applied sciences

Article

A Domain-Specific Generative Chatbot Trained from Little Data

Jurgita Kapočiūtė-Dzikienė [1,2]

[1] JSC Tilde Information Technology, Naugarduko str. 100, LT-03160 Vilnius, Lithuania; jurgita.kapociute-dzikiene@vdu.lt

[2] Department of Applied Informatics, Vytautas Magnus University, Vileikos str. 8, LT-44404 Kaunas, Lithuania

Received: 20 February 2020; Accepted: 16 March 2020; Published: 25 March 2020

Abstract: Accurate generative chatbots are usually trained on large datasets of question–answer pairs. Despite such datasets not existing for some languages, it does not reduce the need for companies to have chatbot technology in their websites. However, companies usually own small domain-specific datasets (at least in the form of an FAQ) about their products, services, or used technologies. In this research, we seek effective solutions to create generative seq2seq-based chatbots from very small data. Since experiments are carried out in English and morphologically complex Lithuanian languages, we have an opportunity to compare results for languages with very different characteristics. We experimentally explore three encoder–decoder LSTM-based approaches (simple LSTM, stacked LSTM, and BiLSTM), three word embedding types (one-hot encoding, fastText, and BERT embeddings), and five encoder–decoder architectures based on different encoder and decoder vectorization units. Furthermore, all offered approaches are applied to the pre-processed datasets with removed and separated punctuation. The experimental investigation revealed the advantages of the stacked LSTM and BiLSTM encoder architectures and BERT embedding vectorization (especially for the encoder). The best achieved BLUE on English/Lithuanian datasets with removed and separated punctuation was ~0.513/~0.505 and ~0.488/~0.439, respectively. Better results were achieved with the English language, because generating different inflection forms for the morphologically complex Lithuanian is a harder task. The BLUE scores fell into the range defining the quality of the generated answers as good or very good for both languages. This research was performed with very small datasets having little variety in covered topics, which makes this research not only more difficult, but also more interesting. Moreover, to our knowledge, it is the first attempt to train generative chatbots for a morphologically complex language.

Keywords: generative chatbot; encoder–decoder architecture; LSTM; BiLSTM; fastText and BERT word embeddings

1. Introduction

Modern society is not imaginable without personal virtual assistants and conversational chatbots such as, e.g., Siri, Alexa, Google assistant, Cortana; they are completely changing our communication habits and interactions with technology. Intelligent chatbots are fast and can substitute some human functions; however, artificial intelligence (AI) and natural language processing (NLP) technologies are still limited with respect to replacing humans completely.

The first chatbot ELIZA [1] invented in 1966 was based on keyword recognition in the text and acted more as a psychotherapist. ELIZA did not answer questions; it asked questions instead, and these questions were based on the keywords in human responses. Currently, there exist different types of chatbots according to communication channels (voice-enabled or text-based), knowledge domain (closed, general, or open domain), provided service (interpersonal or intrapersonal), used learning methods (rule-based, retrieval-based, or machine learning-based), and provided answers (extracted or generated answers).

Chatbot features such as its appearance/design or friendly human-like behavior are important, but not as important as adequate reactions to human requests and correct fluent answers. Therefore, a focus of our research is on natural language understanding (NLU) (responsible for comprehension of user questions) and natural language generation (NLG) (responsible for producing answers in the natural language) modules. Accurate generative-based chatbots are trained on large datasets; unfortunately, such datasets are not available for some languages. Despite this, companies/institutions often own small question–answer datasets (at least in the form of frequently asked questions (FAQs)) and want to have chatbot technology on their websites.

In this research, we explore seq2seq-based deep learning (DL) architecture solutions by training generative chatbots on extremely small datasets. Furthermore, our experiments are performed with two languages: English and morphologically complex Lithuanian. The generative chatbot creation task for the Lithuanian language is more challenging due to the following language characteristics: high inflection, almost twice larger vocabulary (compared to English), rich word-derivation system, and relatively free word-order in a sentence.

2. Related Work

In this review, we exclude all outdated rule-based or keyword-based techniques used in chatbot technology, focusing on the machine learning (ML) approaches [2] only, with particular attention paid to DL solutions (a review on different types of chatbots can also be found in Reference [3]). Existing ML-based chatbots can be grouped according to their model types, i.e., intent-detection or generative.

An intent-detection model is a typical example of a text classification task [4], where a classifier learns how to predict intents (classes) from a set of requests/questions (text instances) incoming from the user (a review of different intent detection techniques can also be found in Reference [5]). Since the intent is usually selected from a closed set of candidates and the related answer is prompted to the user, there is no flexibility in the content and the wording of these answers. Traditional ML approaches typically rely on a set of textual feature types and discrete feature representations. The research in Reference [6] described the investigation of *n*-grams, parts of speech, and support vector machine (SVM) classifiers with three categories (expressing user intent to purchase/quit, recommend/warn, or praise/complain) on the English dataset.

In general, intent detection is a broad task that can be defined as a churn detection (when a user expresses their intent to leave a service usually due to another customer) problem, a question topic identification problem, etc. The authors in Reference [7] offered an effective knowledge distillation and posterior regularization method for churn detection. Their approach enabled a convolutional neural network (CNN) applied on top of four types of pre-trained word embeddings (random, skip-gram, continuous bag-of-words, and gloVe) to learn simultaneously from three logic rules and supervised microblog English data. The research in Reference [8] also tackled the churn detection problem; however, in this case, it was solved for multilingual English and German conversations. The offered solution employed a CNN method with a bidirectional gated recurrent unit (BiGRU) applied on the pre-trained English and German fastText embeddings. The authors experimentally proved that their churn detection method tested on Twitter conversations was accurate and benefited from the multilingual approach. The authors in Reference [9] reported question topic intent-detection results for five languages (English and morphologically complex Estonian, Latvian, Lithuanian, and Russian). They investigated two neural classifiers: feed forward neural network (FFNN) and CNN with fastText embeddings. The accuracy of these classifiers was tested on three benchmark datasets (the datasets were originally in English, but the authors machine-translated them into other languages as well): askUbuntu (with 53 and 109 questions for testing and training, respectively);

chatbot (with 100 and 106); webApps (30 and 59). The authors claimed that, despite extremely small training data, their system demonstrated state-of-the-art performance.

In addition to previously summarized closed-set intent detection problems, some researchers dealt with a so-called zero-shot intent-detection problem by attempting to detect even those intents for which no labeled data are currently available. The research in Reference [10] presented a two-fold capsule-based architecture which (1) discriminates the existing intents with the bidirectional long short-term memory (BiLSTM) network and multiple self-attention heads, and (2) learns to detect emerging intents from the existing ones on the zero-shot via knowledge transfer (based on the evaluated similarity). The authors achieved sufficient results on two benchmark datasets: one containing English and another containing Chinese conversations.

Sometimes, the correct intent cannot be determined from one incoming question and must be clarified in further conversation. This type of intent detection is a so-called multi-turn response selection problem, which was addressed in Reference [11]. The offered deep attention matching network (DAMN) method uses representations of text segments at different granularities with stacked self-attention and then extracts truly matched segment pairs with attention across the whole context and the author's response. The authors proved the effectiveness of their method on the Ubuntu Corpus V1 (regarding the Ubuntu system troubleshooting in English) and the Douban Conversation Corpus (from social networking on open-domain topics in Chinese), both having ~0.5 million multi-turn contexts for training.

Furthermore, we focus on another large group of chatbots—in particular, generative chatbots—typically functioning in the machine translation manner; however, instead of translating from one language to another, they "translate" the input sequence (question) into the output sequence (answer) by sequentially generating text elements (usually words). This particular family of ML approaches is called sequence-to-sequence (abbreviated as seq2seq). Seq2seq chatbots can be created using either statistical machine translation (SML) or the recently popular neural machine translation (NMT) approaches. The pioneering work [12], describing how the SMT approach was applied to ~1.3 million conversations from the Twitter, demonstrated promising results and encouraged other researchers to continue working on generative chatbot problems.

Neural approaches typically employ the encoder–decoder architecture, where the encoder sequentially reads the input text (question) and encodes it into a fixed-length context vector and the decoder sequentially outputs the text (answer) after reading the context vector. The encoder–decoder approaches learn to generate answers based on either the isolated question–answer (QA) pairs or on the whole conversation.

The majority of approaches reuse (or slightly modify) the encoder–decoder architecture (described in Reference [13]) that was initially offered for NMT; both encoder and decoder are composed of long short-term memory (LSTM) networks applied on word2vec embeddings [14]. The authors in Reference [15] applied their seq2seq model to two benchmark English datasets: the closed-domain Information Technology (IT) Helpdesk Troubleshooting dataset and the open-domain OpenSubtitles dataset. A subjective evaluation of their system demonstrated its superiority over the Cleverbot (a popular chatterbot application). Similar research was performed in Reference [16]; however, instead of the English language, the authors applied the seq2seq method to a Chinese dataset, containing ~1 million QA pairs taken from Chinese online forums. The performed subjective evaluation proved that the offered approach achieved good results in modeling the responding style of a human. The authors in Reference [17] used an encoder–decoder architecture enhanced with an attention mechanism. Their method was successfully applied to a mixture of words and syllables as encoding/decoding units. Two Korean corpora were used to train this model: the larger non-dialogue corpus captured the Korean language model and the smaller dialogue corpus (containing ~0.5 million sentence pairs) collected from mobile chat rooms was used to train the dialogue.

The research in Reference [18] presented a conversational model focusing on previous queries/questions. They offered a hierarchical recurrent encoder–decoder neural-based approach that considers the history (i.e., sequences of words for each query and sequence of queries) of previously submitted queries, which was successfully trained on ~0.4 million English queries, demonstrating sufficient performance. Some researchers went even further by offering solutions for how to generate responses for a whole conversation to be successful. Reference [19] described a seminal work toward the creation of a neural conversational model for the long-term success of dialogues. The authors addressed the problem of long-term dialogues when some generated utterance influences the future outcomes. They integrated encoder–decoder (generating responses) and reinforcement learning (optimizing a future reward by capturing global properties of a good conversation) paradigms. The reward was determined with ease of answering, information flow, and semantic coherence conversational properties. The offered method was trained on an English dataset containing ~0.8 million sequences and, afterward, the conversation was successfully simulated between two virtual agents.

The comparative experiments with an encoder–decoder architecture demonstrated its superiority over SMT approaches and even information retrieval (IR)-based chatbots. The recurrent neural network (RNN) encoder–decoder trained with ~4.4 million Chinese microblogging post-response pairs outperformed SMT approaches [20]. The BLUE scores with the encoder–decoder (containing two LSTM neural networks with one-hot encoding for the input and the output) trained on ~0.7 million of English conversations were significantly better compared to the results with the IR method in Reference [21]. Moreover, the seq2seq-based generative conversational model (adapted from Reference [13] with tf-idf features) can enhance the results of IR systems [22]; it was experimentally proven with a model trained on ~0.66 million QA pairs in English, but sometimes mixed with Hindi. The hybrid approach in Reference [23] benefited from the combination of both IR and generative chatbot technology; it outperformed IR and generative chatbots used alone. The method uses IR to extract QA pair candidates and then re-ranks candidates based on the attentive seq2seq model. If some candidate is scored higher than the determined threshold, it is considered as the answer; otherwise, the answer is generated with the generation-based model. This chatbot trained on ~9 million QA pairs from an online customer service center was mainly adjusted for conversations in Chinese.

The analysis reveals that chatbot research mostly focused on English and Chinese languages, which have enough resources. Despite this, chatbots for other languages were sometimes created using artificial data as in Reference [9], where Estonian, Latvian, Lithuanian, and Russian intent-detection-based chatbots were trained on machine-translated English benchmark datasets. The existing conversational generative chatbots can be used with a wide range of languages, as demonstrated in Reference [24]. However, chatbot technology was not presented for any of these languages; instead, chatbot technology was demonstrated for English, testing the power of Google machine translation tools. Intent-detection models can be accurate even when trained on the smaller datasets, whereas, for generative models, hundreds of thousands or even millions of QA pairs are usually used.

The main contribution of our research is that we create a closed-domain generative chatbot by training it on an extremely small, but real dataset. Moreover, we train it on two languages (English and morphologically complex Lithuanian) and perform compare analysis. Furthermore, we investigate several encoder–decoder architectures applied to different word embedding types (one-hot encoding, fastText, and BERT). We anticipate that our findings could be interesting for researchers training generative chatbots on small data. To our knowledge, this is the first paper reporting generative chatbot results for a morphologically complex language.

3. Dataset

Experiments with generative chatbots were performed using a small domain-specific manually created dataset, having 567 QA pairs. This dataset contains real questions/answers about the company Tilde's (https://tilde.lt/) products, prices, supported languages, and used technologies. The dataset is available in two versions: English (EN) and Lithuanian (LT) (statistics about these datasets can be found in Table 1). Questions and answers in English are manually translated from Lithuanian.

The datasets for both languages were pre-processed using the following steps:

- Punctuation removal (when punctuation is ignored). This type of pre-processing is a right choice under the assumption that the punctuation in generated answers will be restored afterward.
- Punctuation separation (when punctuation is not removed, but separated from the tokens). This type allows training the generative chatbot on how to generate answers with words and punctuation marks at once. The punctuation is particularly important for languages (such as, e.g., Lithuanian) having relatively free word-order in a sentence where a single comma can absolutely change the sentence meaning. For example, the sentence *Bausti negalima pasigailėti* depending on the position of a comma can obtain two opposite meanings: *Bausti, negalima pasigailėti* (punishment, no mercy) and *Bausti negalima, pasigailėti* (no punishment, mercy).

Table 1. English (EN) and Lithuanian (LT) datasets used in generative chatbots (Q and A abbreviations stand for questions and answers, respectively).

	Number of *Q* and *A*	Number of Tokens in Set *Q*		Number of Tokens in Set *A*		Average *Q* Length in Tokens		Average *A* Length in Tokens	
	EN/LT	**EN**	**LT**	**EN**	**LT**	**EN**	**LT**	**EN**	**LT**
Removed punctuation	567	4025	2987	11,594	11,349	7.4	5.3	19.6	19.2
Separated punctuation	567	4728	3549	14,818	14,531	8.5	6.8	25.4	24.9

4. Seq2seq Models

The seq2seq framework introduced by Google was initially applied to NMT tasks [13,25]. Later, in the field of NLP, seq2seq models were also used for text summarization [26], parsing [27], or generative chatbots (as presented in Section 2). These models can address the challenge of a variable input and output length.

Formally, the seq2seq task can be described as follows: let $(x_1, x_2, \ldots, x_n)$ be an input (question) sequence and $(y_1, y_2, \ldots, y_m)$ be an output (answer) sequence, not necessarily of the same length ($n \neq m$). The architecture of seq2seq models contains two base components (see Figure 1): (1) anencoder, responsible for encoding the input $(x_1, x_2, \ldots, x_n)$ into an intermediate representation $h_n^{(Q)}$ (which is the last hidden state of the encoder), and (2) a decoder, responsible for decoding the intermediate representation $h_n^{(Q)} = h_0^{(A)}$ into the output $(y_1, y_2, \ldots, y_m)$. The conditional probability for $(x_1, x_2, \ldots, x_n)$ to generate $(y_1, y_2, \ldots, y_m)$ is presented in Equation (1).

$$\boldsymbol{p(y_1, y_2, \ldots, y_m | x_1, x_2, \ldots, x_n) = \prod_{t=1}^{m} p\left(y_t \middle| h_n^{(Q)}, y_1, \ldots, y_{t-1}\right)}. \tag{1}$$

Thus, the main goal of the task is to maximize the conditional probability $max(p(y_1, y_2, \ldots, y_m | x_1, x_2, \ldots, x_n))$ that, for $(x_1, x_2, \ldots, x_n)$, the most probable sequence $(y_1, y_2, \ldots, y_m)$ would be found.

More precisely, the encoder–decoder architecture is composed of several layers: (1) an encoder embedding layer, converting the input sequence into word embeddings: $(x_1, x_2, \ldots, x_n) \rightarrow (x_1', x_2', \ldots, x_n')$; (2) an encoder recurrent-based layer; (3) **a** decoder embedding layer, converting the output sequence into the word embeddings: $(y_1, y_2, \ldots, y_m) \rightarrow (y_1', y_2', \ldots, y_m')$; (4) a decoder recurrent-based layer; (5) a decoder output layer generating the conditional probability $o_t = p(y_t | h_n^{(Q)}, y_1, \ldots, y_{t-1})$ for y_t from the hidden state $h_t^{(A)}$ at each time step t. Virtual words <bos> and <eos> represent the beginning and the end of the answer, respectively; the generation of the answer is terminated after generation of <eos>.

In the encoder and decoder layers, we use the recurrent-based approach (suitable for processing sequential data, i.e., text), but not the simple recurrent neural network (RNN), because it suffers from the vanishing gradient problem and, therefore, is not suitable for longer sequences. Instead of RNN, we employ long short-term memory (LSTM) (introduced in Reference [28]) and bidirectional LSTM (introduced in Reference [29]) architectures, both having long memory. LSTM runs the input forward and, therefore, preserves information from the past, whereas bidirectional LSTM can run the input in two ways, forward and backward, thus preserving information both from the past and from the future in any hidden state. Both encoder and decoder are jointly trained; errors are backpropagated through the entire model and weights are adjusted. The encoder–decoder weights are adjusted in 200 epochs with the batch size equal to 5.

Three different seq2seq architectures (in Figures 2–4) explored in our experiments were implemented with the functional API using Tensorflow [30] (the python library used for dataflow and machine learning) with Keras [31] (the python library for deep learning). Used architectures were plotted with the ***plot_model*** function in Keras.

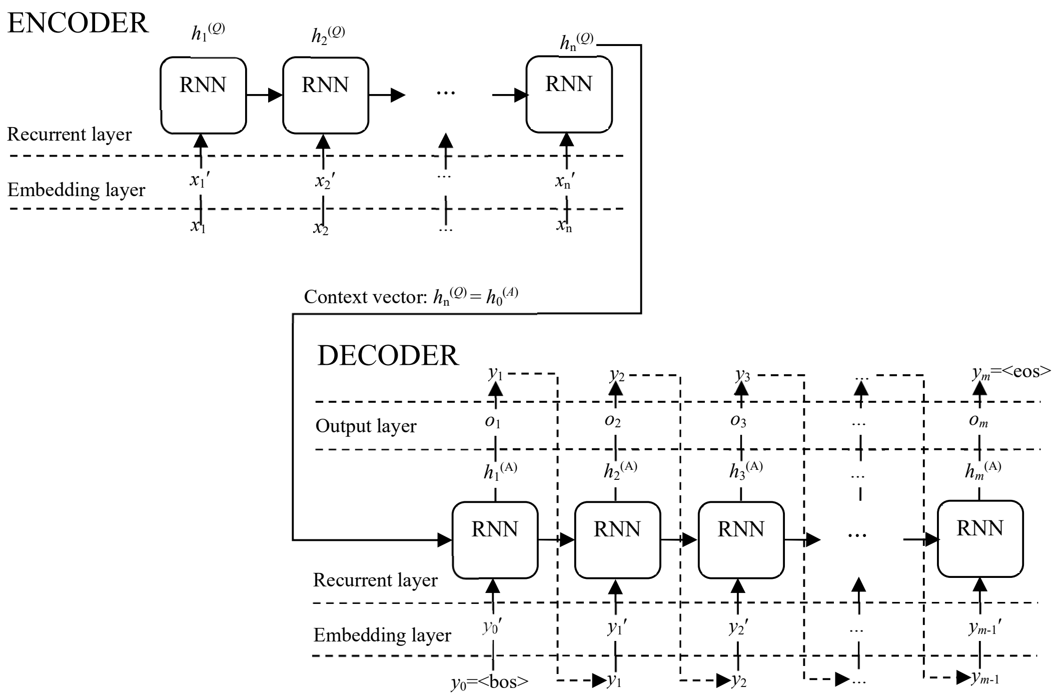
ENCODER
$h_1^{(Q)}$
$h_2^{(Q)}$
$h_n^{(Q)}$
RNN
Recurrent layer
Embedding layer
x_1' x_2' … x_n'
x_1 x_2 … x_n
Context vector: $h_n^{(Q)} = h_0^{(A)}$
DECODER
y_1 y_2 y_3 … y_m=<eos>
Output layer
o_1 o_2 o_3 … o_m
$h_1^{(A)}$ $h_2^{(A)}$ $h_3^{(A)}$ … $h_m^{(A)}$
y_0' y_1' y_2' … y_{m-1}'
y_0=<bos> y_1 y_2 … y_{m-1}

Figure 1. The basic recurrent-based encoder–decoder architecture.

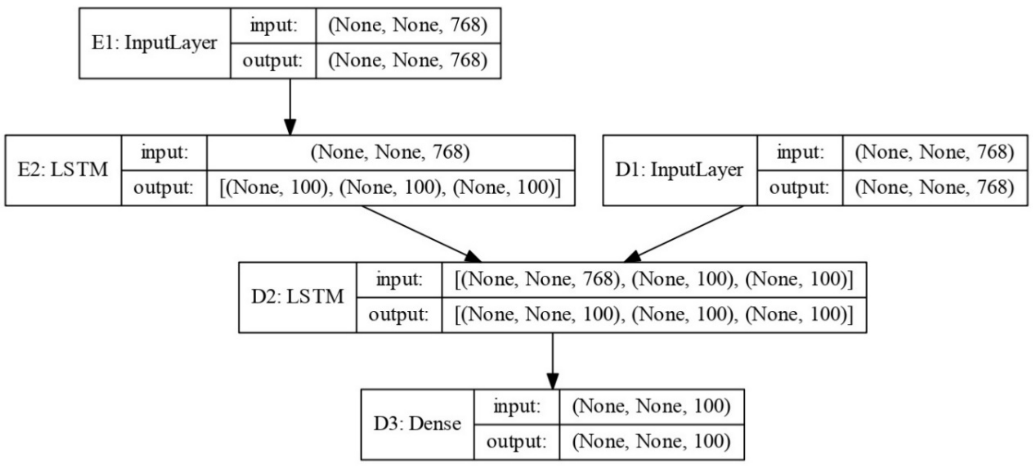
E1: InputLayer | input: (None, None, 768) | output: (None, None, 768)
E2: LSTM | input: (None, None, 768) | output: [(None, 100), (None, 100), (None, 100)]
D1: InputLayer | input: (None, None, 768) | output: (None, None, 768)
D2: LSTM | input: [(None, None, 768), (None, 100), (None, 100)] | output: [(None, None, 100), (None, 100), (None, 100)]
D3: Dense | input: (None, None, 100) | output: (None, None, 100)

Figure 2. One-layer long short-term memory (LSTM) for encoder and decoder (abbreviated as LSTM encoder and decoder). E1, E2 represent the encoder; D1, D2, D3 represent the decoder.

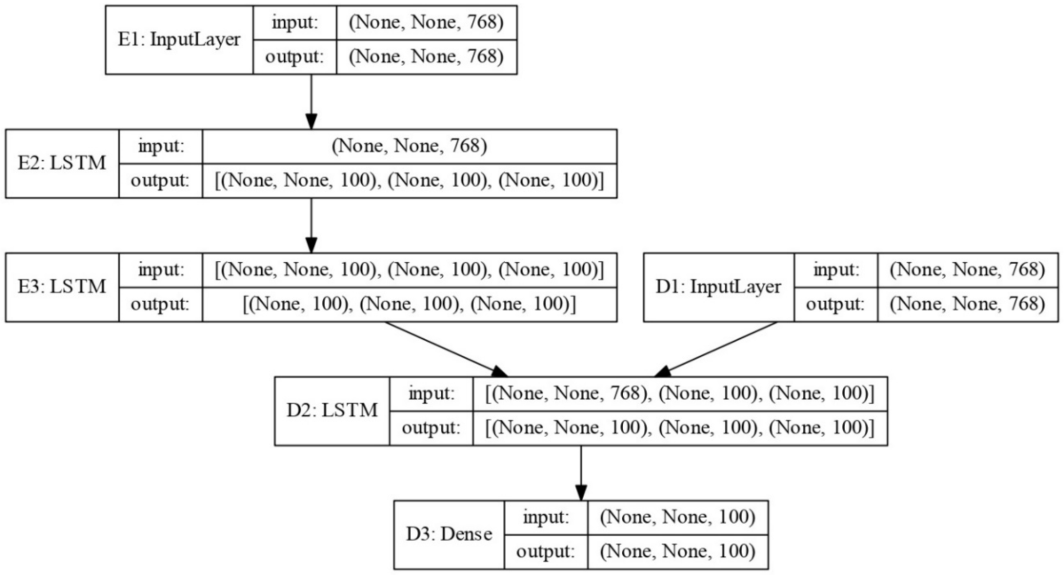
E1: InputLayer
input: (None, None, 768)
output: (None, None, 768)
E2: LSTM
input: (None, None, 768)
output: [(None, None, 100), (None, 100), (None, 100)]
E3: LSTM
input: [(None, None, 100), (None, 100), (None, 100)]
output: [(None, 100), (None, 100), (None, 100)]
D1: InputLayer
input: (None, None, 768)
output: (None, None, 768)
D2: LSTM
input: [(None, None, 768), (None, 100), (None, 100)]
output: [(None, None, 100), (None, 100), (None, 100)]
D3: Dense
input: (None, None, 100)
output: (None, None, 100)

Figure 3. Two-layer stacked LSTM encoder and one-layer LSTM decoder (abbreviated as LSTM2 encoder and LSTM decoder). E1, E2, E3 represent the encoder; D1, D2, D3 represent the decoder.

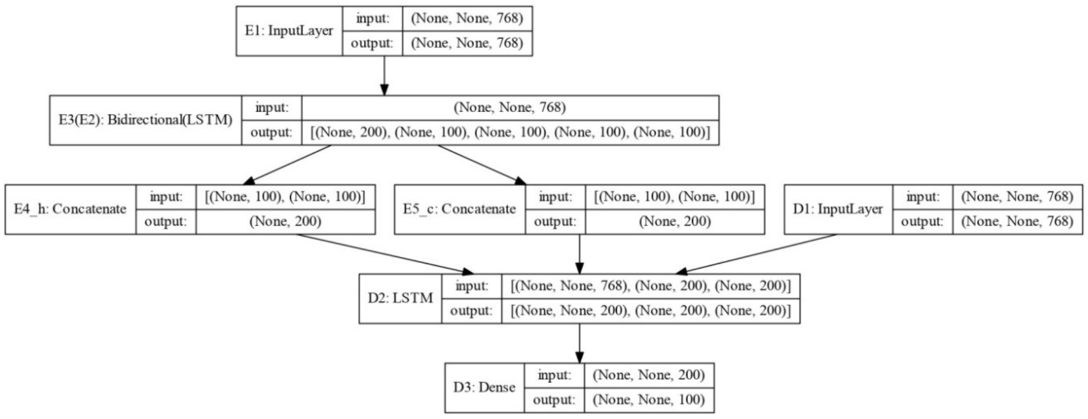
E1: InputLayer
input: (None, None, 768)
output: (None, None, 768)
E3(E2): Bidirectional(LSTM)
input: (None, None, 768)
output: [(None, 200), (None, 100), (None, 100), (None, 100), (None, 100)]
E4_h: Concatenate
input: [(None, 100), (None, 100)]
output: (None, 200)
E5_c: Concatenate
input: [(None, 100), (None, 100)]
output: (None, 200)
D1: InputLayer
input: (None, None, 768)
output: (None, None, 768)
D2: LSTM
input: [(None, None, 768), (None, 200), (None, 200)]
output: [(None, None, 200), (None, 200), (None, 200)]
D3: Dense
input: (None, None, 200)
output: (None, None, 100)

Figure 4. One-layer bidirectional LSTM (BiLSTM) encoder and one-layer LSTM decoder (abbreviated as BiLSTM encoder and LSTM decoder). E1, E2, E3, E4_h, E4_c represent the encoder; D1, D2, D3 represent the decoder.

All seq2seq architectures were applied on top of word embeddings. In our experiments, we used the following types:

- One-hot encoding. This is a discrete representation of words mapped into corresponding vectors containing all zero values, except for one value equal to 1. Thus, each word is represented by its unique vector with the value of 1 at a different position. A length of such a vector becomes the vocabulary size. Sizes of question and answer vocabularies are not necessary equal; therefore, the lengths of their one-hot encoding vectors are also different.
- FastText (presented in Reference [32]). FastText is a library (offered by Facebook's AI Research Lab) used to train neural distributional embeddings able to catch semantic similarities between words. In this research, we experiment with English (cc.en.300.vec) and Lithuanian (cc.lt.300.vec) fastText embeddings trained using continuous-bag-of-words (CBOW) with position-weights, in 300 dimensions, with character 5-g of window size five and 10 negatives [33]. Each fastText word embedding is a sum of *n*-gram vectors incoming into that word (e.g., 5-g *chatb*, *hatbo*, *atbot* *n*-grams compose the word *chatbot*). FastText word embeddings can be created even for misspelled words, and the obtained vectors are close to their correct

equivalents. This is an advantage for languages having the missing diacritics problem in non-normative texts. Despite the Lithuanian language having this problem, the dataset that we used in this research contained only normative texts.

- BERT (Bidirectional Encoder Representations from Transformers) (offered in Reference [34]). This is Google's neural-based technique (with multi-directional language modeling and attention mechanisms), which demonstrates state-of-the-art performance on a wide range of NLP tasks, including chatbot technology. Distributional BERT embeddings are also robust to disambiguation problems, i.e., homonyms with BERT are represented by different vectors. In our experiments, we used the BERT service [35] with the base multilingual cased 12-layer, 768-hidden, 12-head model for 104 languages (covering English and Lithuanian).

Different vectorization types were tested in the following encoder–decoder units:

1. One-hot encoding with both encoder and decoder;
2. FastText (abbreviated ft) embeddings with the encoder and one-hot encoding with the decoder;
3. BERT embeddings with the encoder and one-hot encoding with the decoder;
4. FastText embeddings with both encoder and decoder;
5. BERT embeddings with both encoder and decoder.

5. Experiments and Results

To avoid subjectivity and the high human evaluation cost, the quality of the chatbot output is typically evaluated using machine translation evaluation metrics such as BLUE [36] or text summarization metrics such as ROUGE [37].

The BLEU (Bilingual Evaluation Understudy) score compares the chatbot-generated output text (so-called hypothesis) with a human-produced answer (so-called reference or gold-standard) text and indicates how many *n*-grams in the output text appear in the reference. The BLUE score can be any value within the interval [0, 1] and is mathematically defined in Equation (2).

$$BLUE = BP \times \left(\prod_{n=1}^{N} precision_n\right)^{1/N}, \tag{2}$$

where N is the a maximum *n*-gram number $n = [1, N]$ ($N = 4$ in our evaluations).

BP (brevity penalty) and $precision_n$ are defined with Equations (3) and (4), respectively.

$$BP = \min\left(1, \exp\left(1 - \frac{ref_{length}}{out_{length}}\right)\right), \tag{3}$$

where ref_{length} is a reference length, and out_{length} is the chatbot output length.

$$precision_n = \frac{\sum_n \min\left(m_{out}^n, m_{ref}^n\right)}{\sum_{n'} m_{out}^{n'}}, \tag{4}$$

where m_{out}^n is the number of *n*-grams in the chatbot output matching the reference, m_{ref}^n is the number of *n*-grams in the reference, and $\sum_{n'} m_{out}^{n'}$ is the total number of *n*-grams in the chatbot output.

The BLUE scores were calculated using the python implemented *blue_score* module from the *translate* package in the *nltk* platform [38]. The *blue_score* parameters were initialized with the default values, except for the smoothing function, which was set to *method2* [39] (adding 1 to both numerator and denominator); *n*-gram weights were set to 0.25 and $N = 4$.

Depending on the interval the BLUE score values are in, the chatbot result can be interpreted as useless (if BLUE < 10), hard to get the gist (10–19), the gist is clear, but has significant errors (20–29), average quality (30–40), high quality (40–50), very high quality (50–60), and better than human (>60). These values and interpretations were taken from the Google Cloud's AI and Machine Learning product description in Reference [40].

The ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric measures the similarity of the chatbot-generated output to the reference text. $ROUGE\text{-}N_{precision}$, $ROUGE\text{-}N_{recall}$, and $ROUGE\text{-}N_{f\text{-}score}$ are mathematically defined by Equations (5)–(7), respectively.

$$ROUGE\text{-}N_{precision} = \frac{m^n_{overlap}}{m^n_{out}}, \tag{5}$$

where m^n_{out} is the number of n-grams in the chatbot output, and $m^n_{overlap}$ is the number of overlapping n-grams in the chatbot output and the reference.

$$ROUGE\text{-}N_{recall} = \frac{m^n_{overlap}}{m^n_{ref}}, \tag{6}$$

where m^n_{ref} is the number of n-grams in the reference, and $m^n_{overlap}$ is the number of overlapping n-grams in the chatbot output and the reference.

$$ROUGE\text{-}N_{f\text{-}score} = \frac{2 \times ROUGE\text{-}N_{precision} \times ROUGE\text{-}N_{recall}}{ROUGE\text{-}N_{precision} + ROUGE\text{-}N_{recall}}. \tag{7}$$

In addition to ROUGE-N, we also evaluated ROUGE-L, which measures the longest common subsequence (LCS) of tokens and even catches sentence-level word order. $ROUGE\text{-}L_{precision}$, $ROUGE\text{-}L_{recall}$, and $ROUGE\text{-}L_{f\text{-}score}$ are mathematically defined by Equations (8)–(10), respectively.

$$ROUGE\text{-}L_{precision} = \frac{LCS}{m_{out}}, \tag{8}$$

where m_{out} is the number of tokens in the chatbot output.

$$ROUGE\text{-}L_{recall} = \frac{LCS}{m_{ref}}, \tag{9}$$

where m_{ref} is the number of tokens in the reference.

$$ROUGE\text{-}L_{f\text{-}score} = \frac{2 \times ROUGE\text{-}L_{precision} \times ROUGE\text{-}L_{recall}}{ROUGE\text{-}L_{precision} + ROUGE\text{-}L_{recall}}. \tag{10}$$

Both ROUGE-N and ROUGE-L differ from the BLUE score because they do not have the brevity penalty (responsible for high-scoring outputs matching the reference text in length). ROUGE-N computes the n-gram match only for the chosen fixed n-gram size n, whereas ROUGE-L searches for the longest overlapping sequence of tokens and does not require defining the n-grams at all. Both ROUGE-N and ROUGE-L can be any value within the interval [0, 1]. Despite this, there are neither defined thresholds nor their interpretations; higher ROUGE-N and ROUGE-L values denote the higher intelligence of the chatbot. Both metrics are important for our intra-comparison purposes.

For ROUGE-N and ROUGE-L evaluation, we used Python library *rouge* [41], with $N = 2$ in all our evaluations.

To train and test our seq2seq models and to overcome the problem of a very small dataset (described in Section 3), we used five-fold cross validation (20% of the training data were used for validation). The obtained results were averaged and the confidence intervals (with a confidence level of 95% and alpha = 0.05) were calculated.

The BLUE scores with the LSTM encoder and decoder model applied using different vectorization types to the datasets with removed and separated punctuation are presented in Figure 5 and Figure 6, respectively. The ROUGE-2 and ROUGE-L precision/recall/f-score values are presented in Table 2 and Table 3, respectively.

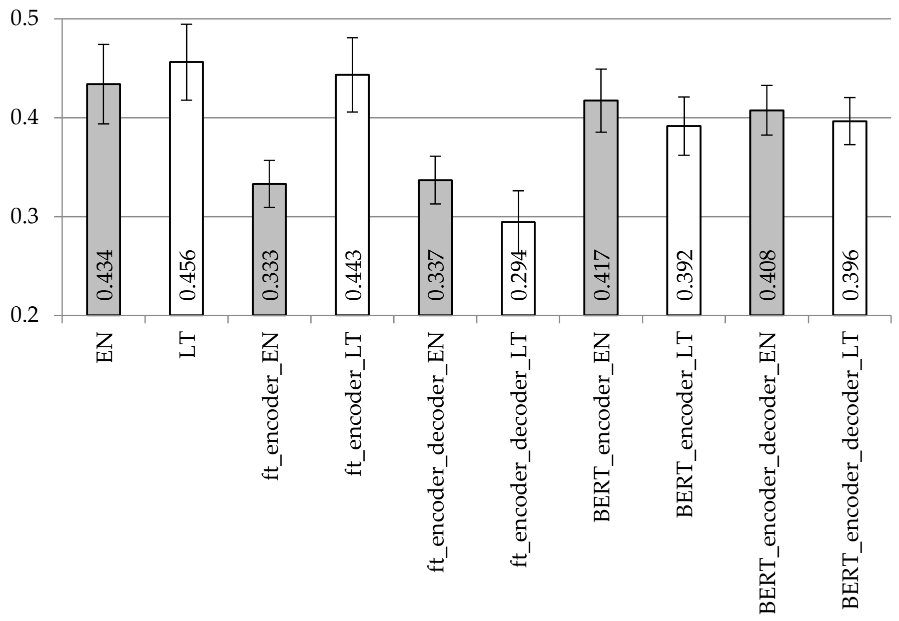

Figure 5. Averaged BLUE scores with confidence intervals using the LSTM encoder and decoder model with different vectorization types on the dataset with removed punctuation.

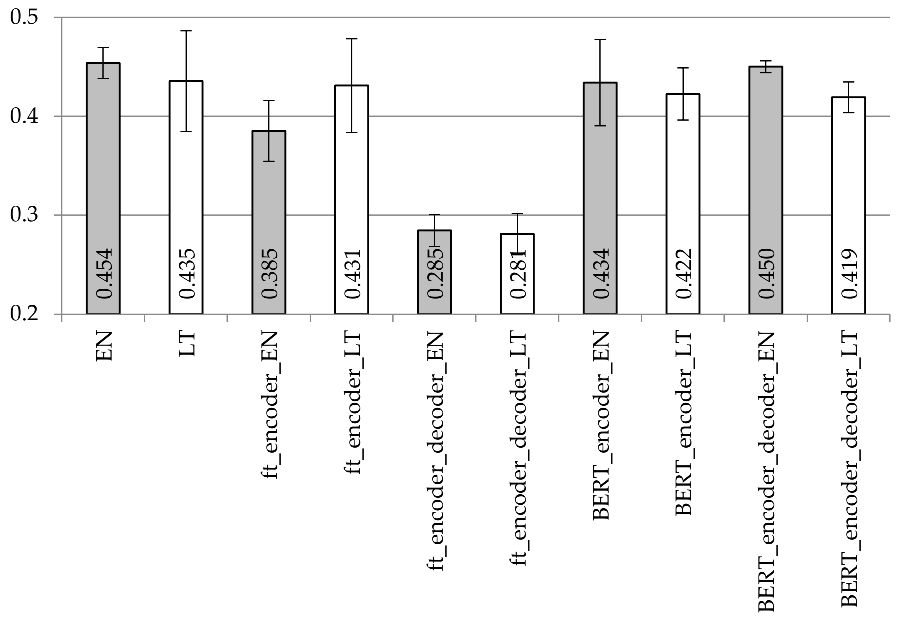

Figure 6. Averaged BLUE scores with confidence intervals using the LSTM encoder and decoder model with different vectorization types on the dataset with separated punctuation.

Despite the used punctuation treatment, the highest BLUE values (as presented in Figures 5 and 6) with the simple LSTM encoder architecture were achieved using one-hot encoding vectorization for both encoder and decoder. FastText vectorization is recommended only for the encoder and only for the Lithuanian language. Differences in BLUE scores are not so huge with the BERT vectorization, whether for different languages or for different vectorization units (BERT for encoder; BERT for encoder plus decoder).

Table 2. Averaged Recall-Oriented Understudy for Gisting Evaluation (ROUGE)-2 and ROUGE-L precision/recall/f-score values with confidence intervals using the LSTM encoder and decoder model with different vectorization types on the dataset with removed punctuation. The best values in the column for EN and LT are in bold. BERT—Bidirectional Encoder Representations from Transformers; ft—FastText.

Experiments	**ROUGE-2**			**ROUGE-L**		
	Precision	**Recall**	**f-Score**	**Precision**	**Recall**	**f-Score**
EN	**0.458 ± 0.035**	**0.478 ± 0.025**	**0.463 ± 0.031**	**0.525 ± 0.033**	**0.559 ± 0.022**	**0.522 ± 0.030**
LT	**0.467 ± 0.042**	**0.475 ± 0.044**	**0.469 ± 0.043**	**0.526 ± 0.035**	**0.552 ± 0.045**	**0.522 ± 0.038**
ft_encoder_EN	0.307 ± 0.020	0.323 ± 0.025	0.310 ± 0.022	0.385 ± 0.018	0.423 ± 0.021	0.382 ± 0.017
ft_encoder_LT	0.401 ± 0.038	0.411 ± 0.038	0.403 ± 0.038	0.476 ± 0.039	0.498 ± 0.041	0.470 ± 0.041
ft_encoder_decoder_EN	0.326 ± 0.022	0.363 ± 0.024	0.337 ± 0.023	0.416 ± 0.023	0.483 ± 0.025	0.422 ± 0.024
ft_encoder_decoder_LT	0.296 ± 0.040	0.405 ± 0.060	0.331 ± 0.044	0.372 ± 0.033	0.491 ± 0.050	0.388 ± 0.036
BERT_encoder_EN	0.395 ± 0.034	0.418 ± 0.032	0.401 ± 0.034	0.474 ± 0.029	0.515 ± 0.026	0.474 ± 0.029
BERT_encoder_LT	0.400 ± 0.026	0.427 ± 0.030	0.407 ± 0.027	0.469 ± 0.032	0.518 ± 0.037	0.471 ± 0.034
BERT_encoder_decoder_EN	0.402 ± 0.019	0.426 ± 0.020	0.409 ± 0.019	0.473 ± 0.023	0.517 ± 0.023	0.477 ± 0.023
BERT_encoder_decoder_LT	0.369 ± 0.026	0.386 ± 0.029	0.372 ± 0.026	0.442 ± 0.030	0.472 ± 0.026	0.436 ± 0.025

Table 3. Averaged ROUGE-2 and ROUGE-L precision/recall/f-score values with confidence intervals using the LSTM encoder and decoder model with different vectorization types on the dataset with separated punctuation. The best values in the column for EN and LT are in bold.

Experiments	**ROUGE-2**			**ROUGE-L**		
	Precision	**Recall**	**f-Score**	**Precision**	**Recall**	**f-Score**
EN	**0.456 ± 0.020**	**0.474 ± 0.015**	**0.459 ± 0.016**	0.526 ± 0.011	**0.558 ± 0.008**	**0.520 ± 0.007**
LT	**0.431 ± 0.043**	**0.431 ± 0.044**	**0.428 ± 0.043**	**0.497 ± 0.040**	**0.515 ± 0.040**	**0.488 ± 0.041**
ft_encoder_EN	0.346 ± 0.028	0.363 ± 0.025	0.349 ± 0.026	0.429 ± 0.024	0.464 ± 0.021	0.425 ± 0.022
ft_encoder_LT	0.380 ± 0.043	0.389 ± 0.045	0.381 ± 0.044	0.454 ± 0.039	0.482 ± 0.037	0.451 ± 0.038
ft_encoder_decoder_EN	0.290 ± 0.018	0.362 ± 0.049	0.312 ± 0.027	0.388 ± 0.023	0.479 ± 0.049	0.397 ± 0.030
ft_encoder_decoder_LT	0.287 ± 0.018	0.411 ± 0.029	0.328 ± 0.022	0.374 ± 0.026	0.508 ± 0.027	0.393 ± 0.026
BERT_encoder_EN	0.428 ± 0.012	0.426 ± 0.007	0.421 ± 0.009	0.511 ± 0.020	0.525 ± 0.011	0.494 ± 0.013
BERT_encoder_LT	0.401 ± 0.028	0.417 ± 0.026	0.404 ± 0.028	0.464 ± 0.022	0.503 ± 0.020	0.460 ± 0.022
BERT_encoder_decoder_EN	0.455 ± 0.012	0.454 ± 0.020	0.448 ± 0.016	**0.531 ± 0.007**	0.546 ± 0.015	0.513 ± 0.012
BERT_encoder_decoder_LT	0.409 ± 0.027	0.423 ± 0.028	0.410 ± 0.028	0.473 ± 0.021	0.513 ± 0.024	0.471 ± 0.023

In Table 2 and Table 3, we see the same trend as in Figure 5 and Figure 6, respectively. Hence, one-hot encoding vectorization for both units is the best solution with the simple LSTM encoder–decoder architecture for both English and Lithuanian languages. FastText vectorization with the encoder and decoder units is not recommended.

The BLUE scores with the LSTM2 encoder and decoder model applied with different vectorization types to the datasets with removed and separated punctuation are presented in Figure 7 and Figure 8, respectively. The ROUGE-2 and ROUGE-L precision/recall/f-score values are presented in Table 4 and Table 5, respectively.

When comparing Figure 7 with Figure 8 representing the BLUE scores with the stacked LSTM encoder, it is difficult to draw very firm conclusions. One-hot encoding for both encoder and decoder units remains the best vectorization technique on the English dataset with the separated punctuation. However, if excluding this particular result, BERT vectorization (for the encoder or encoder plus decoder) stands out with the best achieved BLUE values for both languages. Surprisingly, with the separated punctuation and BLUE vectorization, the results on the Lithuanian dataset are even better. When comparing Figure 5 with Figure 7 (simple LSTM with stacked LSTM on the removed punctuation) and Figure 6 with Figure 8 (simple LSTM with stacked LSTM on the separated punctuation), the stacked LSTM encoder demonstrates similar results with the fastText encoding;

however, BERT encoding is recommended with the removed punctuation for both languages, but it is definitely not the choice with separated punctuation in the English language.

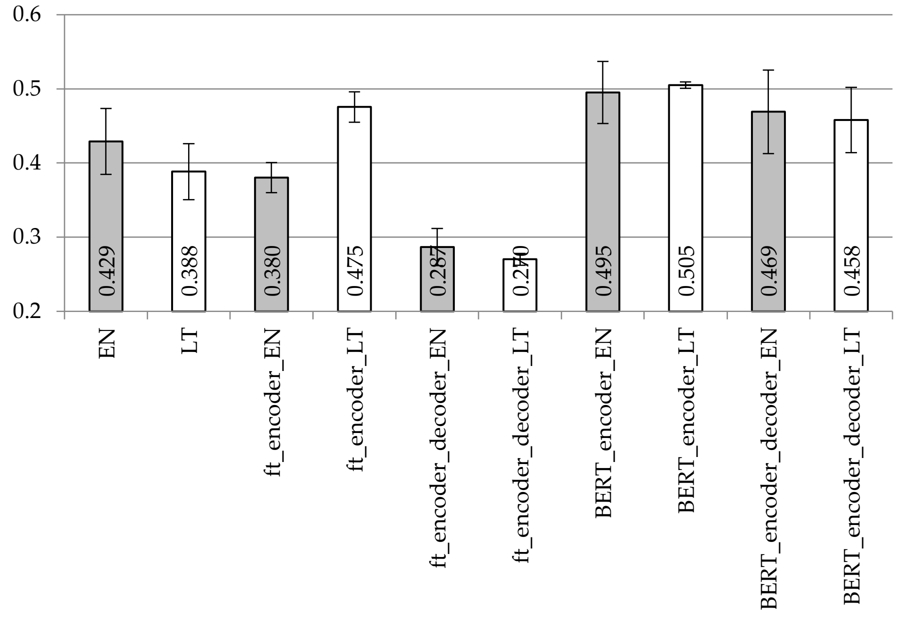

Figure 7. Averaged BLUE scores with confidence intervals using the LSTM2 encoder and LSTM decoder model with different vectorization types on the dataset with removed punctuation.

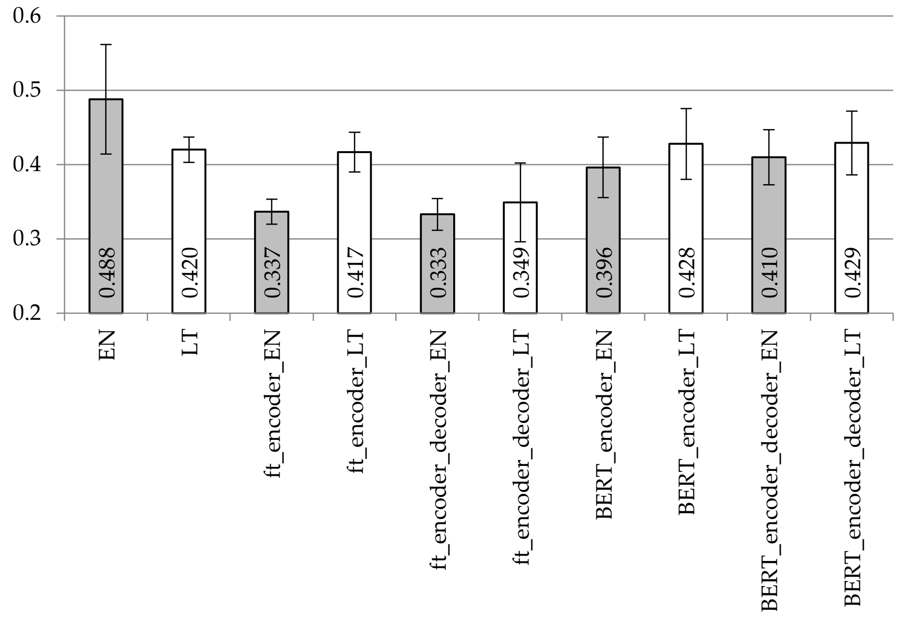

Figure 8. Averaged BLUE scores with confidence intervals using the LSTM2 encoder and LSTM decoder model with different vectorization types on the dataset with separated punctuation.

Table 4. Averaged ROUGE-2 and ROUGE-L precision/recall/f-score values with confidence intervals using the LSTM2 encoder and LSTM decoder model with different vectorization types on the dataset with removed punctuation. The best values in the column for EN and LT are in bold.

Experiments	ROUGE-2			ROUGE-L		
	Precision	**Recall**	**f-Score**	**Precision**	**Recall**	**f-Score**
EN	0.420 ± 0.033	0.432 ± 0.024	0.416 ± 0.029	0.482 ± 0.032	0.508 ± 0.022	0.469 ± 0.029
LT	0.368 ± 0.031	0.378 ± 0.025	0.368 ± 0.027	0.432 ± 0.025	0.455 ± 0.022	0.423 ± 0.024
ft_encoder_EN	0.331 ± 0.030	0.348 ± 0.045	0.332 ± 0.037	0.416 ± 0.023	0.451 ± 0.047	0.410 ± 0.032
ft_encoder_LT	0.395 ± 0.010	0.405 ± 0.010	0.397 ± 0.011	0.470 ± 0.007	0.490 ± 0.002	0.463 ± 0.006
ft_encoder_decoder_EN	0.298 ± 0.025	0.384 ± 0.030	0.328 ± 0.026	0.404 ± 0.033	0.506 ± 0.027	0.418 ± 0.030
ft_encoder_decoder_LT	0.290 ± 0.030	0.365 ± 0.041	0.315 ± 0.033	0.368 ± 0.027	0.456 ± 0.030	0.379 ± 0.027
BERT_encoder_EN	**0.488 ± 0.020**	**0.488 ± 0.030**	**0.483 ± 0.026**	**0.573 ± 0.015**	**0.585 ± 0.033**	**0.557 ± 0.026**
BERT_encoder_LT	**0.469 ± 0.021**	**0.476 ± 0.018**	**0.470 ± 0.020**	**0.528 ± 0.018**	**0.553 ± 0.019**	**0.525 ± 0.017**
BERT_encoder_decoder_EN	0.446 ± 0.048	0.441 ± 0.047	0.438 ± 0.046	0.531 ± 0.036	0.536 ± 0.030	0.510 ± 0.032
BERT_encoder_decoder_LT	0.421 ± 0.042	0.435 ± 0.032	0.423 ± 0.039	0.481 ± 0.040	0.518 ± 0.028	0.481 ± 0.039

Table 5. Averaged ROUGE-2 and ROUGE-L precision/recall/f-score values with confidence intervals using the LSTM2 encoder and LSTM decoder model with different vectorization types on the dataset with separated punctuation. The best values in the column for EN and LT are in bold.

Experiments	ROUGE-2			ROUGE-L		
	Precision	**Recall**	**f-Score**	**Precision**	**Recall**	**f-Score**
EN	**0.445 ± 0.044**	**0.458 ± 0.046**	**0.446 ± 0.042**	**0.503 ± 0.031**	0.530 ± 0.025	**0.497 ± 0.026**
LT	0.415 ± 0.051	0.429 ± 0.052	0.417 ± 0.052	0.475 ± 0.043	0.498 ± 0.049	0.464 ± 0.047
ft_encoder_EN	0.320 ± 0.024	0.337 ± 0.028	0.323 ± 0.025	0.408 ± 0.020	0.445 ± 0.029	0.406 ± 0.020
ft_encoder_LT	0.369 ± 0.038	0.380 ± 0.040	0.371 ± 0.039	0.442 ± 0.031	0.460 ± 0.033	0.435 ± 0.026
ft_encoder_decoder_EN	0.347 ± 0.021	0.399 ± 0.023	0.365 ± 0.021	0.449 ± 0.023	0.505 ± 0.031	0.448 ± 0.022
ft_encoder_decoder_LT	0.339 ± 0.039	0.421 ± 0.039	0.368 ± 0.037	0.421 ± 0.033	0.502 ± 0.028	0.433 ± 0.030
BERT_encoder_EN	0.396 ± 0.045	0.417 ± 0.048	0.400 ± 0.048	0.477 ± 0.036	0.519 ± 0.040	0.473 ± 0.039
BERT_encoder_LT	**0.418 ± 0.051**	**0.444 ± 0.050**	**0.424 ± 0.050**	**0.479 ± 0.045**	**0.520 ± 0.039**	**0.475 ± 0.042**
BERT_encoder_decoder_EN	0.407 ± 0.042	0.431 ± 0.045	0.413 ± 0.043	0.484 ± 0.033	**0.533 ± 0.040**	0.484 ± 0.037
BERT_encoder_decoder_LT	0.392 ± 0.037	0.417 ± 0.036	0.399 ± 0.037	0.452 ± 0.040	0.496 ± 0.030	0.455 ± 0.039

Table 4 and Table 5 demonstrate a similar trend to Figure 7 and Figure 8, respectively.

The BLUE scores with the BiLSTM encoder and decoder model applied with different vectorization types to the datasets with removed and separated punctuation are presented in Figure 9 and Figure 10, respectively. The ROUGE-2 and ROUGE-L precision/recall/f-score values are presented in Table 6 and Table 7, respectively.

With the BiLSTM encoder, BLUE values in Figures 9 and 10 are the highest with the BERT vectorization in both languages, especially when BERT vectorization is used for both encoder and decoder units. However, fastText vectorization for both encoder and decoder is not recommended for any of the used languages. When comparing BiLSTM encoder values with the appropriate simple LSTM or stacked LSTM values in Figures 5–8, the recommendation would be to use more sophisticated architectures. If choosing the appropriate vectorization, stacked LSTM works better using the English dataset and BiLSTM works better using the Lithuanian dataset.

As presented in Tables 6 and 7, BERT vectorization is the best choice with the BiLSTM encoder. The conclusions from ROUGE-2 and ROUGE-L values in Table 6 and Table 7 are consistent with the conclusions drawn from the appropriate BLUE values in Figure 9 and Figure 10, respectively.

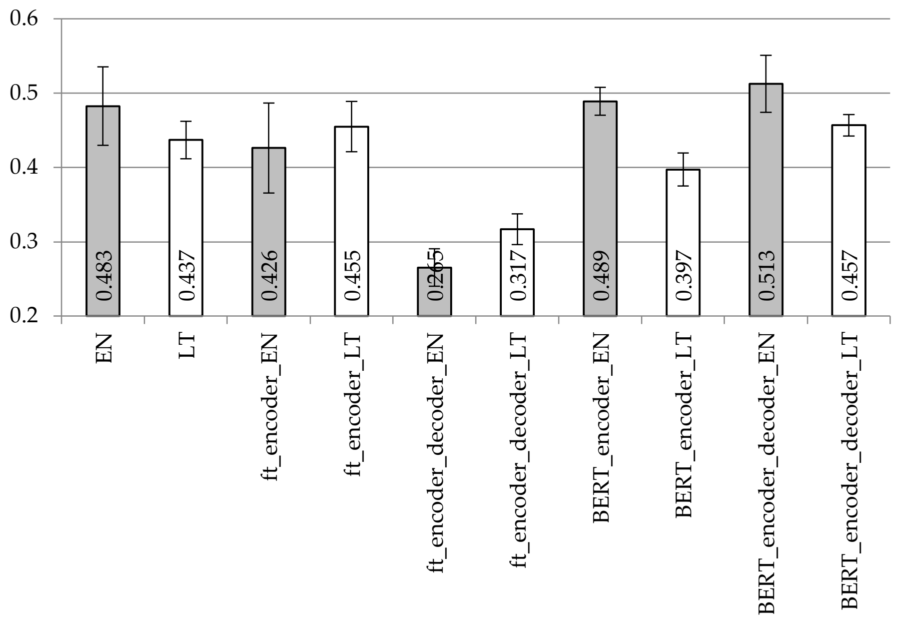

Figure 9. Averaged BLUE scores with confidence intervals using BiLSTM encoder and LSTM decoder model with different vectorization types on the dataset with removed punctuation.

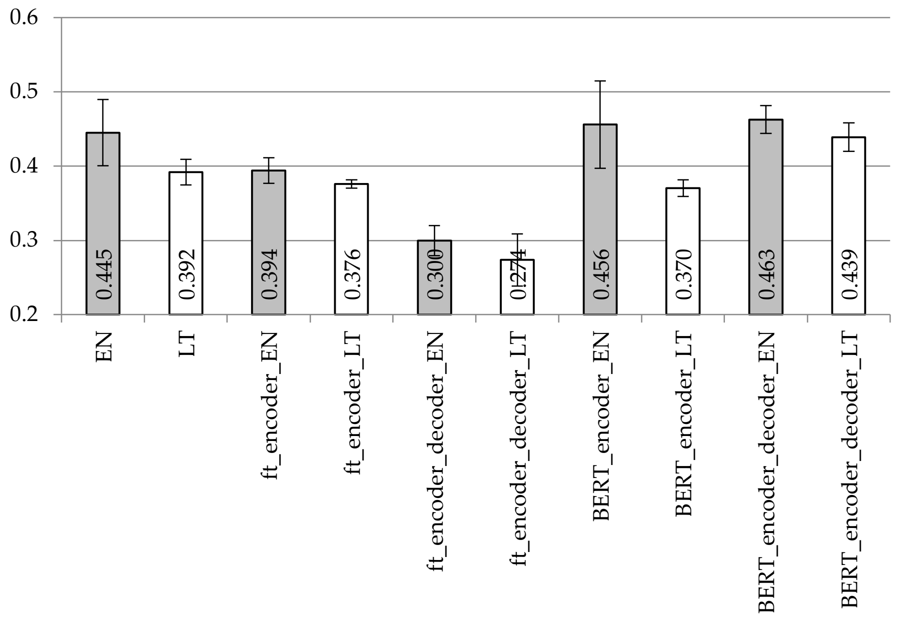

Figure 10. Averaged BLUE scores with confidence intervals using the BiLSTM encoder and LSTM decoder model with different vectorization types on the dataset with separated punctuation.

Table 6. Averaged ROUGE-2 and ROUGE-L precision/recall/f-score values with confidence intervals using the BiLSTM encoder and LSTM decoder model with different vectorization types on the dataset with removed punctuation. The best values in the column for EN and LT are in bold.

Experiments	**ROUGE-2**			**ROUGE-L**		
	Precision	**Recall**	**f-Score**	**Precision**	**Recall**	**f-Score**
EN	0.442 ± 0.018	0.455 ± 0.024	0.445 ± 0.020	0.512 ± 0.014	0.545 ± 0.019	0.510 ± 0.014
LT	0.413 ± 0.058	0.415 ± 0.057	0.411 ± 0.056	0.477 ± 0.059	0.488 ± 0.058	0.466 ± 0.055
ft_encoder_EN	0.368 ± 0.045	0.387 ± 0.043	0.373 ± 0.045	0.462 ± 0.043	0.492 ± 0.033	0.456 ± 0.039
ft_encoder_LT	0.414 ± 0.027	0.419 ± 0.021	0.413 ± 0.025	0.472 ± 0.026	0.491 ± 0.015	0.464 ± 0.026
ft_encoder_decoder_EN	0.290 ± 0.038	0.407 ± 0.043	0.328 ± 0.040	0.391 ± 0.039	0.508 ± 0.028	0.406 ± 0.037
ft_encoder_decoder_LT	0.325 ± 0.007	**0.479 ± 0.016**	0.372 ± 0.011	0.398 ± 0.007	**0.563 ± 0.016**	0.421 ± 0.009
BERT_encoder_EN	0.468 ± 0.015	0.477 ± 0.025	0.467 ± 0.018	0.561 ± 0.006	0.576 ± 0.018	0.546 ± 0.009
BERT_encoder_LT	0.392 ± 0.031	0.415 ± 0.027	0.398 ± 0.031	0.465 ± 0.031	0.520 ± 0.021	0.465 ± 0.032
BERT_encoder_decoder_EN	**0.496 ± 0.020**	**0.504 ± 0.016**	**0.495 ± 0.018**	**0.571 ± 0.018**	**0.601 ± 0.007**	**0.565 ± 0.014**
BERT_encoder_decoder_LT	**0.428 ± 0.024**	0.445 ± 0.024	**0.431 ± 0.023**	**0.483 ± 0.021**	0.525 ± 0.031	**0.482 ± 0.023**

Table 7. Averaged ROUGE-2 and ROUGE-L precision/recall/f-score values with confidence intervals using the BiLSTM encoder and LSTM decoder model with different vectorization types on the dataset with separated punctuation. The best values in the column for EN and LT are in bold.

Experiments	**ROUGE-2**			**ROUGE-L**		
	Precision	**Recall**	**f-Score**	**Precision**	**Recall**	**f-Score**
EN	0.420 ± 0.033	0.432 ± 0.024	0.416 ± 0.029	0.482 ± 0.032	0.508 ± 0.022	0.469 ± 0.029
LT	0.368 ± 0.031	0.378 ± 0.025	0.368 ± 0.027	0.432 ± 0.025	0.455 ± 0.022	0.423 ± 0.024
ft_encoder_EN	0.331 ± 0.030	0.348 ± 0.045	0.332 ± 0.037	0.416 ± 0.023	0.451 ± 0.047	0.410 ± 0.032
ft_encoder_LT	0.395 ± 0.010	0.405 ± 0.010	0.397 ± 0.011	0.470 ± 0.007	0.490 ± 0.002	0.463 ± 0.006
ft_encoder_decoder_EN	0.298 ± 0.025	0.384 ± 0.030	0.328 ± 0.026	0.404 ± 0.033	0.506 ± 0.027	0.418 ± 0.030
ft_encoder_decoder_LT	0.290 ± 0.030	0.365 ± 0.041	0.315 ± 0.033	0.368 ± 0.027	0.456 ± 0.030	0.379 ± 0.027
BERT_encoder_EN	0.446 ± 0.048	0.441 ± 0.047	0.438 ± 0.046	0.531 ± 0.036	0.536 ± 0.030	0.510 ± 0.032
BERT_encoder_LT	0.421 ± 0.042	0.435 ± 0.032	0.423 ± 0.039	0.481 ± 0.040	0.518 ± 0.028	0.481 ± 0.039
BERT_encoder_decoder_EN	**0.488 ± 0.020**	**0.488 ± 0.030**	**0.483 ± 0.026**	**0.573 ± 0.015**	**0.585 ± 0.033**	**0.557 ± 0.026**
BERT_encoder_decoder_LT	**0.469 ± 0.021**	**0.476 ± 0.018**	**0.470 ± 0.020**	**0.528 ± 0.018**	**0.553 ± 0.019**	**0.525 ± 0.017**

6. Discussion

Focusing on the results (when BLUE is considered as the primary metric and ROUGE is considered as the auxiliary metric for our analysis) allows us to make the statements below. Although with some small exceptions, there is a correlation between calculated BLUE scores and ROUGE-2/ROUGE-L values.

In most of the cases, results on datasets with separated punctuation were a bit lower compared to results with the removed punctuation pre-processing. This is because the separated punctuation task is more complicated; instead of generating words only, the seq2seq model has to generate both words and punctuation. The detailed error analysis revealed that the majority of errors were due to the incorrectly generated punctuation marks. Despite the language and the punctuation pre-processing type, the worst BLUE score values were calculated with the fastText embedding vectorization for both encoder and decoder units (*ft_encoder_decoder*). Furthermore, results on the Lithuanian dataset were even worse compared to English. On the contrary, the performance of the fastText encoder with one-hot decoding on the Lithuanian dataset was obviously better compared to English in all cases except for the dataset with separated punctuation and BiLSTM encoder and LSTM decoder.

However, BERT embeddings are definitely the better choice compared to fastText. With the LSTM encoder and decoder and the BiLSTM encoder and LSTM decoder models, BERT performed better in English, whereas the stacked LSTM encoder architecture (i.e., with the LSTM2 encoder and LSTM decoder) applied to BERT embeddings was more suitable for the morphologically complex Lithuanian.

One-hot vectorization for both encoder and decoder units with the simple LSTM encoder and decoder model outperformed other vectorization types. With more complex encoder units (i.e., stacked LSTM or BiLSTM), one-hot encoding was the best choice only for the separated punctuation dataset for the English language; however, in all other cases, it was outperformed by BERT embeddings.

The overall best BLUE/ROUGE-$2_{f\text{-}score}$/ROUGE-$L_{f\text{-}score}$ values on the English dataset with removed punctuation pre-processing were ~0.513/~0.495/~0.565. These results were achieved with the BiLSTM encoder and LSTM decoder model applied with BERT embeddings for both encoder and decoder units. The best BLUE/ROUGE-$2_{f\text{-}score}$/ROUGE-$L_{f\text{-}score}$ values on the Lithuanian dataset with removed punctuation were a bit lower, i.e., ~0.505/~0.470/~0.525. Values were obtained with the stacked LSTM2 encoder and LSTM decoder with BERT embeddings for the encoder unit and one-hot vectorization for the decoder. The best BLUE value (~0.488) on the English dataset with separated punctuation was calculated using the stacked LSTM2 encoder and LSTM decoder model applied on top of one-hot vectorization for both units. The best ROUGE-$2_{f\text{-}score}$ and ROUGE-$L_{f\text{-}score}$ values equal to ~0.483 and ~0.557, respectively, were achieved with the BiLSTM encoder and LSTM decoder model and BERT embeddings for both units. The best BLUE/ROUGE-$2_{f\text{-}score}$/ROUGE-$L_{f\text{-}score}$ values equal to ~0.439/~0.470/~0.525 on the Lithuanian dataset with separated punctuation were achieved using the BiLSTM encoder and LSTM decoder model and BERT embeddings for both units.

The lowest calculated BLUE score values (as a percentage) for both English and Lithuanian fell into the range 20–29, which means that the gist of the generated answers was still clear, but had significant errors (as explained in Section 5). The highest achieved values on the dataset with removed punctuation were at the beginning of the interval 50–60; therefore, the quality of the generated answers was considered as very high; the highest values in the dataset with separated punctuation was in the range 40–50 that is still considered as high quality. This allows us to conclude that the results are accurate enough to be applied in practice.

Despite practical benefits, this work also brings a scientific contribution. Generative chatbots were never previously trained for a morphologically complex language (such as Lithuanian). Furthermore, in general, they were never trained on such a small dataset. In addition to testing different encoder–decoder architectures (simple LSTM encoder, stacked LSTM encoder, BiLSTM encoder), different embedding types (one-hot, fastText, BERT), and different vectorization options (only encoder, encoder plus decoder), we tested two very different languages and formulated recommendations regarding what works best for each of these languages. Moreover, we anticipate that, for languages with similar characteristics under similar experimental conditions, similar results can be expected.

In addition to positive things, limitations of the investigated approach need to be mentioned as well. Similar accuracy can be expected only if the used dataset is in a closed domain and contains a limited number of topics. In our case, we tested the dataset covering rephrased questions and answers about a company’s products, as well as their prices, used technologies, and supported languages. Thus, such a generative chatbot cannot answer questions that are unrelated. Furthermore, BERT embeddings are recommended as the best vectorization option; however, these embeddings are supported for a limited number of languages (i.e., ~100 languages).

7. Conclusions

In this research, we presented the first generative chatbot results obtained on very small domain-specific datasets for English and morphologically complex Lithuanian languages.

We investigated different encoder–decoder architectures (with LSTM, stacked LSTM, and BiLSTM), different embedding types (one-hot, fastText, and BERT), different vectorization types in different encoder and decoder units, and different punctuation treatment (its removal or separation).

The best BLUE values on the English/Lithuanian datasets with removed and separated punctuation were ~0.513/~0.505 and ~0.488/~0.439, respectively. This research revealed that more complex architectures (based on the stacked LSTM and BiLSTM encoders) are more accurate compared to simple LSTM. It also demonstrates the advantages of BERT embeddings. Despite BLUE values for the Lithuanian language being a bit lower, compared to English, the results are good enough to be applied in practice. This research is important and interesting because (1) generative chatbots are trained on very small domain-specific data, and (2) it reports the first generative chatbot results for a morphologically complex language.

Despite the search for effective solutions on small datasets being much more challenging, in future research, we are planning to augment our datasets with newly covered topics, to experiment with different seq2seq architectures by tuning their hyper-parameters.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Weizenbaum, J. ELIZA—A computer program for the study of natural language communication between man and machine. *Commun. ACM* **1966**, *9*, 36–45.
2. Kotsiantis, S.B. Supervised Machine Learning: A Review of Classification Techniques. *Informatica* **2007**, *31*, 249–268.
3. Almansor, E.; Hussain, F.K. Survey on Intelligent Chatbots: State-of-the-Art and Future Research Directions. *Complex, Intell. Softw. Intensive Syst.* **2020**, *993*, 534–543.
4. Sebastiani, F. Machine Learning in Automated Text Categorization. *ACM Comput. Surv.* **2002**, *34*, 1–47.
5. Liu, J.; Li, Y.; Lin, M. Review of Intent Detection Methods in the Human-Machine Dialogue System. *J. Phys. Conf. Ser.* **2019**, *1267*, 012059.
6. Akulick, S.; Mahmoud, E.S. Intent Detection through Text Mining and Analysis. In Proceedings of the Future Technologies Conference (FTC), Vancouver, Canada, 29–30 November 2017; pp. 493–496.
7. Gridach, M.; Haddad, H.; Mulki, H. Churn identification in microblogs using convolutional neural networks with structured logical knowledge. In Proceedings of the 3rd Workshop on Noisy User-generated Text, Copenhagen, Denmark, 7 September 2017; pp. 21–30.
8. Abbet, C.; M'hamdi, M.; Giannakopoulos, A.; West, R.; Hossmann, A.; Baeriswyl, M.; Musat, C. Churn Intent Detection in Multilingual Chatbot Conversations and Social Media. In Proceedings of the 22nd Conference on Computational Natural Language Learning, Brussels, Belgium, 31 October–1 November 2018; pp. 161–170.
9. Balodis, K.; Deksne, D. FastText-Based Intent Detection for Inflected Languages. *Information* **2019**, *10*, 161.
10. Xia, C.; Zhang, C.; Yan, X.; Chang, Y.; Yu, P. Zero-shot User Intent Detection via Capsule Neural Networks. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 – November 4 2018; pp. 3090–3099.
11. Zhou, X.; Li, L.; Dong, D.; Liu, Y.; Chen, Y.; Zhao, W.X.; Yu, D.; Wu, H. Multi-Turn Response Selection for Chatbots with Deep Attention Matching Network. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers, Melbourne, Australia, 15–20 July 2018; pp. 1118–1127.
12. Ritter, A.; Cherry, C.; Dolan, W. Data-driven response generation in social media. In Proceedings of the Conference on Empirical Methods in Natural Language Processing(EMNLP 11), Edinburgh, Scotland, UK, 27–31 July 2011; pp. 583–593.
13. Sutskever, I.; Vinyals, O.; Le, Q.V. Sequence to Sequence Learning with Neural Networks. *Adv. Neural Inf. Process. Syst.* **2014**, *27*, 3104–3112.
14. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed Representations of Words and Phrases and their Compositionality. *Adv. Neural Inf. Process. Syst.* **2013**, *26*, 3111–3119.
15. Vinyals, O.; Le, Q. A neural conversational model. In Proceedings of the ICML Deep Learning Workshop, Lille, France, 10–11 July 2015.
16. Zhang, W.N.; Zhu, Q.; Wang, Y.; Zhao, Y.; Liu, T. Neural Personalized Response Generation as Domain Adaptation. *World Wide Web* **2019**, *22*, 1427–1446.
17. Kim, J.; Lee, H.-G.; Kim, H.; Lee, Y.; Kim, Y.-G. Two-Step Training and Mixed Encoding-Decoding for Implementing a Generative Chatbot with a Small Dialogue Corpus. In Proceedings of the Workshop on Intelligent Interactive Systems and Language Generation (2IS&NLG), Tilburg, the Netherlands, 5 November 2018; pp. 31–35.

18. Sordoni, A.; Bengio, Y.; Vahabi, H.; Lioma, C.; Simonsen, J.G.; Nie, J.-Y. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In Proceedings of the ACM International Conference on Information and Knowledge Management, Melbourne, Australia, 19–23 October 2015; pp. 553–562.
19. Li, J.; Monroe, W.; Ritter, A.; Jurafsky, D.; Galley, M.; Gao, J. Deep Reinforcement Learning for Dialogue Generation. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 2–6 November 2016; pp. 1192–1202.
20. Shang, L.; Lu, Z.; Li, H. Neural Responding Machine for Short-Text Conversation. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguisticsand the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; pp. 1577–1586.
21. Xu, A.; Liu, Z.; Guo, Y.; Sinha, V.; Akkiraju, R. A New Chatbot for Customer Service on Social Media. In Proceedings of the CHI Conference on Human Factors in Computing Systems, Denver, Colorado, USA, 2017; pp. 3506–3510.
22. Tammewar, A.; Pamecha, M.; Jain, C.; Nagvenkar, A.; Modi, K. Production Ready Chatbots: Generate if Not Retrieve. In Proceedings of the AAAI Workshops, New Orleans, LA, USA, 2–3 February 2018.
23. Qiu, M.; Li, F.-L.; Wang, S.; Gao, X.; Chen, Y.; Zhao, W.; Chen, H.; Huang, J.; Chu, W. AliMe Chat: A Sequence to Sequence and Rerank based Chatbot Engine. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Vancouver, Canada, 30 July–4 August 2017; pp. 498–503.
24. Vanjani, M.; Aiken, M.; Park, M. Chatbots for Multilingual Conversations. *J. Manag. Sci. Bus. Intell.* **2019**, *4*, 19–24.
25. Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1724–1734.
26. Zhang, Y.; Li, D.; Wang, Y.; Fang, Y.; Xiao, W. Abstract Text Summarization with a Convolutional Seq2seq Model. *Appl. Sci.* **2019**, *9*, 1665.
27. Konstas, I.; Iyer, S.; Yatskar, M.; Choi, Y.; Zettlemoyer, L. Neural AMR: Sequence-to-Sequence Models for Parsing and Generation. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Vancouver, Canada, 1 July – 4 August 2017, *CoRR* 2017, abs/1704.08381.
28. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780.
29. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610.
30. An End-to-End Open Source Machine Learning Platform. Available online: https://www.tensorflow.org/ (accessed on 12 November 2019).
31. Keras: The Python Deep Learning library. Available online: https://keras.io/ (accessed on 13 November 2019).
32. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146.
33. Grave, E.; Bojanowski, P.; Gupta, P.; Joulin, A.; Mikolov, T. Learning Word Vectors for 157 Languages. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7–12 May 2018.
34. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, USA, 3–5 June 2018; pp. 4171–4186.
35. Xiao, H. Bert-as-Service. Available online: https://github.com/hanxiao/bert-as-service (accessed on 9 December 2019).
36. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.-J. Bleu: A Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA, 7–12 July 2002; pp. 311–318.
37. Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In Proceedings of the ACL-04 Workshop, Barcelona, Spain, 21–26 July 2004; Volume 8.

38. Natural Language Toolkit: NLTK 3.4.5 Documentation. Available online: https://www.nltk.org/ (accessed on 20 December 2019).
39. Chin-Yew, L.; Och, F.J. Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics. In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04), Barcelona, Spain, 21–26 July 2004; pp. 605–612.
40. Google Cloud AI & Machine Learning Products. Evaluating Models. Available online: https://cloud.google.com/translate/automl/docs/evaluate (accessed on 29 January 2020).
41. Python Library for Rouge. Available online: https://github.com/pltrdy/rouge/ (accessed on 8 January 2020).