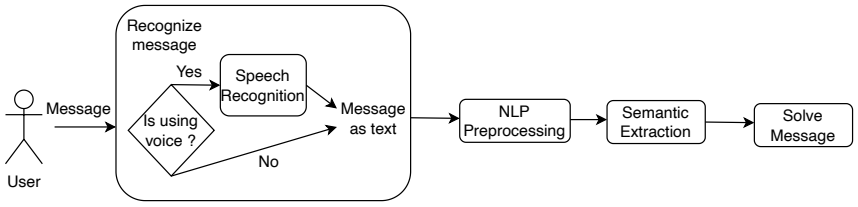

Figure 1: Overview of data flow from user message to execution of in-application requests. In the first part, the message is always converted into a textual representation. Then a series of preprocessing operations are performed so that the output text is compatible with the needs of the next component in the flow. The SemanticExtraction component extracts the intent of the message and makes the semantic connections with the application's custom database to understand what actions are required to provide feedback on the user message. Details on each individual component are presented along Section 3.

action.

- *AnswerQuestion*
  This category includes user questions related to the application. Some common examples are questions about mechanisms that are not clear to the user. This is a way for the user to request help with common things, which can increase their engagement with the application.
- *DoActionRequest*
  In general, this category is reserved for messages addressed to companion NPCs, asking the user for help in solving various tasks. Typical examples in simulation applications and games include asking for help in solving a puzzle or challenging a character, pointing the way to a certain location, or assisting the user with various actions. This kind of category is usually divided into different sub-actions, e.g. in our demo into three sub-actions: *FollowAction* - help in finding a region or place, i.e. the companion NPC shows the way (see our supplementary video examples in the suggested links in Section 1), *EliminateEnemy* - ask the NPC to eliminate a particular enemy NPC or user, and *HealSupport* - ask the NPC to do whatever it takes to improve the user's health in the virtual simulated environment.

The slots. Describes the slots in the message that are important for understanding their connections to the simulation environment. For each identified slot, its category type and corresponding values are output from the message. The slot types form a set derived from the Inside-Outside-Beginning (IOB or BOI in the literature) methodology [Zhang et al., 2019]. In this method, B marks the beginning of a sequence of words belonging to a particular slot category, I marks the continuation of words in the same slot, and O marks a word that is outside an identified slot category. Each of the words in the message is assigned one of these three markers. Examples can be found in Listing 1. The set of slots is further formalized as $S = \{S_1, S_1, \ldots, S_{ns}\}$. Note that different intentions and multiple semantic parts of different B-types may occur in the same sentence (especially in longer sentences). In this case, the sentence is split into simpler sentences as much as possible, so that each sentence has a different intention. Each of the propositions/intentions and their associated slots are passed along as different messages.

```
Example 1:
-Player input text: "Show me where the potion shop is"
-Intent: FollowAction
-Slots: [O O O O B-location I-location O]

Example 2:
-Player input text: "Find me anything to get my health fixed cause I will be over soon".
-Intent: HealSupport
-Slots: [O O O O B-heal I-heal I-heal I-heal O O O O O O]
```

Listing 1: Examples of two user inputs and their classified intents and slots

The classification of intentions and slots follows the work in [Song et al., 2021]. We briefly describe the classification methods and mechanisms behind it. First, as required for the BERT [Devlin et al., 2018] model, the user-input text message is tagged with markers indicating the beginning of a question (*[CLS]*) and the end of each sentence *[SEP]*. We denote the input sequence as $x = (x_1, \ldots, x_T)$, where T is the sequence length. This is then processed by BERT, Figure 2, which outputs a hidden state representation for each input token, i.e. $H = (h_1, \ldots, h_T)$. Given that this is a bidirectional model and the fact that the first token is always the injected $[CLS]$ marker, the intent of the question can be determined by calculating the probability of each intent category $i \in I$ according to the formula in Eq. 1 can be determined. This categorical distribution can then be sampled to determine the intent of each question at runtime.

$$y^i = \text{softmax}\left(\mathbf{W}^i h_1 + b^i\right) \tag{1}$$

For slot prediction, all hidden states must be con-

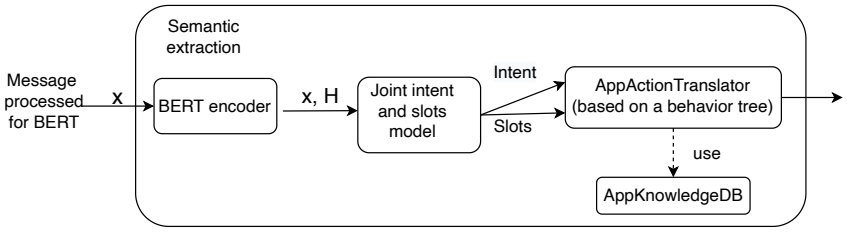

Figure 2: The SemanticExtraction from Figure 1 component in detail. The input is a preprocessed sequence x of tokens, which is encoded into a sequence of hidden states by the model BERT [Devlin et al., 2018]. Both inputs are passed to the joint classification model for intention and slots [Chen et al., 2019b]. A semantic correlation is then made between the database knowledge provided by the developer, specific to each application, and the identified slots. The result of this correlation is a tree of tasks that must be solved by the application in order to provide appropriate feedback. The subcomponents are explained in more detail in Section 3.3

sidered when they are fed into the softmax layer so that the distribution can be calculated as shown in Eq. 2.

$$y_n^s = \text{softmax}\left(\mathbf{W}^s h_n + b^s\right), \;\; n \in 1 \ldots ns \qquad (2)$$

The goal of the training phase is to maximize the conditional probability of correct classification of joint slots and intent (Eq. 3), in a supervised manner using the database of application knowledge created as mentioned in Section 4. The model BERT is also fine-tuned to this objective function by minimizing cross-entropy loss.

$$p\left(y^i, y^s \mid x\right) = p\left(y^i \mid x\right) \prod_{n=1}^{ns} p\left(y_n^s \mid x\right) \qquad (3)$$

The proposed framework architecture provides a way to bring in custom knowledge bases for applications that are generally specific between developers and titles. This is also shown in Figure 2, where the *AppKnowledgeDB* component is used for this purpose by matching identified slot categories, texts, and actions that need to be performed for the application. The component that semantically transitions from slot information to actions to be performed in the application is called *AppActionTranslator*. Its implementation in our current framework is based on a behavior tree [Paduraru and Paduraru, 2019], which can be adapted or extended with various methods by any application. The method used on the game side can be as simple as an expert system judging by the intents, slots and the strings attached to them that relate to the game units and actions, but behavior trees were preferred by default to have a better semantic description. At the lowest level, a Levenshtein distance [Miller et al., 2009] is used for string comparisons to identify the application's slot data and knowledge base. When processing the inputs, the model assigns a probability of match between the application knowledge and the identified slots. If the threshold is not above a parameter set by the developer, the framework provides feedback that the message was not understood. In the current implementation, a threshold of 0.5 was used, and the identification score was based on Levenshtein distance and averaged across slots.

3.4 Question-response model

To answer user questions, the models and tools available in the DeepPavlopv library[3], called *Knowledge Base Question Answering*, are adapted to our requirements and reused. In short, the strategy used by the tools is to provide a ready-to-use general question answering model based on the large Wikidata corpus [Vrandečić and Krötzsch, 2014]. Then, the library allows the insertion of custom knowledge representations to enable custom retraining/fine-tuning of the existing models. At the core of the methods, the same BERT model is reused to perform the following operations:

- Recognize a query template from the user message. Currently there are 8 categories in our framework. This is also extensible on the developer side. The model behind it is based on BERT.
- detection of entities in the message and their associated strings. Recognized entities are linked to the entities contained in the knowledge base. Top-k matches are evaluated based on Levenshtein distance and stored. The same BERT model is behind the operation.
- ranks the possible relationships and paths in the set of recognized and linked entities. Thus, this step provides an evaluation of the valid combinations of entities and semantic relations. For this step, modified from the original version for

[3]https://docs.deeppavlov.ai/en/master/features/models/kbqa.html

our needs of better linking of entities and a user-defined knowledge base, we use the methods of BERT-ER [Chatterjee and Dietz, 2022].

- A generator model, based at its core on BERT, is used to populate the answer query templates. The slots are filled with the candidate entities and relations found in the previous steps.

4 EVALUATION

This section presents the design used for the evaluation, the creation of the dataset used for the training, the results of the quantitative and qualitative evaluation using automatic and human perception tests, and finally performance-related aspects to demonstrate the usefulness of the proposed methods.

4.1 Experimental design and discussion

The framework implementation and the demo based on it work as a plugin in both Unreal Engine versions 4 and 5 and are available at https://github.com/AGAPIA/NLPForVideoGames. Examples from the demo can be viewed in our repository and Figures 3, 4, and 5. The architectural decision to use a plugin was made to achieve separation of concerns and minimal dependencies between the application implementation and the proposed framework. During development and experimentation, Flask[4] was used to quickly switch between different models and parts, parameters used for training or inference, etc. In the production phase and in our evaluation phase, the models were deployed locally and accessed through Python bindings to C++ code, since the engine source code specific to the application implementation was written in C++.

In our evaluation,we used the BERT-Base English uncased model[5], which has 12 layers, 768 hidden states, and 12 multi-attention heads. To fine-tune the model for the common goal of classification and slot filling, the maximum sequence length was set to 30 and the batch size to 128. Adam [Kingma and Ba, 2014] with an initial learning rate of 5e-4 is preferred as the optimizer. The dropout probability was set to 0.1. On our custom dataset provided in the demo, the model was then fine-tuned for approximately 1000 epochs. The threshold for confidence is set to 0.5 in our evaluation, meaning that the output of the behavior tree must be deemed correct with a probability of at least 0.5, otherwise the feedback from the application side is that the user's message is not understandable.

Figure 3: Demo capture of the user requesting details of self-healing either by speech or text, and then asking the NPC companion to help them find the healing location. Note that for speech input for debugging purposes, the understood message is also displayed as text

.

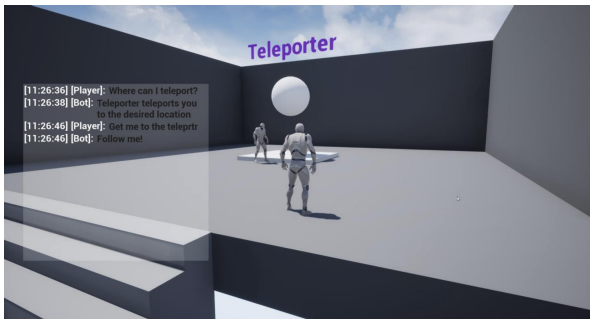

Figure 4: A similar message exchange as in Figure 3, this time with the request for the teleport action

.

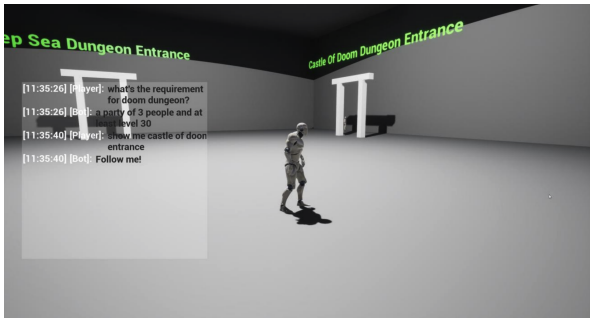

Figure 5: The user asks for details about some quests he can play with the current level, and then asks the NPC to accompany them there.

.

Fine-tuning and transfer learning. As described in Sections 3.3 and 3.4, the methods proposed in our work are divided into several small modules. Fine-tuning by reusing parts of the existing state-of-the-art model architecture and pre-trained parameters on publicly available datasets helps considerably in

[4]https://flask.palletsprojects.com/en/2.2.x
[5]https://github.com/google-research/bert

answering questions. Without most of the knowledge coming from large scale corpuses, the text answers generated by the application looked unnatural. Therefore, little fine-tuning is required for question-answering models or entity linking models to perform well. On the other hand, more fine-tuning is required for the models used to classify messages in query masks or to identify the entities in the messages. This is necessary because the semantic information for understanding user messages must come primarily from the application knowledge bases, rather than from Wikidata or other public datasets that were used to pretrain the models.

Hierarchical Contexts. We decided to add another layer for both identifying intentions/slots and for the models to answer questions. This came out of our evaluation process, where we found that there were a few different contexts in the application that could improve feedback results. A specific example from our demo: when the user is in the shopping area in the simulated environment, a context variable is set to *shopping*. This information was provided in the training dataset and considered as input to each of the models during training or fine-tuning. The additional information is helpful in understanding simple questions that the user may have difficulty understanding, e.g., they might ask for *what would you recommend I buy from here?* The answer to this question could vary depending on the context, e.g. shopping opportunities, locations of craft items, a simple map section with items that can be collected, etc.

4.2 Datasets used

The dataset was created by 15 different volunteer students from University of Bucharest during their undergraduate studies. We use the 80/20 rule for splitting training and evaluation datasets. To train the models containing slots and intentions on our demo, the dataset of examples looks like the examples in Listing 1. In total, there were 1500 pair examples, split between 8 contexts, and almost the same number for each of the three classes (SentimentAnalysis, AnswerQuestion, and DoActionRequest).

For the AnswerQuestion category, each example also contained a sequence of words that described the desired answer as text from the annotator's perspective. For the other two categories, the example contained the specific application action to be performed in the demo application with numeric IDs between 1 to 5. Specifically, we had two actions in our demo for the SentimentAnalysis case: change the difficulty level and show instructions for performing various contextual operations. For the DoActionRequest, we use the three subcategories mentioned in Section 3.3: FollowAction, EliminateEnemy, HealSupport.

As an aside, participants were asked to include typos and abbreviations in their texts so that the models could adapt to them as well.

4.3 Quantitative evaluation

The quantitative assessment of the joint model is based on the two outputs of the model, i.e., intentions and slot identification. For intentions, common metrics such as precision, recall, and F1 score are used, while for slots, only the F1 score is useful due to the highly imbalanced data. As Table 1 shows, the intent classification model manages to give the correct answers in most cases, with an overall lower scoring value for the DoActionRequest, since there is a more diverse set of actions in the training set.

For the slot classification, where the models were re-trained to better fit the application knowledge database, as explained above, an F1 score of 89.621% is obtained.

The metric METEOR [Banerjee and Lavie, 2005] is used to score the question response. Since this metric comes from the field of machine translation, it can also be easily adapted for question answer scoring. Briefly, the idea behind it is to match the sequence of words in the (correct) target answer with the predicted answer, taking into account stemming and synonymy matching between words. The score is formed from a harmonic mean of precision and recall, with more weight given to recall. In addition, the method penalises answers that do not consist of consecutive paragraphs in the target text, as this is natural to the human comprehension process. In our dataset of AnswerQuestionCategory, the METEOR metric scored 0.7095, which is a reliable value that answers look almost natural according to the literature [Chen et al., 2019a].

4.4 Qualitative evaluation

The experiments and statistical results evaluating our framework and demo from a qualitative point of view come from a group of 12 people (volunteers from the quality assurance departments of our industry partners and students from the University of Bucharest) who played the demo for two hours and tried to move through the application asking questions to the NPCs about different topics. There are three research questions that we explored during our qualitative evaluation.

Table 1: Quantitative evaluation of the classification model for predicting intentions.

Metric	Category		
	AnswerQuestion	SentimentAnalysis	DoActionRequest
Precision	0.91744841	0.89811321	0.95881007
Recall	0.978	0.952	0.838
F1-score	0.94675702	0.92427184	0.89434365

RQ1. Do the application or NPCs provide correct feedback, i.e., do they seem to correctly understand the questions or requests asked and respond correctly in the context?

To assess this, each of the 12 participants played the demo for 2 hours and were asked to give 100 messages to the application/NPCs, with equal amounts of input via voice and text. After each response from the application (including a 1 minute delay for action types such as path tracking or other user support operations that cannot be evaluated immediately), they were asked whether the NPCs or the application responded correctly.

Table 2 shows the averaged feedback from participants for both types of input, broken down by category of message. The results show that users are generally satisfied with the feedback, with lower ratings for speech input, as expected, due to the need for another layer to convert speech to text and, of course, performance degradation along the way. Scores are also lower in the DoActionRequest category, as it is more difficult in two ways: (a) to correctly understand the requested action, (b) to execute it through application mechanisms, which also implies using the behaviour trees in the AppActionTranslator component, as shown in Figure 2. This can be further improved by providing a larger dataset of such action requests and dividing them into a larger set of subcomponents and/or categories.

RQ2. Is the method suitable for real-time use? When run on a separate thread on a CPU -only, Core i7 gen11 processor, the average time for a full pipeline inference was 13.24 milliseconds for user input through a text message. For a voice input, the time to process and convert the voice to text (Figure 1) averaged 1.96 seconds for sentences of about 15 words (Figure 1). This proves that the method can be used in real-time and without bottlenecks on the simulation side when running on a separate thread. The time needed to understand the question can be disguised as a natural process anyway, since humans also need to understand, process, and formulate an answer, rather than expecting it in the same frame when it is asked.

RQ3. Do users feel that conversing with the application or NPCs helps them in general (e.g., better understanding of the mechanics of the simulation environment, removal of obstacles, assistance with difficult puzzles or actions, etc.)?

To assess this question, a response template was given to each participant at the end of the test. Overall, 10 out of 12 participants felt that the discussions helped them during the demo, while 2 of them wished they could discover the application and mechanisms themselves. In contrast to the correctness results shown in Table 2, users felt that the DoActionRequest category was the most useful, as it really helped them get through difficult parts of the demo.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose a set of methods and tools for applying NLP techniques to simulation applications and video games. We provide an open source software that can be used for both academic and industrial evaluation purposes. The requirements, dataset creation, and evaluation of the results have been created in collaboration with industry partners that publish software in the relevant field. From our point of view, this contributes significantly to understanding and addressing the right problems and then evaluating how the proposed methods correctly solve the requirements and existing problems. Both the quantitative and qualitative evaluation show that the use of NLP as an active assistant for the user within the simulation software can increase user satisfaction and improve user engagement. Our ideas for future work include first introducing an active learning methodology where users can provide live feedback and improve the models online, rather than just training them on collected datasets in a supervised manner. This could greatly improve the models because once deployed, a large amount of data and users could interact with the models and tell the algorithms where they went wrong in providing feedback or understanding intent. Another planned research topic is to create an ontology for simulation software and

Table 2: Qualitative evaluation of the correctness of the feedback depending on the type of input and the respective category of the message.

Input type	**Category**		
	AnswerQuestion	SentimentAnalysis	DoActionRequest
Text	91%	87%	79%
Voice	88%	85%	71%

games in general, similar to the ontology used in web development, so that common terms and notations can be used. This would facilitate the adoption and reuse of the tools in multiple projects.

Acknowledgments

This research was supported by European Union's Horizon Europe research and innovation programme under grant agreement no. 101070455, project DYNABIC. We also thank our game development industry partners from Amber, Ubisoft, and Electronic Arts for their feedback.

REFERENCES

Allouch, M., Azaria, A., and Azoulay, R. (2021). Conversational agents: Goals, technologies, vision and challenges. *Sensors*, 21(24).

Baltezarevic, R., Baltezarevic, B., and Baltezarevic, V. (2018). The video gaming industry (from play to revenue). *International Review*, pages 71–76.

Banerjee, S. and Lavie, A. (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Burtsev, M., Seliverstov, A., Airapetyan, R., Arkhipov, M., Baymurzina, D., Bushkov, N., Gureenkova, O., Khakhulin, T., Kuratov, Y., Kuznetsov, D., Litinsky, A., Logacheva, V., Lymar, A., Malykh, V., Petrov, M., Polulyakh, V., Pugachev, L., Sorokin, A., Vikhreva, M., and Zaynutdinov, M. (2018). DeepPavlov: Open-source library for dialogue systems. In *Proceedings of ACL 2018, System Demonstrations*, pages 122–127, Melbourne, Australia. Association for Computational Linguistics.

Chatterjee, S. and Dietz, L. (2022). Bert-er: Query-specific bert entity representations for entity ranking. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1466–1477.

Chen, A., Stanovsky, G., Singh, S., and Gardner, M. (2019a). Evaluating question answering evaluation. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 119–124, Hong Kong, China. Association for Computational Linguistics.

Chen, Q., Zhuo, Z., and Wang, W. (2019b). Bert for joint intent classification and slot filling. *ArXiv*, abs/1902.10909.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dingler, T., Kwasnicka, D., Wei, J., Gong, E., and Oldenburg, B. (2021). The use and promise of conversational agents in digital health. *Yearbook of Medical Informatics*, 30:191–199.

Hu, Y., Jing, X., Ko, Y., and Rayz, J. T. (2020). Misspelling correction with pre-trained contextual language model. *2020 IEEE 19th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, pages 144–149.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Laranjo, L., Dunn, A. G., Tong, H. L., Kocaballi, A. B., Chen, J., Bashir, R., Surian, D., Gallego, B., Magrabi, F., Lau, A. Y. S., and Coiera, E. (2018). Conversational agents in healthcare: a systematic review. *Journal of the American Medical Informatics Association*, 25(9):1248–1258.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Miller, F. P., Vandome, A. F., and McBrewster, J. (2009). *Levenshtein Distance: Information Theory, Computer Science, String (Computer Science), String Metric, Damerau Levenshtein Distance, Spell Checker, Hamming Distance*. Alpha Press.

Munday, P. (2017). Duolingo. gamified learning through translation. *Journal of Spanish Language Teaching*, 4(2):194–198.

Ogilvie, L., Prescott, J., and Carson, J. (2022). The use of chatbots as supportive agents for people seeking help with substance use disorder: A systematic review. *European Addiction Research*, 28(6):405–418.

Paduraru, C. and Paduraru, M. (2019). Automatic difficulty management and testing in games using a framework based on behavior trees and genetic algorithms.

Schuetzler, R. M., Giboney, J. S., Grimes, G. M., and Nunamaker, J. F. (2018). The influence of conversational agent embodiment and conversational relevance on socially desirable responding. *Decision Support Systems*, 114:94–102.

Sedlakova, J. and Trachsel, M. (2022). Conversational artificial intelligence in psychotherapy: A new therapeutic tool or agent? *The American Journal of Bioethics*, 0(0):1–10. PMID: 35362368.

Sezgin, E. and D'Arcy, S. (2022). Editorial: Voice technology and conversational agents in health care delivery. *Frontiers in Public Health*, 10.

Song, X., Salcianu, A., Song, Y., Dopson, D., and Zhou, D. (2021). Fast WordPiece tokenization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2089–2103. Association for Computational Linguistics.

Stefan, S., Lennart, H., Felix, B., Christian, E., Milad, M., and Björn, R. (2022). Design principles for conversational agents to support emergency management agencies. *International Journal of Information Management*, 63:102469.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A free collaborative knowledge base. *Communications of the ACM*, 57:78–85.

Wankhade, M., Rao, A., and Kulkarni, C. (2022). A survey on sentiment analysis methods, applications, and challenges. *Artificial Intelligence Review*, 55:1–50.

Yunanto, A. A., Herumurti, D., Rochimah, S., and Kuswardayan, I. (2019). English education game using non-player character based on natural language processing. *Procedia Computer Science*, 161:502–508. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

Zhang, F., Fleyeh, H., Wang, X., and Lu, M. (2019). Construction site accident analysis using text mining and natural language processing techniques. *Automation in Construction*, 99:238–248.