Technical University of Košice
Faculty of Electrical Engineering and Informatics

Deep Neural Networks Architectures for Chatbot Creation

Master's Thesis

2023 Bc. Kristína Rolfová

Technical University of Košice
Faculty of Electrical Engineering and Informatics

Deep Neural Networks Architectures for Chatbot Creation

Master's Thesis

Study Programme:	Intelligent Systems
Field of study:	Computer Science
Department:	Department of Cybernetics and Artificial Intelligence (DCAI)
Supervisor:	Ing. Martina Szabóová, PhD.
Consultant(s):	

Košice 2023 **Bc. Kristína Rolfová**

Abstract

This thesis presents the development of an emotion-aware chatbot aimed at improving conversational experiences through the recognition and appropriate response to users' emotions. Lack of emotional fluency is a significant factor in conversational breakdowns in human-robot interaction, especially for smaller models, yet it rarely gets focused explicitly on. The goal of this work is to peruse different approaches to integrating emotional information into model training and create an open-domain chatbot model based on said approaches. We evaluate the created models by both automated and human means and incorporate the best model into a stand-alone web application for user convenience.

Keywords

empathetic chatbots, HRI, transformers, web application

Abstrakt

Táto práca prezentuje vývoj chatbota so zameraním na emócie, s cieľom zlepšiť konverzačný tok prostredníctvom rozpoznávania a adekvátnej odpovede na emócie používateľov. Neadekvátna reakcia na emočné vstupy je významným faktorom konverzačných zlomov v interakcii človek-robot, najmä pri menších modeloch, avšak zriedkavo sa tomuto faktoru venuje osobitná pozornosť. Cieľom tejto práce je preskúmať rôzne prístupy k integrácii emočných vstupov do procesu trénovania chatbot modelov a vytvoriť konverzačného agenta schopného interakcie v otvorenej doméne na základe týchto prístupov. Vytvorené modely sú hodnotené automatizovanými aj ľudskými prostriedkami a najlepší model integrujeme do samostatnej webovej aplikácie pre pohodlie používateľa.

Kľúčové slová

empatický chatbot, interakcia človek-robot, transformerové modely, webová aplikácia

68274

TECHNICAL UNIVERSITY OF KOŠICE

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATICS

Department of Cybernetics and Artificial Intelligence

DIPLOMA THESIS ASSIGNMENT

Field of study:	**Computer Science**
Study programme:	**Intelligent Systems**

Thesis title:

Deep Neural Networks Architectures for Chatbot Creation

Architektúry hlbokých sietí pri vývoji chatbotov

Student:	**Bc. Kristína Rolfová**
Supervisor:	**Ing. Martina Szabóová, PhD.**
Supervising department:	**Department of Cybernetics and Artificial Intelligence**
Consultant:	
Consultant`s affiliation:	

Thesis preparation instructions:

1. Give an overview of the current state of deep neural network architectures for chatbot development.
2. Analyze existing approaches and techniques for creating chatbots using deep neural networks.
3. Design and implement a custom deep neural network architecture for chatbot.
4. Evaluate the performance of the custom architecture through experimentation and testing.
5. Prepare documentation according to the guidelines of the supervisor.

Language of the thesis:	English
Thesis submission deadline:	21.04.2023
Assigned on:	31.10.2022

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
-12-

[illegible]

prof. Ing. Liberios Vokorokos, PhD.

Dean of the Faculty

Declaration

I hereby declare that this thesis is my own work and effort. Where other sources of information have been used, they have been acknowledged.

Košice, April 21, 2023 ....

Signature

Acknowledgement

I would like to thank my supervisor, Ing. Martina Szabóová, PhD., for her guidance and constructive advice during the making of this thesis. I thank my mother for her unwavering emotional support and helpful perspective - in some ways, this is your work too. I also offer my gratitude to all the participants in the model evaluation survey.

Contents

List of Figures

List of Tables

List of Terms

CNN - Convolutional Neural Network

DD - Daily Dialog dataset

ED - Empathetic Dialogues dataset

HRI - Human-Robot Interaction

LLaMA - Large Language Model Meta AI

LSTM - Long Short Term Memory

NER - Named Entity Recognition

NLG - Natural Language Generation

NLP - Natural Language Processing

POS - Part of Speech

RNN - Recurrent Neural Network

seq2seq - Sequence to Sequence models

Introduction

What would be the content of sci-fi franchises just a few decades previous is now a reality. The inclusion of artificial intelligence in everyday life has increasingly emerged in areas ranging from retail, hotels, transportation hubs, e-commerce and restaurants.

The nature of customer experience is being revolutionized, the paradigm-shifting from high-touch to high-tech, allowing customers to get simple requests fulfilled on demand and freeing the human employee to focus on more complicated, important tasks while the robot takes care of the minutiae.

Such applications of robots are no longer just theoretical. In 2019, Hilton Worldwide has successfully applied the robot Connie as a concierge, and as early as 2015 Japanese Kinosaki Onsen employed an anthropomorphic robot in their tourist information centre.

Even more recently, in November 2022, a prototype of ChatGPT, a chatbot based on the GPT3 transformer model was launched, gathering worldwide interest for its long, articulate answers on a wide variety of topics and the ability to keep the conversation topic going for a notable amount of time. The model, trained by a mixture of supervised and reinforcement learning, was released as a demo by the OpenAI foundation for common use. Microsoft, the multinational corporation of operating system fame, was not far behind, equipping their search engine Bing with an AI chatbot module. Today, it may very well be the case that even the most tech-illiterate person has heard about the technology at least once. The era of human-robot interaction - particularly verbal interaction - is fast upon us and unlikely to go away.

In this work, we seek to explore an area of chatbot research not often explicitly focused on - that of the empathetic chatbot. Large language models such as ChatGPT or LLaMA can get away with little to no focus on empathy training simply because of the sheer scope of their training data and the number of their parameters. Such a path is not viable for smaller architectures.

This thesis focuses on developing an emotion-aware chatbot model aimed at improving conversational experiences through the recognition and appropriate response to users' emotions. The goal of this work is to explore different approaches to integrating emotional information into model training and create an open-domain chatbot model based upon these approaches. We then evaluate the created models based on both automated and human criteria and integrate the highest-ranking model into a simple web application for user convenience.

1 The problem expression

In this work, we:

- **Provide a theoretical grounding in what conversational agents (chatbots) are**, and an overview of different approaches to building said chatbots. We explain the differences between rule-based and AI based approaches and elaborate on their key traits. This is done in sections 2.1 and 2.2 of this work.
- **Provide a more in-depth overview of the AI based approach**, in particular generative models. We focus on its base principles, as well as 3 key phases in generative model evolution - recurrent neural networks, recurrent neural networks with attention mechanism, and transformer networks. This is done in sections 2.3.
- **Focus in detail on the transformer architecture**, its variants and categorization, the advantage of transfer learning with said models, and the T5 transformer in particular. This is done in sections 2.4.
- **Explain the role of empathy in HRI systems**, its effect on user satisfaction, and different strategies of implementing it, citing specific chatbot models implemented in the past 5 years. This is done in section 2.5.
- **Provide an overview of common evaluation methods** suitable for conversational agents. This is done in section 2.6.
- **Design and develop several different chatbot architectures**, focusing on transformer-based generative models incorporating empathy. Using FLAN-T5 as our base, we create 4 main model types - baseline, encoder-aware model with separate emotional classifier, decoder-expressive model with integrated emotional classifier, and an extension of the decoder-aware variant incorporating model history. We train the designed models on two different datasets with incorporated emotion labeling, testing various hyper-parameters and classifier

setups. This is covered by sections 4 and 5.

- **Evaluate the created models** in terms of automated BLEU score metric, as well as human evaluation survey given to volunteers from within our field. Automated evaluation is done in section 5, then summarized and expanded by human evaluation in section 6.

- **Design, develop and implement a web-based application** to serve as an user interface for the most successful among our models. The application provides various functionalities such as conversation exporting and user control over generation style. This is covered by section 7.

- **Create a full programming documentation** of all modules and classes implemented within the project, their functions and parameters. This documentation can be found in appendix B.

- **Create a user manual** including detailed instructions for installing all prerequisites, hardware requirements, everything the average user needs to know about the functioning of the application, and various functionalities and their location within the application. The user manual is located in appendix C.

2 Chatbots - an overview

In this chapter we will explain the the theoretical foundations of chatbots, the main chatbot-building approaches and various popular architectures. We will also elaborate on the role of empathy in chatbots and how it can be implemented. Lastly, we will provide an overview of common evaluation tactics for chatbots.

Human - robot interaction (HRI) is a multidisciplinary field on the intersection of artificial intelligence, robotics, social sciences and humanities. It concerns itself, as the name indicates, with the interaction between humans and robots and how specific design choices or potential novel functionalities affect said interactions.

For the purposes of HRI, interaction can be defined as any communication between a robot and a human. The goal of HRI is to make the robot's behavior as natural as possible [1].

Interaction between humans and machines has significantly increased in the past few decades, spurred on by the growing technological capabilities of modern machines as well as the increasingly widespread successes of automation. Humans are slowly transferring from the role of the controller, a manual operator, to that of a collaborator - a teammate. This can be seen not only in industrial processes, but the social sphere as well - with the increased presence and sophistication of conversational agents.

A conversational agent, often called simply chatbot, is a computer software program designed to human-robot interaction using either text or speech. It can hold conversations with human users - how complex and natural-sounding these conversations are depends on the complexity and architecture of the chatbot itself.

Today's chatbots range from the very simple to the undeniably complex, utilizing a wide variety of approaches in pursuit of the same goal. This goal is seemingly simple, though technologically complex - providing as smooth and engaging user experience as possible [2]. In the case of domain-specific chatbots, especially in fields such as healthcare, an additional challenge is added - the information provided by the chatbot should be factually true.

Chatbot technology has experienced a boom in the past few years, becoming increasingly commonplace across various industries, such as customer service, retail and e-commerce, banking and even healthcare.

In the service industry, web enterprise, and e-commerce, it reduces service costs and simultaneously enables interaction with many customers. Compared to a static content search, a frequently asked questions (FAQ) page, or a list of resources, chatbots offer users novelty, seemingly personalized assistance, and time-reducing, targeted solutions. This is a proposition most, even the least tech-savvy users, find attractive. The human tendency to anthropomorphise [3] only helps - users often see even rudimentary chatbots as friendly assistants instead of soulless computer programs [4].

Chatbots have also found various uses in the healthcare industry. This includes more administrative roles, such as appointment scheduling, health surveillance, and information dissemination [5], as well as more direct roles such as a diagnostic assistant [6] or a therapy tool [7]. During the Covid-19 pandemic, chatbots were also deployed as part of the public health response, mostly as risk assessment screening tool [8].

There are multiple possible approaches to building a chatbot, with different models more suited to different situations. In general, chatbots can be divided into two main categories based on the technology used - *rule-based chatbots* (also called *pattern-matching* chatbots) and *machine learning chatbots* [9].

2.1 Pattern-matching chatbots

The least complex chatbots are simple rule-based programs, capable of limited interaction. This usually consists of directing the user to commonly searched terms, or various parts of the website the chatbot is deployed on. When presented with a more complex query, the user is typically redirected to a human operative.

These chatbots are widely used and a part of everyday reality in a wide range of fields - nearly every company can benefit from implementing a simple chatbot assistant on their web-page or app.

A rule-based chatbot works by executing pre-determined actions from a back-end “playbook” - a rule-based database. User input is matched to a rule pattern using a **pattern matching algorithm**. In a more simple case, the user does not type their own input. Instead, a limited number of prepared prompt options is offered. A more complex case involves the user writing their own prompt and the program recognizing keywords or groups of keywords present in the user query. In both cases, the chatbot algorithm then selects a predefined response from a pool of potential replies.

These chatbots are not capable of creating new responses, merely selecting them from a database. This knowledge base is hand-coded and organized into conversational patterns by the developers[2]. Naturally, the more extensive the chatbot’s knowledge base is, the more forms of user input it can successfully respond to. Typical rule count for a rule-based chatbot is up there in the thousands or tens of thousands of conversational patterns [9].

This also makes pattern-matching chatbots suited mainly to *closed domain conversation*, that is, conversation within a specific narrow field. There they can provide informative answers even on highly technical topics, if such topics are included in their rule-base. Unexpected input will, however, receive a tangentially related answer at best, and a completely irrelevant one at worst - should such input be allowed in the first place.

Aside from inflexibility, rule-based chatbots have an additional limitation in robustness. If the user is not limited to a select number of responses, the chatbot can easily fall prey to spelling and grammatical mistakes present in user input [10]. Different users have different typing styles - assuming a perfectly grammatically correct reply won't always reflect reality.

Additionally, most rule-based models only base their responses on the last user utterance as an isolated data point (single-turn conversation), instead of the whole discussion like a human might (multi-turn conversation). This shortcoming is not limited only to pattern-matching chatbots, but it is certainly more prevalent in this category. While some rule-based models are capable of multi-turn response selection, taking into account previous parts of the conversation, they are comparatively very rare [10].

Rule-based chatbots, while extremely common, do not meet the definition of artificial intelligence by themselves, as they are incapable of retaining or learning from new information. For more complex tasks, machine learning approaches are often preferred, as pattern-matching chatbots come across as formulaic, repetitive and lacking of human touch. Additionally, no memory of past responses can lead to conversational loops [2].

Rule-based chatbots have a large commercial advantage though - they are comparatively cheap and easy to implement, often sufficient enough for simple commercial purposes, and utterly predictable in their behavior after deployment.

All of the earliest examples of chatbot technology were rule-based chatbots. Common scripting languages for such chatbots include **AIML**, **RiverScript** and **ChatScript** [9].

2.2 Machine learning chatbots

Machine-learning chatbots, or *AI chatbots*, are complex programs based on *natural language processing* (NLP), sometimes accompanied by sentiment analysis. They are capable of significantly smoother and more natural conversation, both with human users and other chatbots. While once comparatively rare outside of academia, recent advancements in text generation have seen them becoming more common in the commercial sphere as well.

Instead of depending on a predetermined number of possible queries with hard-coded outcomes, machine learning chatbots analyze the underlying sentence structure to establish probable intent and language entities present in the query. This is accomplished with the aid of NLP techniques.The chatbot then composes a tailored answer based on knowledge gained from pre-existent data. This answer can be constructed based on just the last conversational turn, a given number of turns, or the entire conversation. Compared to pattern-matching chatbots, this leads to a more logically consistent and smooth conversation with less conversational breakdowns.

Unlike pattern-matching chatbots, AI chatbots also possess the key trait of artificial intelligence - *the ability to learn from experience.* As such, AI chatbots require a training period to develop a basic level of competency. After that, however, they have the potential to improve continuously with each query they receive for the rest of their deployment.

AI chatbots are well-suited to open-domain conversations, more so than their rule-based counterparts. They are not always superior, however - when it comes to domain-specific interactions, a simpler rule-based chatbot may be just as effective, or even better.

Additionally, AI chatbots require a large conversational corpus to train on. Finding a good dataset may be difficult by itself, especially for less common tasks or specific domains. The performance of an AI model is heavily dependent on what data was used to train it, and while a bot deployed to serve a domain-specific function may fail in its task if it was trained on too generic data, a more broad, conversational agent may do badly if trained solely on domain-specific conversations.

An intelligent chatbot faces two main tasks:

1. Correctly processing user input

2. Returning a relevant reply.

2.2.1 Processing the input

Input pre-processing in AI chatbots consists of several key steps, some more common than others. It must be noted that not every step is present for every model - indeed, some may be counterproductive for certain architectures.

The universal steps include [11]:

1. Tokenization - the process of breaking down the input sentence into individual works - tokens.
2. Text cleansing - the removal of any unwanted special symbols from the input string. This may include emojis, URL links, or other special characters. Most commonly, the removal of punctuation also falls within this step.
3. Standardization - converting the input into a standard format, such as all lowercase and without diacritics.
4. Stop-word removal - common words such as prepositions, articles and conjunctions do not add much informative value to the text. As such, they are typically removed.
5. Feature extraction - the conversion of processed text into features usable by a machine learning model - a tensor of numerical values. In modern chatbots, this is typically accomplished using word embeddings which map words to dense vectors in multi-dimensional space. A chatbot model may use its own embedding layer, or receive already embedded input created for example by GloVe, FastText, or Word2Vec.
6. Padding - this step involves adding zeros to the feature matrix so that all input vectors have the same length, if it was not already done in previous step. The majority of machine learning algorithms requires for all inputs to have the

same input shape.

The are, additionally, several optional steps that could be conducted after stop-word removal but before feature extraction. This is because several of these steps can generate additional usable features.

These include [10]:

- Named entity recognition (NER)- the process of identifying entities such as names, locations or dates. NER is usually referred to as a slot-filling problem and consists of sequential tagging of parts of a sentence. An entity-tagging model may be pre-trained on a manually annotated corpus of conversations.
- Part of speech (POS) tagging - the labeling of individual words with their respective part of speech designation, such as noun, adverb, verb etc.
- Spell correction - the identification and correction of grammatical mistakes in the input text.

Any and all of these may be present in a given model depending on intended purpose and complexity.

2.2.2 Producing a reply

The second task an AI chatbot faces is the production of a viable reply. This step has had various executions in the past. Generally, we can talk about **retrieval** and **generative** models.

In both approaches, the usage of neural networks is standard. Recurrent neural networks (RNNs) in particular were - and still are - very popular, as they are well suited to working with sequential data like text. In particular, RNNs with LSTM layer - Long Short Term Memory layer - have be utilized to great effect [9]. A newer neural network architecture which has also become very common is the Transformer network, utilizing an attention mechanism in place of recurrent connections.

While they use similar technologies, the true difference between retrieval and generative models is what purpose these technologies serve.

A *retrieval-based model* does not create a new reply “on the spot” - instead, it refers back to a corpus of appropriate replies. From this database it then chooses a reply judged most fitting the the content of the user query. Depending on the approach used, it can utilize either shallow or deep learning methods.

The reply selection process can rely on simple keyword matching, but also on more complicated approaches such as the use of intent embeddings during training or named entity recognition.

Generative models do not rely on selecting pre-defined responses from a database. Instead, a brand new reply is synthesized for any given input using natural language generation (NLG) techniques. This is done one word at a time, by sampling from a learned probability distribution of possible next words.

As generative models form the core of our chosen approach to the problem presented in this thesis, they will be discussed at length in their own section.

2.3 Generative models

Generative chatbots are, at its core, an example of a sequence-to-sequence model - a very well known and widely utilized architecture.

Sequence-to-sequence models, commonly abbreviated as *seq2seq* models, form the core of many a state-of-the-art approach in a wide variety of NLP tasks. These tasks are not limited to generating responses in dialogue systems but also encompass machine translation, speech recognition, text summarization, and image captioning, among others. They were first introduced by Sutskever et al. [12] in 2014.

The name serves as a fitting description of these models’ basic working principle - a seq2seq model takes a variable length text sequence as input, and produces a variable length text sequence on the output. In this, they stand in contrast to the typical neural network data-flow with singular numerical output or class probability vector. These models typically have an encoder-decoder architecture.

The encoder serves to map the variable length source sequence to a *fixed length vector representation*, which the decoder then maps to a, once again, variable length target sequence, as seen in Fig.$2-1$.

The fixed length vector representation created by the encoder is called a **context vector**. It is a compact summation of the input sentence itself - all the key information extracted, ready to serve as a basis for generation.

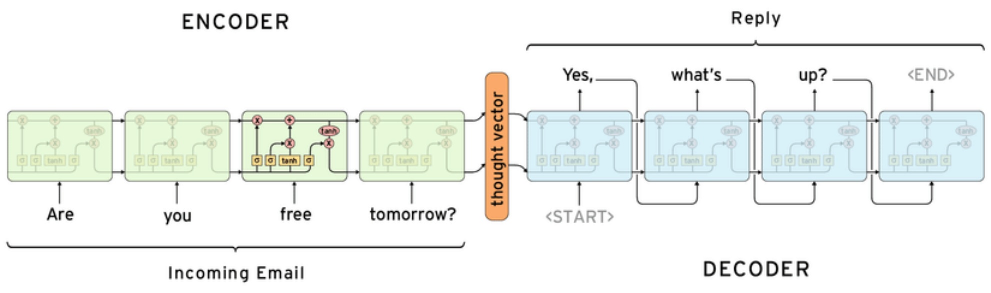


Fig. $2-1$ A sequence to sequence model

The decoder then generates the output sequence one token at a time - the output at step t is the t-th word in the sentence being generated. At each step, the decoder takes as an input *the previous output token* as well as the encoder context vector, and predict the probability distribution over possible next symbols.

The initial input of the decoder cell is a special $\langle START \rangle$ or $\langle SOS \rangle$ token, and the sequence being generated continues until the decoder predicts a second special token - $\langle END \rangle$ or $\langle EOS \rangle$. This enables the production of variable length output.

Before the arrival of transformers, which will be discussed in the next section, the most common method of implementing this approach entailed employing a pair of *recurrent neural networks* as an encoder-decoder.

Today, the encoder and decoder need not be recurrent neural networks - or if yes, they are often modified in various ways. Indeed, some of today's models do not use both encoder and decoder at all - most notably decoder-only transformer models.

Training

During training, the input at time step $t + 1$ is **not** the previous predicted token $y_{t_{pred}}$, as the model at the beginning will output more or less random probabilities. Using the predicted token has been shown to lead to lengthy convergence and model instability. Instead, the actual true token $y_{t_{true}}$ is used, in a method commonly known as **teacher-forcing** [13].

The loss function commonly used for this purpose is the **cross-entropy loss**. Given a source sequence $x = (x_1, x_2, ..., x_n)$ and target sequence $y = (y_1, y_2, ..., y_m)$, at a given time step t the model predicts a probability $p(t) = (*/y_1, ..., y_{t-1}, x_1, ..., x_n)$. The $*$ in this equation is a placeholder for model's prediction at time step t, given the previous predicted tokens and the input sequence.

It is desirable that the model assigns probability 1 to the correct token and zero to the rest, i.e. our target probability is $p^* = onehot(y_t)$. Cross-entropy loss for the target distribution p^* and the predicted distribution p is defined as

$$Loss(p, p^*) = -p^* \cdot log(p) = -\sum_{i=1}^{N} p_i^* \cdot log(p_i)$$

where N represents the total number of classes in the classification problem. Given that only one of the summed probabilities will be non-zero (and equal to 1), the equation can be further simplified to

$$Loss(p, p^*) = -log(p_{y_t}) = -log(p(y_t/y_{<t}, x))$$

where y_t is the correct class index and $y_{<t}$ represents the previously predicted classes [14].

Loss is calculated on the predicted outputs for each time step and the errors propagated trough the network using back-propagation trough time. Both encoder and decoder are trained jointly, in order to maximize the conditional probability of the target sequence given the source sequence [15].

Inference

During inference, ground truth tokens are no longer used. Instead, the generated token at time t serves as the input at time step $t + 1$, resulting in the generation of the $t + 1$-th word. There are multiple approaches to which word gets picked as the next output token.

In the case of **greedy decoding**, the token picked at each time step is the one with the highest probability. The model does not consider the effect of its decision on future symbols. This is the simplest, efficient decoding strategy, and least computationally exhaustive - but may lead to sub-optimal results [16].

Alternatively, **beam-search** approach keeps track of multiple token possibilities for each time step, continuing each of them down the path and picking the top N. The parameter N is called beam size and typically ranges from $\langle 4, 10 \rangle$. While often leading to better results than greedy decoding, beam search is also more computationally expensive and sometimes prone to getting stuck in local optima [16].

Sampling-based decoding introduces a degree of randomness to the generation process in the hopes of making the output more diverse. The decoder randomly samples the next symbol from its predicted probability distribution, rather than selecting the most likely symbol.

The most popular approach within this category is *Top-K Sampling*, which limits random choice to only K most promising token candidates. Compared to standard sampling, the model is less likely to generate unlikely or irrelevant words. One potential drawback of Top-K sampling, however, is that it can lead to inconsistency in the output. This is because the set of top K words can vary greatly depending on the input and the model's current state [16].

2.3.1 Recurrent models with fixed length context vector

A recurrent neural network (RNN) is a type of neural network designed for processing sequential data and thus uniquely suited to tasks where the order of the input matters. The output produced by RNNs is dependent on all prior elements of a sequence. As such, these networks are quite common in natural language processing, where the target reply depends not only on the current token, but a certain broader context of conversational history.

A recurrent model consists of a hidden state **h** and an optional output **y**, which takes form of a variable length sequence $\mathbf{x} = (x_1, x_2 ... x_n)$. At each time step, the hidden state from the previous timestep is combined with the input of the current one and passed trough a nonlinear activation function [17]:

$$f(Wh_{t-1} + Ux_t)$$

which can range in complexity from simple element-wise logistic sigmoid to an LSTM unit [18]. The leters U and W in the formula denote weight matrices for hidden-to-hidden and input-to-hidden connections respectively.

This creates the hidden state h_t of the current time step. The output is then given as [17]:

$$y_t = Vh_t$$

where V is the weight matrix for hidden-to-output connections. The current h_t is subsequently combined with the input at time t+1, producing hidden state h_{t+1} and output y_{t+1}, and so forth. The number of time steps is equal to the length of the sequence.

A recurrent model learns to predict the next symbol in a sequence by learning the conditional probability of such a symbol appearing - probability distribution $\mathbf{p}(x_t / x_{t-1}, ..., x_1)$. In seq2seq models, this problem is broadened, setting out to learn the probability distribution over a variable length sequence in relation to another variable length sequence $\mathbf{p}(y_1, ..., y_n / x_1, ..., x_m)$, where one length may not correspond to the other.

While sufficient for other tasks, for natural language processing standard RNNs with sigmoid activation are not good enough - the longer the input data sequence, the poorer the generated result. This is due to the destructive influence of what is called *the vanishing gradient problem.*

Vanishing gradient is an issue deep learning models face in certain conditions. Most neural networks update their weights using a back-propagation algorithm -that is, starting from the last layer in the network and iterating to the first.

This algorithm, however, uses a recursive formula involving a chain of multiplied partial derivations, one of them the derivation of the activation function. If the activation function has a small range - like sigmoid with range $\langle 0, 1\rangle$ or tanh with range $\langle -1, 1\rangle$ - the derivation is always a decimal number smaller than 1. This is not that much of a problem for shallow networks. However, the deeper a network is, the smaller these multiplied derivations get, especially for the first few layers. In extreme cases the gradient can reach zero and weights don't get updated at all.

As a result, in feed-forward neural networks activations such as sigmoid and tanh were replaced by ReLU and its variants. In recurrent neural networks, popular choice shifted towards its own form of vanishing gradient resistant solution - LSTM units or GRU units.

LSTM recurrent unit, also called a *memory cell*, is an extension of a standard recurrent unit. It is extended by adding a **cell state** - while the hidden state serves to simulate short-term memory, the role of the cell state is to play the long term one.

Access to the cell state is controlled by a series of *gates*. The hidden state h_{t-1} from time step $t-1$ and current input vector x_t are combined before passing through these gates. There are three types of gates. A *forget gate* decides which long term information should be forgotten, an *input gate* decides which new information should be added to the cell state, and an *output gate* helps form the new hidden state. Each gate receives its own copy of the combined state to work with. The full information flow can be seen in Fig. 2 − 2.

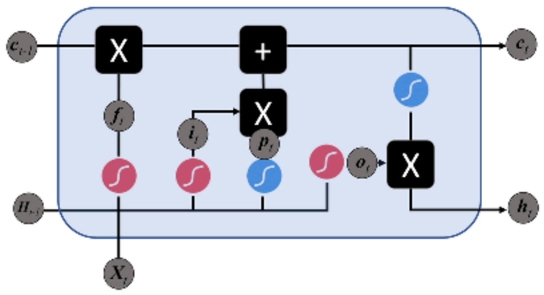


Fig. 2 − 2 LSTM cell information flow [19]

LTSM units were first introduced in 1997 [20] and have been a common choice for seq-2-seq models since the very beginning - the earliest pioneering paper on seq2seq models by Sutskever, et al. [12] utilizes LTSM.

Gated Recurrent Units (GRU) were first proposed by Cho et al.[15] in 2014, in one of the early bits of research on seq2seq models. GRU is simpler then the older LSTM unit, consisting only of two gates - the *reset gate* and the *update gate*. Additionally, it does not maintain a cell state - information is transferred solely via hidden states, as shown in Fig. 2 − 3.

As has already been mentioned, until now information in the standard seq2seq model was passed from the encoder to the decoder in the form of a fixed length context vector - the final hidden state of the encoder. This final hidden state then served as the zeroth hidden state for the decoder network.

2.3.2 Time to pay attention

The standard encoder-decoder architecture described in the last section is not without it's drawbacks - because the decoder depends so heavily upon the final state

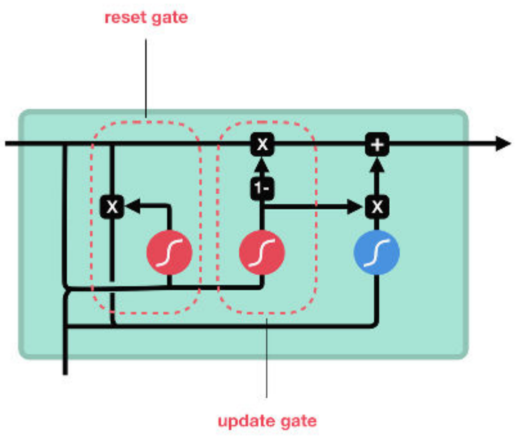


Fig. 2 – 3 GRU cell information flow [19]

vector of the encoder, dealing with long sentences can be challenging.

By the time a sequence reaches it's end, the initial context may be completely lost. Passing on information in the form of a single fixed-length vector is clearly not ideal, and, indeed, has been shown to lead to a rapid deterioration in terms of performance the longer an input sequence gets [18].

To address the challenge of capturing long-term dependencies in sequence-to-sequence tasks, **attention-based models** were introduced by Bahdanau et al. in 2014 [21], and further developed by Luong et al. in 2015 [22]. While originally used for machine translation, this approach has since been applied to a wide range of tasks, such as summarization, image description, and dialogue generation.

The key advantage of attention-based models is their ability to selectively focus on different parts of the input sequence at different stages of generating the output, thus preserving sentence context more effectively.

Instead of using a fixed-length context vector, the models pass all encoder hidden states to the decoder. At each decoding step, the context vector is dynamically created as a weighted combination of these hidden states, enabling the decoder to selectively attend to the most relevant parts of the input sequence. This results in more accurate and flexible modeling of long-term dependencies [21].

To achieve this, attention-based models use an extension of the encoder-decoder architecture called **the attention mechanism**. During decoding, the mechanism selects a subset of context vectors from the encoder, based on the most relevant information in the source sentence, and creates the weighted context vector. This allows the decoder to effectively focus on the most informative parts of the input, resulting in improved performance across various sequence-to-sequence tasks.

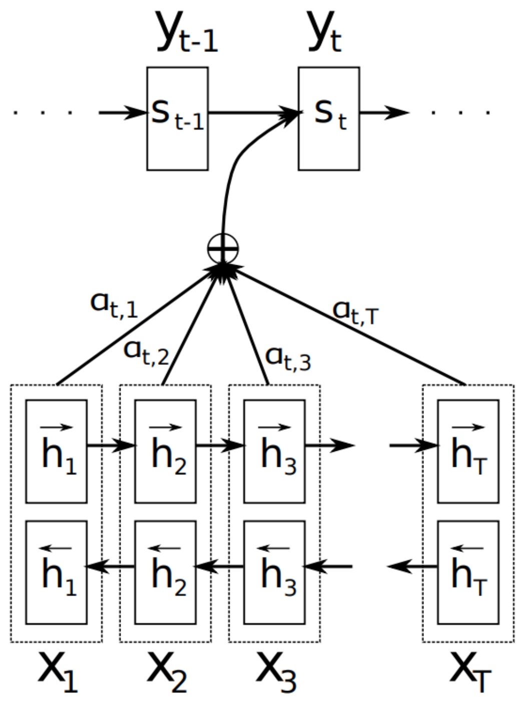


Fig. 2 − 4 Attention mechanism as proposed by Bahdanau et al. [21]

At each decoder step, the attention mechanism performs the following steps, as illustrated in Fig. 2 − 4:

1. Receives the decoder hidden state h_t from the previous time-step, as well as all the encoder states $(s_1, s_2, ..., s_n)$. This is **the attention input**.
2. Computes an attention score for each encoder state s_i, expressing the relevance

of that time step's state in relation to the hidden state h_t. This is done by applying an attention **scoring function** to these inputs, resulting in a vector of scalar score variables.

3. Applies softmax activation to these scores, creating a probability distribution of attention weights, also called **alignment scores**.

4. Computes the **attention output** as a weighted sum of encoder states and alignment scores for each state [21].

There are multiple forms the scoring function may take. Luong et al., in their research [22] outlined three main approaches: dot product, general product, and concat product. Concat product was the scoring function previously introduced by Bahdanau et al. [21]. General product, on the other hand, was introduced by Luong et al. themselves. The equations for these functions can be seen in $2-5$:

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

Fig. 2 − 5 Different attention scoring functions [22]

The original Bahdanau and Luong implementations, however, varied in more than just the scoring function used.

The *Bahdanau model* [21] used bidirectional RNN as an encoder (essentially two RNNs where one reads the input forward, one backward), concatenating the two hidden states for each token. The attention mechanism then received these concatenated hidden states. The scoring was implemented via a multi-layer perceptron with tanh activation function. Attention was then applied in-between decoder steps: both the hidden state h_t and the generated attention context vector c_t were passed to the next step $t + 1$ of the decoder.

The *Luong model* was simpler - the encoder was unidirectional and general product was used as the scoring function. The attention was applied after generating the hidden state h_t for step t, but before prediction for said step.

After using the hidden state to generate the attention vector, these two vectors were combined using the concat approach, identical equation-wise with the Bahdanau attention. It's worth noting that instead of a "global attention" computed as a weighted sum over all source hidden states, Luong et al. implemented "local attention" which focused only on a small subset of source states for each target token.

Both approaches produced quality results and were considered state-of-the-art at their time. RNN-based seq2seq models with attention mechanism served as the main approach for text generation tasks for a number of years - until the introduction of transformers.

2.4 Transformers

Currently, transformer models can be considered the standard approach to seq2seq tasks and text-focused tasks more broadly. First introduced in 2017 by Vaswani et al. [23], the transformer departs from the before-then predominant architecture of two recurrent neural networks, opting to leave the recurrence behind entirely and use solely attention in both encoder and decoder.

The reason for this is simple - natural language processing tasks can be extremely time-intensive to train on using recurrent architectures. As recurrent neural networks "read" a sentence one word at a time, the contextual meaning of any particular word cannot be learned until the entire sequence is processed, with a time complexity of $O(n)$, where n equals the length of the sequence.

It is clear that this can become time intensive rather quickly, especially with longer sentences, as NLP datasets by necessity contain thousands of data samples. In contrast, a transformer model reduces the time period necessary to a constant number of steps, essentially viewing the entire sentence at once by passing it in parallel. This allowed the first transformer model to significantly speed up its training process and achieve (at that point) state-of-the-art results in machine translation in as little as 12 hours of training.

2.4.1 Vanilla architecture

In its originally proposed variant, the model takes the form of an encoder-decoder pair, utilizing two main forms of attention: **self-attention** and **encoder-decoder attention**, both realized via the **multi-headed attention** mechanism.

Layers within the network are organized in identical stacks, repeated n times for both encoder and decoder. Vaswani's transformer had 6 stacks in the encoder and 6 in the decoder [23], today, larger numbers are also common.

In the **encoder**, each layer is composed of 2 sub-layers: a multi-headed attention layer fulfilling the role of *encoder self-attention*, and a simple, feed-forward network. Each sub-layer is followed by layer normalization. Additionally, each sublayer block contains a residual connection going around the sub-layer itself, such that the input for the normalization layers is $x + sublayer(x)$.

In the **decoder**, each layer is composed of 3 sublayers. These sublayers are multi-headed attention layer in the role of decoder self-attention, multi-headed attention layer as encoder-decoder attention, and afeed-forward network. Identical to the encoder, each sublayer block contains layer normalization and a residual connection going around the sublayer itself.

As can be seen in Fig. $2-6$, during training the encoder receives the prompt (tokenized sentence to be translated / generated from / summarized...) as input. Meanwhile, the decoder receives the golden reply (the pre-processed ground truth), shifted right by one position. This is done so that the model would not learn to rely on future ground truth tokens - information it should not have yet.

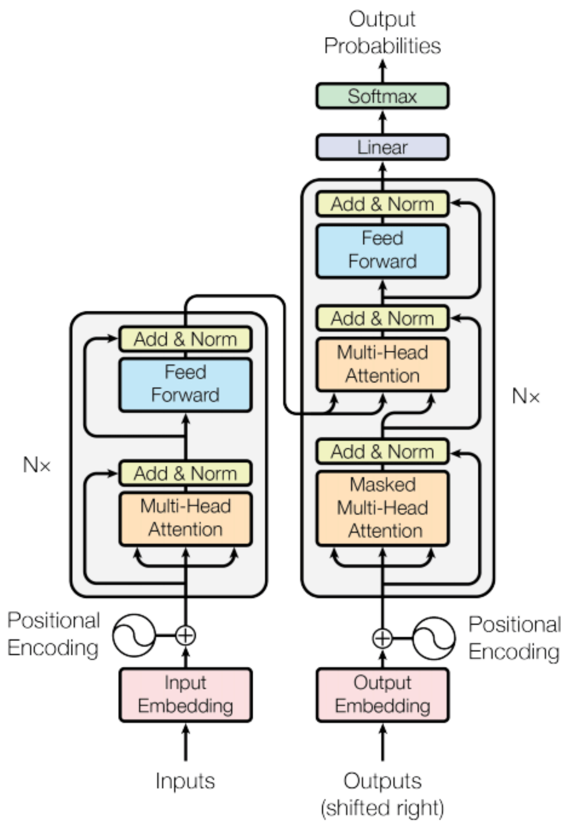


Fig. 2 − 6 Encoder (left) and decoder (right) of a transformer model, Vawani et al.[23]

Also important to mention is the presence of a **positional encoding** layer for both the input and golden reply sequences. A sentence's meaning is by its very nature dependent on the order of its elements. A reorganized sentence may change meaning, or lose it completely. As transformers no longer utilize a recurrent neural network which would process input tokens in order, standard embedding (where each word in a sentence is represented in the form of an embedding vector) is no longer enough.

Positional encoding represents positional information of individual tokens in relation to other tokens. It has the same dimensions as the input embedding, which allows these two embeddings to be summed. Thus modified input embedding carrying positional information can be then passed to the model.

The original 2017 paper [23] used sine and cosine functions of different frequencies to perform the encoding. Some transformer models use the same strategy, but various other positional encoding schemes, including fixed encoding, are also used.

The attention mechanism

Transformer attention is very similar to the previously described attention mechanism in recurrent neural networks. It is realized via an attention function, which takes three parameters - the query matrix Q, keys K, and values V.

The query in this case represents a single token in the sequence - a single word representation. The query matrix has all the queries packed into it, allowing for simultaneous computation across queries. Keys and values both consist of all the vectors of all the sequence tokens.

However, the original transformer paper used a **scaled dot product attention** - a modification of the previously shown dot product attention. First the attention weights are obtained as:

$$\alpha = sotfmax(\frac{QK^T}{\sqrt{d_k}})$$

Then, the output is computed as a weighted sum of values V and these weights:

$$attention(Q, K, V) = \alpha * V$$

The key modification is the scaling factor present while generating weights. The dot product, before applying the softmax activation function to get probability distributions, is scaled by the square root of d_k, which represent both query and key length [23].

The purpose of the scaling is to prevent degradation of dot product attention performance for larger lengths d_k, potentially caused by large dot product values pushing the softmax function into regions with very small gradients.

Multi-headed attention parallelizes this process into multiple attention mechanisms that work side by side, as shown in Fig. $2-7$. Each individual attention function is carried out on a different linear projection of Q, K and V, which are created by multiplying the Q, K, and V matrices with corresponding weight matrices learned during training. These individual results are then concatenated and once again projected. The number of parallel attentions executed at the same time can also be referred to as the number of **heads** - a hyper-parameter defined when building the network itself [23].

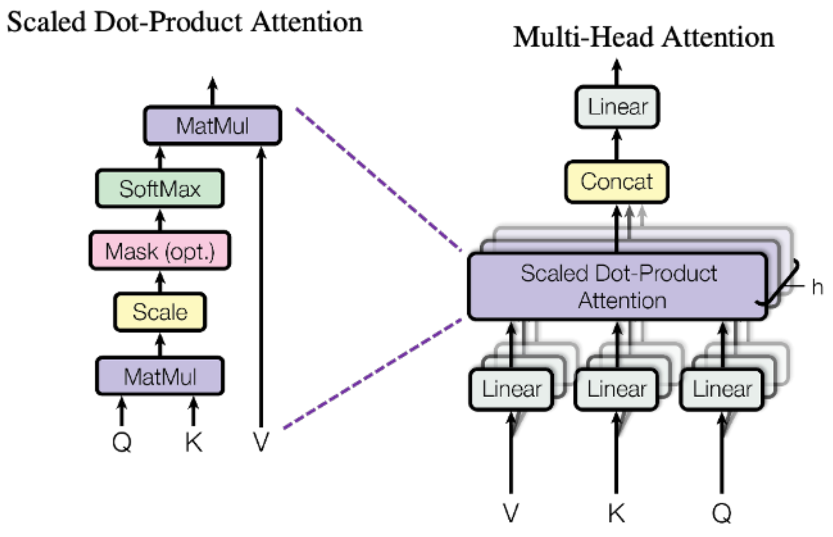


Fig. 2 − 7 Multi-head attention mechanism [23]

Self-attention and encoder-decoder attention

The role of **self-attention** is to determine the relative importance of a given token in relation to the other tokens in the sequence. This importance is represented in the form of an attention vector. Individual attention vectors are generated for every word in the sequence.

In the case of encoder self-attention, all the queries, keys and values come from the same place - the output of the previous encoder layer [23]. Each encoder position can attend to all the positions in the previous layer.

Decoder self-attention, likewise, takes all its inputs from the previous decoder layer output, allowing all decoder positions to attend to all previous decoder positions up to that point. However, decoder self-attention is masked by setting all values corresponding to illegal leftward connections in the softmax input to $-\inf$ or 0 [23].

The reason for this is simple - the role of the decoder is to generate the next sequence token based only on the previous words in the sentence. Therefore, it cannot be allowed to know the actual ground truth token while predicting.

Encoder-decoder attention enables each position in the decoder to attend to all positions in the input sequence. In the context of machine translation, this would be where the original language and the translated-to language mapping happens. In the case of a chatbot model, this is the mapping between the user utterance and the bot reply. Encoder-decoder attention functions essentially the same as the classic attention mechanism described for recurrent seq2seq models does.

Position-wise feed-forward network -
The last notable part of the transformer architecture is the presence of the position-wise feed-forward sublayer, present in all of the encoder and decoder layer blocks.

Each position in the model input sequence is represented by an embedding vector. The term “position-wise” means that, in contrast to standard feed-forward networks, each of these embedding vectors is processed by the network independently.

The feed forward network uses two linear transformations with a ReLU activation in-between, applied separately but identically to each of the input positions like so [23]

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2$$

In the formula above, x represents the layer input. W_1 and W_2 are the weight matrices and b_1 and b_2 the biases for the first and second linear transformation, respectively.

Its main role is to transform the attention vectors received from the previous sublayer into a form accepted by the next layer in the stack, and its functioning can be described as equivalent to a series of two convolutions with kernel size = 1 [23].

2.4.2 Model variants

A large number of transformer model variants exist, implementing diverse modifications to different layers of the original transformer architecture. Accordingly, multiple ways of categorize them according to those modifications exist.

In our work we will focus on one main categorization - which parts of the original transformer a given model uses. Using this taxonomy, we can divide transformer models into **encoder-decoder** models, **encoder-only** models, and **decoder-only** models.

Encoder-decoder models are the standard and oldest configuration. These models follow the original architecture proposed by Vaswani et al. [23], with both an encoder and a decoder. The advantage of these models is that they are capable of both language understanding and language generation, enabling greater freedom when creating architectural modifications [24]. An example of this model variant is the T5 universal transformer [25].

Encoder-only models are transformer variants that exclusively use the encoder part of the architecture. These models are specifically designed for language understanding tasks, and do not typically generate text output. Instead, the models output the raw hidden states of the encoder, which can be employed for various applications that require sentence embeddings, as discussed in [24]. An example of such a task may be sentence similarity, which has a whole category of models for itself - sentence transformers. These models append a pooling layer on top of the encoder-only architecture. The most popular encoder-only model is BERT [26].

Decoder-only models are suited for language generation but not language understanding. Only the decoder part of the original transformer is used. The output of such a model is generated text, often a continuation of a prompt [24]. A very well known example of these models is the GPT family and its newest members, ChatGPT (GPT 3.5) [27] and GPT4 [28].

2.4.3 Transfer learning and transformers

Transfer learning - the usage of previously pre-trained models or their parts when solving a given task - is common in the field of deep learning as a whole. It is, however, near ubiquitous when it comes to natural language processing and transformer architectures especially. This has several reasons.

First of all, transformers are very large models - their weight counts number between hundreds of millions and hundreds of billions. Despite the computational speed advantages they have when compared to recurrent models, they are still rather lengthy to train.

Additionally, getting a large enough corpus of labeled text data for a given task - especially when it is a rather niche or domain-specific task - can be problematic and costly. Such data needs to be labeled by a human operative.

However, here large language models have a sizable advantage over their computer vision counterparts - it does not, necessarily, take labeled data to teach a model basic language understanding. Transformers models are probabilistic - and the basic structure of language with probabilities of certain word combinations occurring can be inferred from unlabeled text. Studies suggest that transformers pre-trained on large quantities of data using universal language modelling tasks can learn basic language understanding without the need for any supervised training [29].

This allows the programmer to start with a model that already possesses general-purpose knowledge of understanding text. This includes low-level understanding of grammatical structures and the meaning of words, as well as high level abstract capability of understanding when something is unlikely or impossible given context (a car wont fit in a handbag) [30].

Popular transformer models, such as the GPT family, T5, BERT and its variants, as well as many others, can be accessed already pre-trained on through various online repositories. These models were all trained using extremely large datasets on a wide variety of preparatory language modelling tasks.

Examples of a few common language modelling pre-training tasks include [30]:

- **Masked Language modelling** - in this task, a given percentage of words within the training sentence is replaced by a mask token (e.g.[MASK]). The model is then trained to correctly infer the missing word or string of words. Used by BERT and some of its variants (RoBERTa, DistilBERT, AlBERT). In a slight modification used by T5 (in addition to supervised downstream tasks).
- **Masked language modelling with sentence shuffling** - whole spans of words may get masked in this MLM task variation. Additionally, sentences are shuffled beforehand. Used by BART [31].
- **Next sentence prediction** - also used by BERT, this binary pre-training objective tasks the model with learning whether two sentences are or are not consecutive.
- **Permutated language modelling** - a modification of masked language modelling used by, for example, XLNet. A random permutation is defined for the input tokens, and masked tokens are inferred one by one in the permutation order using antecedent tokens.
- **Causal language modelling** - all models from the GPT family are pre-trained using this task. The model is trained to predict the next token given a sequence of previous tokens.

2.4.4 T5 Transformer

The T5 transformer (Unified Text to Text Transformer) by Raffel et al. [25] is a transformer-based model first proposed in 2020, with the goal of re-framing all text-processing problems into a unified text to text format (Fig. 2 – 8).

This approach allowed the authors to treat all NLP tasks essentially the same, using the same model, objective, decoding approach and training procedure, including loss function and hyper-parameters. The concept of unification of NLP tasks is not new - previously proposed methods included casting everything as question answering, span extraction or language modelling [25].

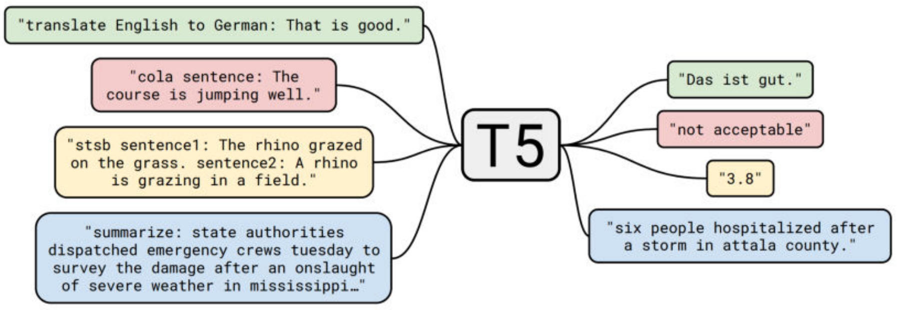

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Fig. 2 – 8 T5 working concept [25]

The model was pre-trained on the sizable 700GB C4 dataset (Colossal Clean Crawled Corpus) and achieved state-of-the-art results on a variety of NLP benchmarks. It is also flexible enough to allow for fine-tuning for basically any downstream NLP task the user might require.

The T5 architecture closely resembles the originally proposed Transformer architecture [23] as described in the previous section of this thesis. A few modifications, however, are worth mention.

First of all, the normalization used is simplified compared to the original transformer model, with no additive bias. Secondly, instead of using 6 encoder layer blocks and 6 decoder layer blocks, T5 uses 12 of each, making it a larger network.

Lastly, the positional embeddings used are no longer sine and cosine functions, but relative positional embeddings as proposed by Shaw et al. [32] and Huang et al. [33]

in 2018.

Relative positional embeddings create a different learned embedding for each position based on the offset between the key and the query input parameters of the self-attention mechanism.

In the T5 transformer, this method is further simplified. Each embedding is a scalar which gets added to the corresponding logit for computing attention weights. The embeddings are also shared across all layers, although each head in the multi-headed attention mechanism uses a different embedding than the other heads. A fixed number of embeddings is learned, corresponding to a range of possible key-query offsets.

The training objective of T5 is masked language modelling. Masked language modelling constitutes a de-noising objective - the goal is to predict missing or corrupted tokens located somewhere in the input. In the case of T5, 15% of input tokens for each data points were randomly sampled and dropped. All consecutive spans of missing tokens got replaced by a single sentinel token - a special token not corresponding to any word in the vocabulary. This process is shown in Fig. 2 – 9.

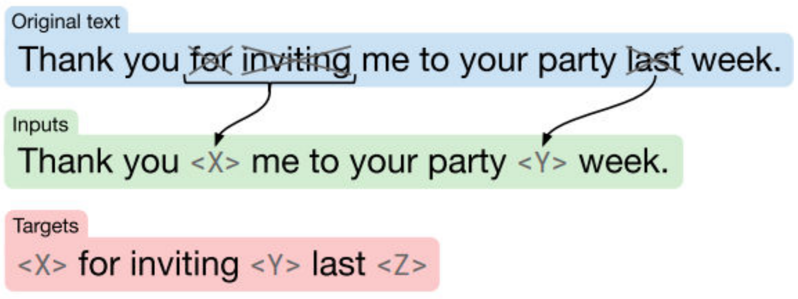

Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Fig. 2 – 9 T5 unsupervised training objective [25]

Tasks and fine-tuning - the T5 model recognizes a number of downstream supervised tasks to choose from when fine-tuning for a new dataset. In the original paper [25] the researchers trained the model on 18 different downstream tasks, including text summarization, sentiment analysis, translation from English to German, French, and Romanian, question answering, sentence similarity, and more.

The model is instructed to perform a particular task by adding the relevant **prefix** to the start of the input sequence (e.g. “translate English to German”). This may be an already defined task or any arbitrary text string, not previously known by the model - it is only important that every data point presented includes this prefix, and that the prefix remains the same across data points. Picking a previously unknown prefix defines a new task.

Each input sample given to the model should thus take the form

```
 <prefix>: <input_text> </s>
```

with target

`<output_text></s>`.

FLAN-T5 is a later variation of T5, first introduced by Chunk et al. in 2022 [34]. The model architecture remains unchanged - what makes FLAN-T5 better at generalizing is the number of additional tasks it has been trained on. This is a rather large number - over 1800 fine-tuning tasks have been used by the researchers. While FLAN-T5 does not strictly require a prefix to function, considering the number of different situations the model has been trained to handle, a well defined prefix may help the model focus on the desired type of output.

2.5 Enter: empathy

This section describes the importance of implementing empathy in conversational agents, as well as various methods of doing so.

2.5.1 The role of emotion in human robot interaction

While increasingly widespread, the integration of robots into environments where their tasks would usually be undertaken by human actors is still to some degree clunky. Despite the advancements of chatbot technology, machine learning models in particular, most chatbot models cannot compare to a human conversationalist.

A not insignificant part of this can be attributed to the lack of natural flow in HRI due to the robots inability to acknowledge emotional subtext. The user's negative feelings during such an interaction, if present, cannot be discounted, as they can lead to negative reactions and decreased satisfaction.

The importance of leveraging emotions in robot-human interaction cannot be underestimated. *Affective computing*, a term coined in the mid 90s by Picard [35], is a vital area in the field of HRI. This sub-field concerns itself with the study and development of systems capable of recognizing, processing, and furthermore simulating, human affects.

Research shows that such practice is beneficial to value co-creation, user satisfaction and acceptance towards new technologies when interacting with robotic agents[36]. Multiple studies have suggested emotion may be the most crucial factor in successful human-robot interaction [37]. Computer initiated emotional support, such as the show of active listening, empathy and sympathy can help users in overcoming and managing negative emotional states [38].

This seems to hold true even for longer interactions, as another research shows - users tend to rank emotional agents higher in terms of appeal, trustworthiness and respectfulness [39].

Emotions form a cornerstone of human experience -emotions such as anger, disgust, joy or sadness have significant influence on both the inner world of the individual (facilitating cognitive processes and helping us make sense of the world) and on external behavior and motivations behind actions.

What more, it has been proven [36] that emotions are "contagious" - we are prone to mirroring each others emotional states as expressed via facial expressions, tone of voice, gestures and body language. This applies not only with other humans, but other primates and mammals more generally.

This phenomenon is reflected in service context as well - emotions displayed by employees on the front desk, especially positive ones, have been observed to transfer to the customer as well, improving their satisfaction with the interaction. Conversely, negative emotions do not seem to have the same transference - likely because negative facial expressions by employees in service profession are generally discouraged, no matter the inner emotional state. Superficial display of positive emotion seems to be sufficient [36].

To what degree the same applies to chatbot agents is still unclear. Research seems to indicate, however, that the answer may be encouraging - display of positive emotion by robots has already been shown to increase perceived trustworthiness [36]. Users also seem to be more receptive to robotic agents delivering their replies in a cheerful voice, as shown in a study with chatbots[36].

Furthermore, the content of the reply is just as, if not more important than the delivery. It has been shown that a not insignificant number of mid-conversation breakdowns in HRI could be avoided if the robot was capable of appropriately judging the emotional context and responding accordingly [40].

This has been noted by the chatbot development community at large - in recent years, the number of projects aimed at incorporating empathy-forcing into the models training loop has been on a rise, with notable examples including bots such as Microsoft Xiaoice [41] among others [42, 43, 44, 45, 46].

2.5.2 Implementing empathy in seq2seq models

This section describes different tactics for incorporating emotional information into the response generation process of a seq2seq model. These methods are applicable to both recurrent and transformer architectures.

Broadly speaking, we can divide the methods of incorporating emotion into the model into two main categories -the use an *emotion aware encoder*, and the use of an *emotion-expressive decoder*, as shown in Fig. 2 – 10.

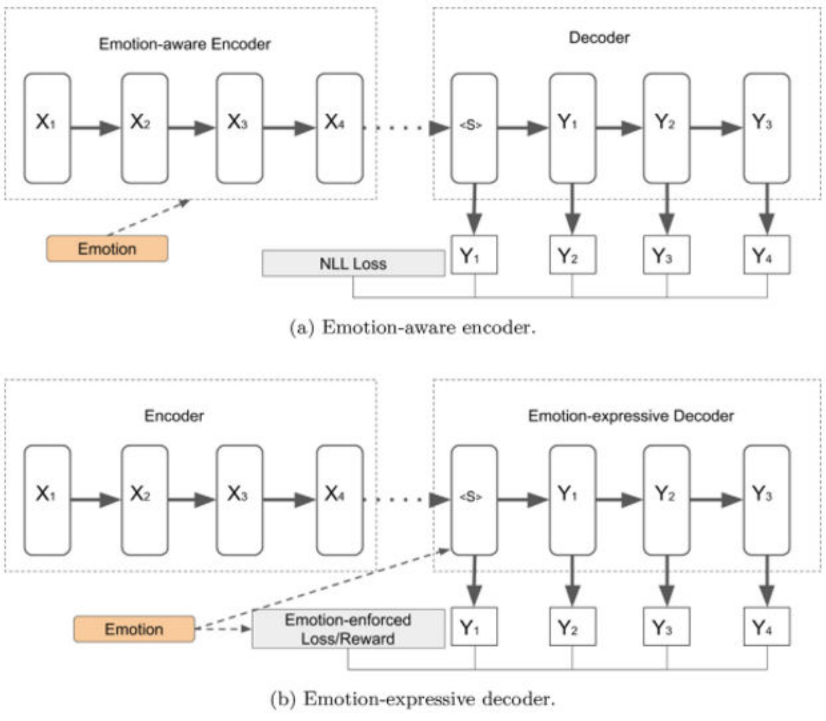


Fig. 2 – 10 Methods of implementing emotional awareness [47]

Emotion-aware encoder is simply an encoder network that has been, in some form, enriched with emotional information. The resulting context vector from this encoder thus contains the necessary clues to produce an empathetic response.

In the case where emotional labels are already present for each user input, this can be accomplished by feeding these labels into the encoder as an additional feature [47]. In a more modular framework, the encoder may also contain a dialogue management module which models dialogue actions based on emotional state as well as other internal states, as a Markov decision process [47].

However, emotional information is not always present for each user input. Even at training time such would require a specialized dataset, and during inference, many users do not use explicit emotional markers such as emojis or tone indicators.

Emotional labels thus need to be obtained by the program itself. For this purpose, many emotional-aware model setups include a separate emotion classifier - a model trained to produce emotional labels for given user input. This classifier can take any form, although it is commonly a recurrent network, and can be trained either on the same data as the main model, or on a separate data set entirely.

The form in which emotional information is provided to the encoder also varies. Prepending the emotion as a string is a well-known, simple approach, but not the only approach possible.

For example, Goel et al. [48] used a separate recurrent classifier to obtain emotional labels for their training data. The predicted emotion was then mapped onto a 256-dim embedding, the same size as the input embeddings used by the encoder. These two embeddings could thus be normalized and summed.

On a more simple note, Li et al. [49], for their chatbot EMMA, simply concatenated the acquired emotional label (from a classifier), emotional cause, and the input itself and passed the result to a GPT model.

Emotion-expressive decoder represents a second approach to seq2seq empathetic generation. In this method, the encoder remains unchanged, and so does the input data. It is decoder - or the information passed to decoder - that is the focus of adjustment, with the aim of forcing more empathetic responses.

Typically, a training method is implemented that favors correct emotional responses over a more neutral, bland response. This can be achieved with the use of specific emotion focused losses in addition to the standard categorical cross-entropy.

For example, the CAiRE model [45] uses pre-trained GPT as a base. The model receives concatenated dialogue history, persona, and a random distractor from a different conversation. The architecture of the GPT model is unchanged, but the training itself uses an emotion classification loss on the output to learn how to distinguish emotions, and a distractor focused loss to learn to distinguish true reply from the distractor.

The emotion loss is calculated on the output of a linear projection layer with 32 output classes, and the distractor loss is likewise calculated on the output of a binary linear classifier.

EmpBot by Zaranis et al. [50] implements an emotion classifier inside the T5 transformer model. This classifier processes encoder outputs and does binary classification. The model is trained, in addition to the standard cross-entropy loss, on binary cross-entropy for the classifier and a cosine similarity loss between the latent emotional representation of input sequence and the generated reply, encouraging emotional mirroring. These latent representations are obtained from the hidden layer of the classifier.

Alternatively, emotional information can be inferred and passed in some form to the decoder itself as a controlled variable.

Wang et al. [51] used this approach in their work, enriching the encoder output of a transformer model with emotion, topic, and situational embeddings as it gets passed to the multi-head cross-attention mechanism.

These embeddings were generated using three independent models and summed with the encoder output. For *situation detection*, a pre-trained Sentence-BERT model was used and trained on the Empathetic dialogues dataset's dialogue contexts. Two more BERT models were used as emotion and topic classifiers.

2.6 Evaluation Metrics

This section describes multiple commonly used evaluation strategies for chatbot models, more specifically BLEU, perplexity, and human evaluation.

2.6.1 BLEU Score

The most common mathematical evaluation metrics in terms of NLP text generative tasks is the **BLEU score** (BiLingual Evaluation Understudy).

The BLEU score evaluates a generated sentence in relation to a reference sentence. The metric was originally developed for translation tasks, with machine-generated translation getting compared to a set of high-quality reference translations.

It can be, and often is, more broadly used for a variety of NLP tasks. This decision is, however, not without its detractors - as a translation metric it is sometimes argued to be somewhat unsuitable to open domain conversations, summarization and other tasks where a very large body of acceptable responses exist.

The score has possible value range of $\langle 0, 1\rangle$ with 0 meaning absolutely no overlap between ground truth output and machine-generated output, and 1 meaning a perfect match.

Mathematically, BLEU score is defined with the following equation [52]:

$$BLEU = \underbrace{min(1, exp(1 - \frac{reference\ length}{output\ length}))}_{\text{b}revity\ penalty} \underbrace{(\prod_{i=1}^{4} precision_i)^{\frac{1}{4}}}_{n-gram\ overlap}$$

Brevity penalty compensates for the lack of recall in the BLEU equation. It penalizes translations that are short compared to the target reference with exponential decay.

The **N-gram overlap** part of the formula counts how many n-grams (uni-grams, bi-grams, tri-grams, four-grams) match their target reference counterpart. It acts as a precision metric, more specifically, modified N-gram precision. The modification consists of setting a firm cap on maximum frequency of each N-gram in relation to calculating precision.

For example, should the target sequence be “the cat sat on the mat”, and the predicted sequence be "the the the the the the", this would receive perfect $\frac{6}{6} = 1$ unigram precision on account of indeed being a token present in the reference sentence. However, modified unigram precision firmly caps the frequency expected for that word at 1, resulting in modified unigram precision of $\frac{1}{6}$.

It should be noted that despite being the closest to commonly accepted evaluation metric text generative tasks have, BLEU score has its limitations as well.

For a start, it is a case-sensitive metric and n-gram matching requires exact matches, where a synonym may fulfill the task equally well. Additionally, all matched words are weighted equally, including prepositions, articles, and conjugations.

Lastly, it is impossible to empirically determine what constitutes a "good" BLEU score, and indeed, realistically achievable BLEU scores will vary between tasks, task variants (for example different languages in machine translation) and even across datasets used. Correspondingly, the presented range of state-of-the-art BLEU scores is somewhat broad.

This can possibly also be attributed to the lack of standardization while setting the metric parameters - the number of changeable parameters is non-negligible, the chosen settings most often unreported, and thus very variable results can be achieved [53]. While an attempt to standardize the metric has been made [53], without information which works do use this framework it is somewhat inadvisable to compare BLEU scores across different articles.

It is therefore good practice to supplement BLEU scores with additional evaluation strategy to get a more complete understanding of model performance.

2.6.2 Perplexity

Perplexity is the second most common evaluation metric for generative chatbots. Broadly speaking, perplexity score can be characterized as a measure of how well the probability distribution predicts a sample. It measures the model's confidence level, which may not necessarily align with its accuracy.

Mathematically speaking, perplexity can be defined as the inverse probability of the test set, normalized by the number of words w

$$PP(W) = P(w_1, w_2, ..., w_n)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, ..., w_n)}}$$

or, after taking into account the chain rule (for unigrams) [54]:

$$PPL = \prod_{i=1}^{N} P(w_i|w_1, ..., w_{i-1})^{-\frac{1}{N}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, ..., w_{i-1})}}$$

Equivalently, it can be defined as the exponential of cross-entropy [54]:

$$PP(W) = 2^{-\frac{1}{N} log(P(w_1, w_2, ..., w_n))}$$

2.6.3 Human Evaluation

Human evaluation is a rather subjective evaluation metric. Nevertheless, as chatbots are programs intended for human-robot interaction, how an audience rates a chatbot offers an important insight into its actual applicability.

Neither BLEU scores nor perplexity were originally intended for free text generation. As a result, a model with high human ratings and low automated metrics may actually perform better than a different model with the inverse.

Various schemes for human evaluation have been proposed, with a relatively common one being grading the model on $\langle 1, 5 \rangle$ Likert scale across various criteria, or blind A/B testing. This is typically achieved using a questionnaire with generated conversations.

In the case of criteria grading, common criteria for empathetic models include relevance, fluency and empathy.

3 Preparatory work

In this chapter we will describe the reasoning behind our implementation choices, including what chatbot characteristics we incorporated in our solution, the technological frameworks picked, and the choice of model.

3.1 Planning the model

This section compares the different chatbot building approaches in terms of their usability for our project. It does so both for the broad choice of chatbot paradigm, and the specific algorithms within our chosen category.

3.1.1 Selecting the paradigm

We were presented with a choice between three basic chatbot paradigms, those being rule-based chatbots, retrieval-based chatbots, and generative chatbots. All three come with their own advantages and disadvantages, which we list within this section.

Rule based chatbots - advantages:

1. Rule-based models show no unpredictable deployment behaviour and never produce unexpected inappropriate output.
2. Creating a rule-based chatbot is quicker and simpler than the alternatives.

Rule-based chatbots - disadvantages:

1. The potential of rule-based models is limited - the technology itself is not significantly evolving, rather just expanding along the predefined guide rails. While still commercially predominant, most if not all advancements are realized within the field of machine learning bots.
2. Rule-based chatbots are not well suited to the open conversation domain. As we wish to build an open-domain conversationalist, rule-based architecture is

not well suited to our goal.

3. Additionally, the creation of a rule-based chatbot requires the creation of the rule base itself. For a model to be capable of more than just the very simplest conversation, often limiting the user's ability to create their own input, such rule base would need to include thousands of rules. We judged the manual creation of such a rule base impractical.

Retrieval-based chatbots - advantages:

1. As retrieval-based chatbots rely on predefined responses, they can usually provide accurate answers to user questions. While an answer can be sometimes irrelevant, it is unlikely to be outright inaccurate.
2. Retrieval-based models often receive good results and provide opportunities for further modifications of architecture.
3. Retrieval-based models generally have quicker response times than generative models.

Retrieval-based chatbots - disadvantages:

1. This architecture requires a constant connection to the dialogue database during not just training, but inference as well, so that the best answer can be chosen. This is unpractical to us in terms of our hardware limitations, as well as the size of publicly available datasets with which we could build such a chatbot.
2. Retrieval-based chatbots can only provide responses based on the predefined responses available in their database, which means that they cannot truly engage in open-ended conversations.
3. Retrieval-based chatbots cannot generate new responses, which limits their ability to provide creative and engaging interactions with users.

Generative chatbots - advantages:

1. Generative chatbots require no additional resource during inference - replies are generated by the model itself based on its learned parameters, making it less hardware-intensive, and easier to train and deploy.
2. Generative chatbots allow for a large degree of experimentation and customization, as they are simply neural networks whose architecture and training process can be modified in a variety of ways.
3. Generative chatbots can engage in open-ended conversations and provide creative responses, which enhances their ability to provide engaging interactions with users.
4. Furthermore, from recent rapid advancement, a large amount of the most prospective, interesting chatbot research is currently happening within the sphere of generative chatbot models.

Generative chatbots - disadvantages:

1. The creativity of generative chatbots can also be consider a disadvantage, as they are likely to provide less accurate responses overall than retrieval-based models. They also have the capability to "hallucinate" - create completely unsupported or even disturbing reply to the user input.
2. Generative chatbots generally have longer response times than their retrieval-based and rule-based counterparts.

Based on our hardware constrains, as well as our desire to experiment with models capable of being general conversationalists, we decided to implement our solution as a generative chatbot.

3.1.2 Deciding on the model type

As already described in overview of the current research, there are, in general, two main categories of model architectural approaches to be considered for a generative seq2seq model. These are a recurrent architecture with added attention mechanism, or the transformer architecture. While other approaches exist - such as using generative adversarial networks or a knowledge map - we did not judge them as effective as the above-mentioned two.

Ultimately, we decided to implement a transformer model, based on these factors:

- Recurrent neural networks take a very long time to train, in particular on large datasets, which conversational datasets typically are.
- In a similar vein, the hardware requirements for such a training process are not ones that were available to us at the time.
- There is no publicly available repository of pre-trained NLP recurrent models in general, not to mention for our type of task more specifically - any model would thus need to be trained for scratch. Transfer learning is a very powerful tool within the field of NLP, and its unavailability in this case would be a disadvantage.
- RNNs are the older technology in this field of research - while still widely utilized and capable of good results, it is broadly agreed that, aside from specific cases, transformers are able to achieve equal or better results with lower computational cost.

3.2 Choosing the basic tools

For programming language, we elected to construct our solution in **Python**, being both a language we are familiar with, and well suited to the task. Python is one of the most widely used and supported languages for artificial intelligence. As a result, robust pre-built libraries for AI development generally and NLP tasks specifically exist.

We decided to work primarily within the environment of **Google Colaboratory** - a free to use, interactive coding IDE for Python. Colaboratory works on the basis of .ipynb files, similarly to Jupyter Notebook. Its greatest advantage is that it is a cloud service, offering free 16GB GPU for user sessions.

Although limited by time and hardware restrictions imposed by Colaboratory on non-subscriber users, it still offers a stronger GPU that we could access for free for the vast majority of our training time. Subsequently, however, the length we could train our models continuously for was limited to the lower range.

We also chose to utilize the **HuggingFace** platform - an open-source resource for NLP tasks. HuggingFace provides a large number of text datasets, tools for working with them, as well as, crucially, a large library of pre-trained transformer models with publicly available code base.

3.2.1 Selecting a machine learning library

HuggingFace offers support for both of Python's most popular machine learning libraries - that is, PyTorch and TensorFlow.

It has been originally developed with PyTorch in mind and thus provides a larger array of tools for this library - in particular, native implementation of data pipelines and training management for a high-level use-cases. However, nearly all models also have a TensorFlow implementation, and have been constructed in such a way that native TensorFlow training commands and data formats also work.

Despite some limitations imposed by this choice (in particular the inability to calculate text based metrics during the training process), we ultimately decided to go with **TensorFlow** for our implementation. Although Pytorch is still regarded as offering a more granular control of the training loop in some circles, we do not personally agree. While that may have been the case at some point, model sub-classing within the TensorFlow framework offers the exact same level of granular control, with the added advantage of being the library we have far more experience with.

3.2.2 Picking a pre-trained model

Despite the recent popularity of GPT models, we opted for the less well-known T5 model over the last GPT version open to the public - GPT-2.

The reasons for this are twofold:

- Firstly, T5-Base is significantly smaller compared to GPT-2. While the first has just over 220 million parameters, the latter has a far larger 1.5 billion. Scaling up language models does, as a rule, lead to improved performance - however, we are limited by our computational resources. Brief experimentation with the models has shown that even just loading GPT-2 to our Colaboratory runtime nearly maxed the available RAM.
- Secondly, GPT-2 is a decoder-only model. We felt that it might be somewhat limiting for us to work with this type of model, as we wanted to try both encoder-aware and decoder-expressive approaches to empathy forcing.

While there are other encoder-decoder models available, such as Facebook's Blender-Bot or a simple Vanilla Transformer, we opted for T5 in order to try the FLAN-T5 variant.

FLAN-T5 has some degree of conversational skill out of the box, which would make a good starting point to compare our empathy training against, and several recent news articles have put it (albeit it larger variants, not the 220mil one) on par with GPT-3. Additionally, it is supposed to be more effective computationally, hopefully leading to smaller training times.

4 Dataset pre-processing

This section describes the selected datasets, the reasoning behind their selection, and all the pre-processing steps we carried out.

4.1 Dataset selection

We found the number of available conversational datasets that could be of use to us to be limited. Ideal dataset would contain open-domain conversations, consist of continuous dialogues, not isolated utterance-reply pairs (so that we could experiment with incorporating dialogue history), and contain additional emotion labels to use in a supervised training setup.

We briefly considered first pre-training our models on one of the largest open-domain conversation datasets - OpenSubtitles [57] with over 300 million conversation sources from movie and tv show subtitles, or the Reddit dataset [58] with over 700 million conversations. After pre-training, smaller emotion focused dataset would be used for fine-tuning.

Ultimately, however, while doing so would likely lead to a better result quality (as several previous experiments showed), we did not have the computational resources required for training this datasets.

Therefore we decided to conduct the entire training on just the smaller, emotion focused dataset originally intended for fine-tuning. For this role, we selected two datasets - **Empathetic Dialogues** and **Daily Dialogues**.

4.1.1 Empathetic dialogues

Empathetic dialogues [55] (ED) is a dataset of 25 thousand emotionally grounded conversations. Each conversation has been manually annotated with the conversational context and an emotional label (32 classes). The conversations can have up to 6 turns.

We obtained the dataset from the HuggingSpace datasets library. The dataset was already partitioned into train, test and validation splits. The dialogues in the dataset were spread across multiple data entries, with each utterance represented as a separate row with associated attributes. These attributes were:

- The associated conversation ID and utterance ID (to properly place it within the larger context),
- Speaker ID - either speaker 1 or speaker 0, switching every other row,
- Context - the conversation's emotion label,
- Prompt - a brief description of the situation talked about, provided by the speakers,
- Utterance - the spoken text itself,
- As well as several others.

Of these, we kept the conversation ID (until utterance-reply pairs were properly linked), the emotional label, and the utterance. Speaker ID always alternated between 1 and 0 after one turn and thus could be inferred. We discarded the prompt column for practicality reasons - during inference time, no such explanation of conversation content would be provided to the machine and thus it would be ill-advised to base our training process on it.

4.1.2 Daily Dialog dataset

Daily Dialog [56] (DD) is a dataset of 13 thousand multi-turn conversations with average length of nearly 8 turns. While the number of conversations is lower than in empathetic dialogues, their overall length actually makes this the bigger dataset. Each conversational entry also has an intention label and an emotion label, with 7 emotions in total.

This dataset was also freely available as part of the HuggingFace dataset library. In this case, however, one data entry did not represent one conversational utterance, but the whole conversation - a list of strings. Separate data points were thus grouped together. The dataset attributes were:

- dialog - an array containing the entire conversation as a list of strings,
- act - an array of classification labels representing utterance intentions (question, directive, commisive, informative) for every utterance in dialog,
- emotion - an array of classification labels representing utterance emotion for every utterance in dialog .

Of these, we kept the dialog column as well as the emotion labels.

4.2 The data processing pipeline

Each data entry was processed using nine primary steps before being linked with its corresponding reply, as can be seen in Fig. 4 − 1. First 8 of those steps pertained to the utterance, ninth step processed the context column.

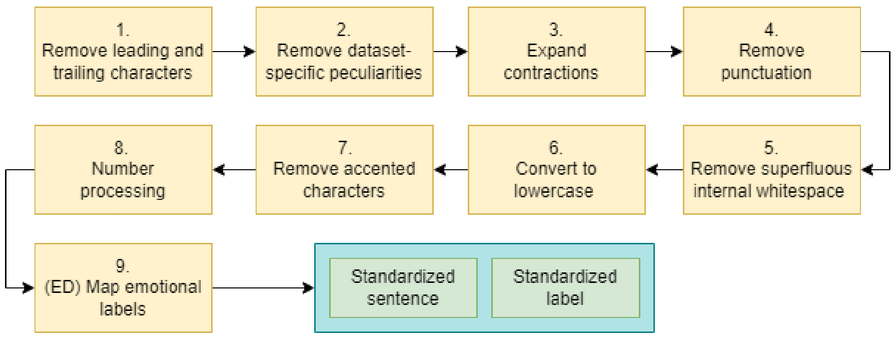

1. Remove leading and trailing characters
2. Remove dataset-specific peculiarities
3. Expand contractions
4. Remove punctuation
5. Remove superfluous internal whitespace
6. Convert to lowercase
7. Remove accented characters
8. Number processing
9. (ED) Map emotional labels
Standardized sentence
Standardized label

Fig. 4 − 1 The pre-processing data pipeline - standardization

These steps were as follows:

1. The removal of leading and trailing characters from each utterance.
2. Processing of dataset-specific peculiarities - in the case of ED, the encoding of commas as "_comma_". We replaced this encoding with standard commas.

In the case of DD, the occasional presence of non-standard apostrophes or random whitespace around apostrophes leading to wrong tokenization. These were replaced by the standard apostrophe.

3. Expanding of contractions - we used a small Python library, Contractions[59], to expand verbs into their full form (I'm $\rightarrow$ I am, hadn't $\rightarrow$ had not...) and thus unify the language across datasets.
4. Removing the punctuation, including the corrected commas.
5. The dataset also sometimes contained unnecessary white-space between words which influenced later pre-processing steps, so we removed it at this point.
6. Converting all text to lowercase.
7. Removing accented characters - this step converted foreign or foreign-originated words into a non-accented form. Accents removed included the acute, grave, circumflex, tilde and others.
8. Number processing - the conversations in the dataset are recorded using several different number conventions - some are written numerically, some verbally. In the case of verbal form, the specific form taken (spaces, no spaces, dot or dash for decimal place..) also differs. We used a second small library [60] to convert all numbers into a numerical form. We build upon the library code to account for differences from conversation to conversation, using a small dictionary of number specific terms.
9. Mapping of emotional labels - as previously mentioned, the ED dataset uses 32 emotional labels across the emotional spectrum. Furthermore, the labels in the dataset were provided as a textual string (e.g. 'guilty'). At this step, we grouped associated emotions into 2 broader categories (positive, negative) and mapped them onto numerical labels. In the case of DD dataset, no mapping was necessary.

These steps were universal across model variations. From this point on, however, the pipelines diverge depending on the planned model architecture. All pipelines executed 3 main steps - linking, tokenization, and the creation of an iterable dataset, as shown in Fig. 4 − 2.

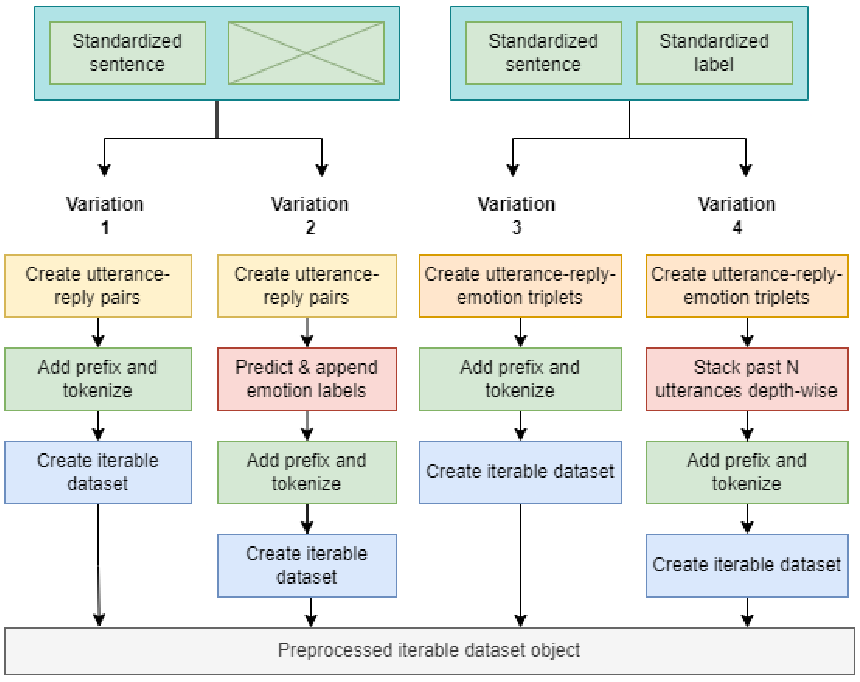


Fig. 4 − 2 The pre-processing data pipeline - variants

These steps were exuceted as follows:

1. Variation for baseline model: linking creates only utterance - reply pairs. Emotion label is not used. Utterance is expanded using a task prefix and utterance reply pairs are tokenized with output length = 128 and iterable dataset dataset is created with 4 main variables - tokenized utterance, utterance attention mask, tokenized reply, reply attention mask. This variation is not particularly memory-intensive. As such, we could use a larger buffer size equal to 1000 and dataset caching to speed up training.
2. Variation for prepend model: linking creates only utterance - reply pairs. Emotion label is not used. Utterance reply pairs are tokenized with output length

= 128 and iterable dataset is created with 4 main variables - tokenized utterance, utterance attention mask, tokenized reply, reply attention mask. The main difference is that, in addition to task prefix, utterances are expanded with predicted emotion labels during pre-processing generated by an independent model.

3. Variation for decoder-expressive model: here, linking creates utterance-reply-emotion triplets. Emotion labels are used by the model and as such form the fifth main variable in the iterable dataset. Additional model size and increase in variables meant we had to lower the buffer size for this variation to 500.
4. Variation for decoder-expressive model with history: this pipeline functions similarly to pipeline three, but tokenizes past utterances as well as the current one. The tokenized utterances are then stacked depth-wise, creating encoded utterance vectors and masks of shape (128, N). Because vector size has to remain constant to create a TensorFlow Dataset object, this pipeline results in a smaller dataset. First N-1 utterances in each conversation are present only as part of the first (128,N) utterance vector, not separate data points like before.

5 Models

This section describes the various implemented architectures, their training procedures and results. Additionally, it contains model evaluation in terms of automated scores as well as human evaluation.

5.1 Fine-tuned model

Our baseline model was obtained from the HuggingFace library. It is a TensorFlow implementation of T5, with language modelling head on top. HuggingFace offers various sizes of the standard T5 model - T5-small, T5-base, T5-large, T5-3B and T5-11B. The same is true for its FLAN variant.

We decided to use the second smallest implementation, T5-base, with 240 million parameters, as it was the largest version we could trian on our setup.

As none of the pre-existent tasks pipelines can be effectively leveraged for our purposes, we defined our own task prefix - “generate a response”.

5.1.1 Model parameters

By default, the pre-trained model consists of an encoder, a decoder, a shared embedding layer, and the language modelling head. There are two possibilities how this architecture can be used, with results either obtained from the LM head, or tying the word embeddings between input and output. FLAN-T5 uses the first option.

Embedding layer vocabulary size was set to 32128. The model has 12 encoder layer blocks, 12 decoder layer blocks, and 12 heads in the multi-head attention mechanism. The size of key, query and value projections is 64. Feed forward layers in both encoder and decoder blocks were set to 2048 neurons, and the encoder layer size was set to 768. Feed-forward layers use the GeLU activation. Lastly, dropout rate is 0.1.

Initial weights were obtained from a pre-trained FLAN-T5 HuggingFace model.

5.1.2 Training setup

Our data was already split into train, validation and test sets by HuggingFace. We used a batch size of 16.

The loss function used was sparse categorical cross-entropy. As optimizer, we experimented with AdaFactor, Adam and AdamW with various set learning rates. The models were evaluated using sparse Top-K categorical accuracy with K=5.

The text specific metrics we would be interested in, such as BLEU, ROGUE and perplexity, cannot be calculated mid-training on the setup we build, one of the unfortunate downsides of not using the native PyTorch architecture of HuggingFace. As a result we only calculated them after training on the model iteration with the lowest loss.

We trained our models for 5 epochs, a common choice for a model of this type and size. During this time it could often be observed that the models would start overfitting. Loss typically started slowly increasing on validation set after 2-4 epochs. One epoch also took almost 95 minutes to run, with additional up to 10 minutes for validation.

For these reasons, to obtain the best iteration of the models in terms of both loss and textual metrics, and to prevent complete data loss and time wastage in case of Google Colaboratory disconnecting us from GPU mid-training, we used a model checkpoint callback. We set it to save our models every epoch with validation loss and accuracy information.

5.1.3 Results

The results presented use the AdamW optimizer with set learning rate of 1e-3, as it seems to lead to slightly better results and converge a little bit quicker than the alternatives.

On empathetic dialogues, over the course of the first epoch, top-5 training accuracy went up from 20 percent to around 95. After the first epoch ends, however, we can witness a slow but steady decrease in validation accuracy and increase in loss - this model starts over-fitting quickly.

From what we have seen, it is not uncommon to fine-tune a model such as this for only one epoch, if the dataset is sufficiently large and the model pre-trained. We will therefore use this first epoch version of the model as our baseline representative. The full training progress can be seen in Fig. 5 − 1.

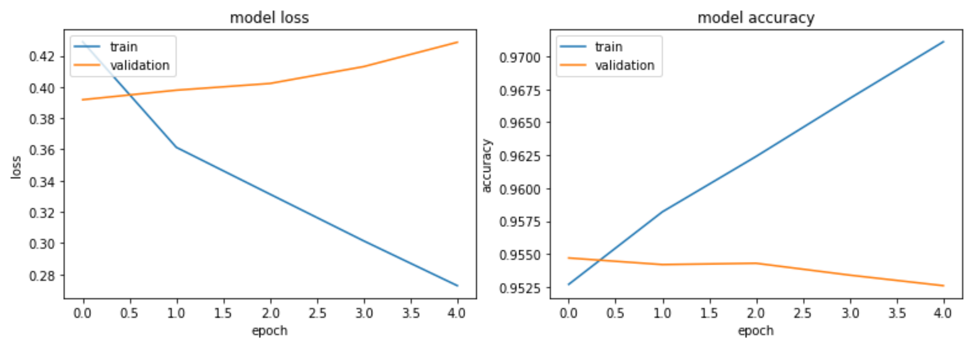


Fig. 5 − 1 Model loss and top=5 accuracy across 5 epochs (ED dataset)

Daily Dialogs behaved a bit differently - over the course of first epoch, top-5 training accuracy has similarly risen from around 30 percent up to 96.30. However, where Empathetic dialogues validation loss starts at just under 0.40 and only worsens, daily dialogs only start to over-fit around 3rd epoch and has lower loss overall. The full training progress can be seen in Fig. 5 − 2

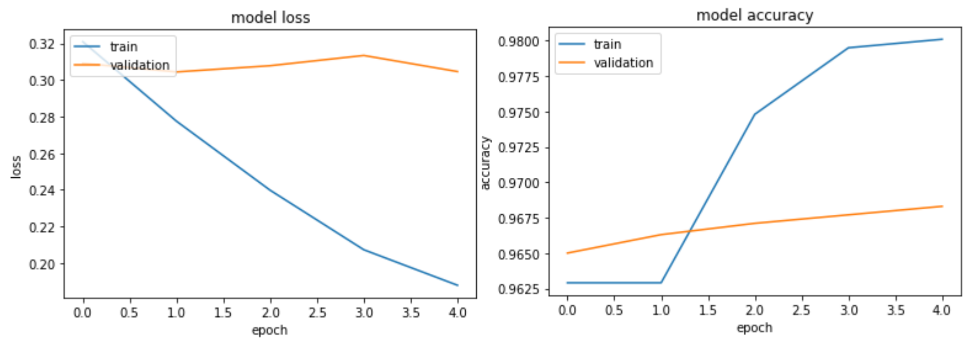


Fig. 5 − 2 Model loss and top=5 accuracy across 5 epochs (DD dataset)

Loss and accuracy, however, do not have a strong informative value in text generation tasks. To actually evaluate the models in a more meaningful way, we used the BLEU score as outlined in theory section of this work. The models achieved the following results on validation set (Tab. 5 – 1) and test set (Tab. 5 – 2).

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
ED Baseline	12.37	5.79	3.52	2.54	6.06
DD Baseline	9.15	4.95	3.41	2.62	5.03

Tab. 5 – 1 BLEU performance on FLAN-T5 baseline models - validation set

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
ED Baseline	11.82	5.43	3.22	2.29	5.69
DD Baseline	9.52	4.96	3.37	2.61	5.12

Tab. 5 – 2 BLEU performance on FLAN-T5 baseline models - test set

Despite lower overall loss and higher top-5 accuracy for Daily Dialogs, in terms of BLEU the ED baseline performs noticeably better. The score disparity seems to arise from better BLEU-1 and slightly better BLEU-2 performance. In terms of BLEU-3 and BLEU-4, Daily Dialogs model actually performs better.

We hypothesize that the larger diversity of conversation styles and topics (including neutral utterances) present in the DD dataset makes it less likely to hit the correct unigrams - the ED model may have an easier time predicting the correct unigrams because they are more frequent in the dataset.

5.2 Encoder aware model

This model uses the baseline FLAN-T5 architecture unchanged - what is modified is the data preparation procedure, with each data point extended by an emotion class label like so: `"<prefix>: <emotion_labels> <input_text> </s>"`

These emotion labels were provided by a separate classifier during pre-processing. We experimented with a prepend-1 and prepend-2 method, expanding the input by the first and first two most likely class labels respectively.

5.2.1 Classifier

To obtain classification labels we opted for a FastText linear classifier [61]. The model architecture can be seen in Fig. 5 − 3.

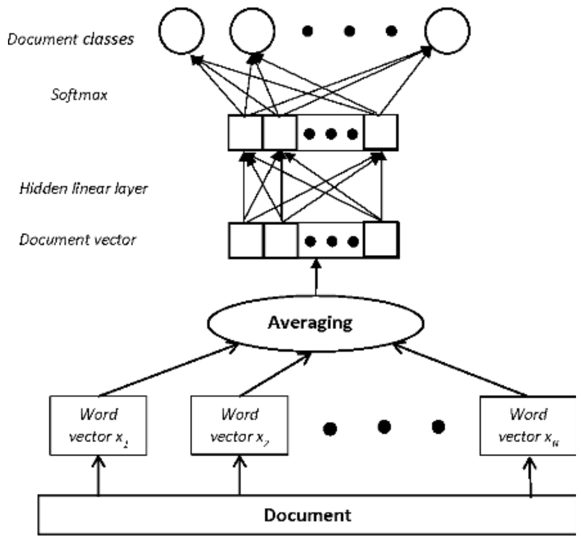


Fig. 5 − 3 Architecture diagram - Fasttext linear classifier [61]

As the empathetic dialogues dataset only provides 1 emotional label for a whole multi-turn conversation, we opted to train the classifier different multi-class emotive text datasets. These were the Emotion dataset available on HuggingFace (6 classes) and the Daily Dialog dataset (7 classes). While the Daily Dialog version achieved slightly lower results, brief testing has proven it to be more broadly applicable beyond the original data. We therefore proceeded with this model version.

Fast-text models accept only a single input, a string in the format

`"__label__<class_label> <input_text>"`, so we adjusted our data accordingly. The pre-processing used was identical to our main data pre-processing procedure.

The model was trained for 15 epochs using a learning rate of 0.1 and a negative sampling loss function. While appearing rather high, similar learning rates (up to 1, even) are commonly used with this classifier.

The evaluation results on test sets can be seen in Tab. 5 – 3. The parameter K denotes number of Top-K predicted labels for each sample.

Dataset	**Test Precision**		**Test Recall**	
	k = 1	k = 2	k = 1	k = 2
Emotion	88.85%	49.13%	88.85%	98.25%
Daily Dialog	**86.30%**	**48.44%**	**86.30%**	**96.89%**

Tab. 5 – 3 Precision and recall of trained fasttext models

5.2.2 Training setup

Similarly to baseline architecture, this variant was also trained using categorical cross-entropy loss and the AdamW optimizer with 1e-3 learning rate. Batch size was kept at 16.

We added one additional callback - a memory cleaning callback to manually call the Python garbage collector at the end of each epoch. Without the memory callback, it was quite common for our training to crash halfway through due to out of memory errors.

5.2.3 Results

On empathetic dialogues, the prepend-2 model shows a similar training progress to the baseline. Model achieves the best validation results after one epoch of training, further training leads to over-fitting (Fig. 5 – 4).

On daily dialog, compared to baseline there was a slight improvement in learning ability - the model seems to slightly benefit from training several epochs, with loss

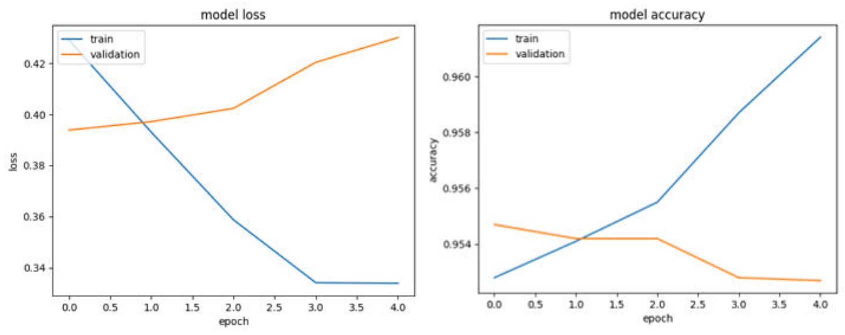


Fig. 5 – 4 Model loss and top-5 accuracy across 5 epochs (ED dataset)

slowly decreasing up to epoch 4. Top-5 accuracy is comparable, with slightly lower loss, as seen in Fig. 5 – 5. Overall, however, the changes are rather small.

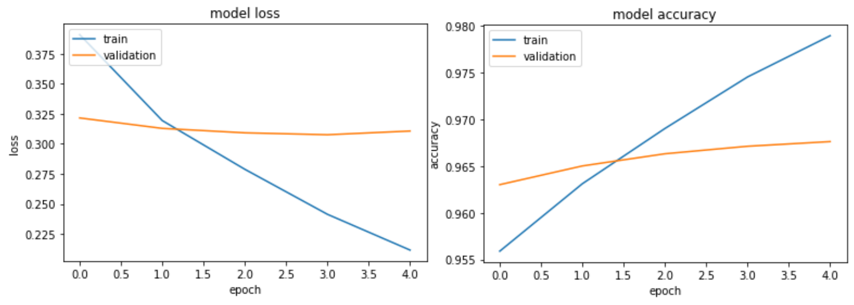


Fig. 5 – 5 Model loss and top-5 accuracy across 5 epochs (DD dataset)

BLEU results on validation and test sets can be seen in Tab. 5 – 4 and Tab. 5 – 5 respectively.

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
ED Prepend-2	12.32	5.87	3.58	2.60	6.09
ED Baseline	12.37	5.79	3.52	2.54	6.06
DD Prepend-2	10.45	5.39	3.69	2.79	5.58
DD Baseline	9.15	4.95	3.41	2.62	5.03

Tab. 5 – 4 BLEU performance on Prepend-2 models - validation set

On empathetic dialogues, we can observe comparable results to the fine-tuned baseline, with slight average BLEU improvement on validation set but slight decrease on

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
ED Prepend-2	11.56	5.39	3.23	2.31	5.62
ED Baseline	11.82	5.43	3.22	2.29	5.69
DD Prepend-2	10.45	5.33	3.55	2.72	5.51
DD Baseline	9.52	4.96	3.37	2.61	5.12

Tab. 5 − 5 BLEU performance on Prepend-2 models - test set

test set. Overall, the actual difference between these two models will likely only be noticable using subjective human evaluation. This seem to be in line with previous research, when architecture modifications typically lower the BLEU compared to fine-tuned models, not increase it, despite improved user experience.

More surprising is the noticeable improvement in BLEU scores for Daily Dialogs. The prepend-2 model delivers a 0.55 jump in terms of average BLEU on validation, and slightly lower but still notable 0.39 jump on test set.

While BLEU-3 and BLEU-4 experience slight increase as well, the main improvement is in terms of BLEU-1 and BLEU-2 scores. Adding emotional direction to the model seems to help it predict uni-grams and bi-grams better, most probably because of increased direction in terms of emotional valence.

5.3 Decoder expressive model

This model was based upon the idea introduced in the EmpBot research article [50]. Standard T5 architecture is expanded by adding a classifier branch which takes encoder hidden state (encoded utterance) and decoder hidden state (encoded generated reply) as input sequentially.

Similarity between the emotional subtext (represented by activations of the first classifier layer) for these two inputs is then maximized via cosine similarity loss. Classifier cross-entropy loss, as well as decoder cross-entropy loss are also computed at each pass as part of the total model loss. This setup encourages emotional mirroring.

The model architecture diagram - depicting a two-layer classifier variant for a binary classification problem - can be seen in Fig. 5 − 6.

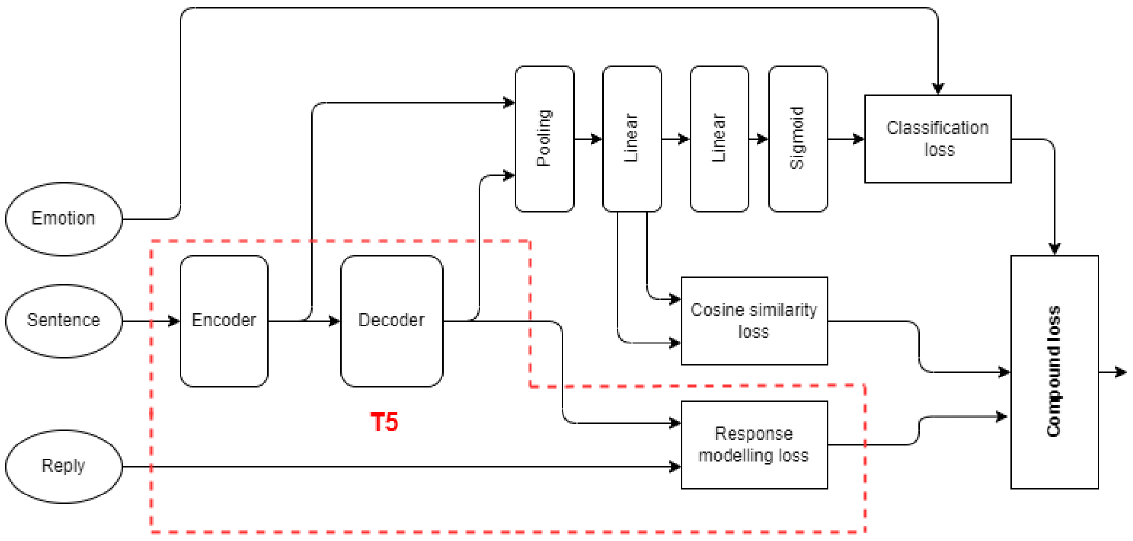


Fig. 5 − 6 Architecture diagram - FLAN-T5 model with a classifier branch

In contrast to the original article, we experimented with several different classifier architectures and activations, and incorporated sentence pooling before the classifier itself (similar to Sentence Transformers such as SentenceBERT). We also experimented with the various parameters the three loss setup provides.

Additionally, by also training this setup on Daily Dialogs without aggregating the number of classes, we experimented with how such a setup performs when extended to multi-class classification.

5.3.1 Classifier architecture experiments

We decided to approach the problem in a modular fashion, first independently deciding the best classifier architecture on a small partial model, then implementing the same architecture as part of the main model to be trained together with T5 encoder and decoder.

Some differences in model accuracy were expected, as simultaneous training of all model parts focuses on minimizing 3 losses at once, and thus won't pay as much attention to classifier accuracy in particular. However, the difference in training time for this partial model and the full model - almost 60 minutes difference per epoch - made this approach imminently practical.

The partial model consisted of a pre-trained T5 encoder (initial weights sourced the same way as for the full model) with a classifier head.

We experimented with multiple hyper-parameters, including:

1. the chosen pooling method (maximum, average, CLS)
2. number of hidden layers (one or two)
3. presence or absence of dropout
4. number of neurons in each hidden layer
5. activation functions.

Additionally, we continued experimenting with optimizer choice, focusing on the AdamW and Adafactor optimizers. We also tested various learning rates.

Models were trained for exploratory purposes - experimentation showed incorporating a separately trained frozen classifier into the final architecture leads to worsened results and in some cases unusable models. As such, to save available disk space (each model requires roughly 1.25GB to store its weights) no model saving callback was used in this phase.

Pooling method

Taking inspiration from sentence transformers such as Sentence-BERT and similar, we implemented a pooling layer between encoder output and classifier. We considered 3 pooling types - max pooling, average pooling and CLS pooling.

Empathetic dialogues in particular have proven rather sensitive to choice of pooling method - out of the above-mentioned, only max pooling avoids the tendency to fall into local minima around 55 percent accuracy range, and even that not always, depending on other architectural factors. Daily dialogs is not as sensitive, but also seems to perform well with max pooling.

Fully connected layers

We experimented with two main architecture versions - a two hidden layers variant with lower neuron counts and a single hidden layer variant with a larger number of neurons.

In both cases, the encoder weights were also trainable through the process - freezing the encoder and training just the classifier has proven a failure, with random or near random prediction results in all cases. It is apparent that correctly extracting emotional context requires the encoder to also cooperate in bringing it forth. The single layer version involved layer sizes of 512,1024, and 2048 neurons. We tried both ReLU and sigmoid activation functions and overall achieved better results with ReLU, with accuracy differences up to 2.5 percent.

The second variant involved two hidden layers with lower neuron counts (sometimes accompanied by dropout). Neuron counts tested were 128-64, 256-128, 512-256 and 1024-512. In lieu of previous results, ReLU activations were used through the network.

We started with the empathetic dialogues dataset. The training process here followed already established trends from previous models - near universally, all models started over-fitting at some point within the span of 5 epochs. We deemed 3 setups the most promising - the 512-256 and 128-64 two layer versions, and the 512 single layer version. Detailed loss and accuracy results of all tested architectures can be seen in Tab. 5 – 6.

The scores displayed are averaged from multiple runs, and are for the best optimizer and learning rate combination for each case.

Model architecture	Neuron counts	Validation loss		Validation accuracy	
		1st epoch	5th epoch	1st epoch	5th epoch
Single Layer	512	0.5211	0.5271	74.51%	75.13%
Single Layer	1024	0.5365	0.5411	73.68%	73.14%
Single Layer	2048	0.5322	0.5846	73.41%	68.05%
Two layers	128-64	0.5145	0.5459	74.44%	74.58%
Two layers	256-128	0.5337	0.5798	73.96%	68.98%
Two layers	**512-256**	**0.5349**	**0.5581**	**72.31%**	**76.08%**
Two Layers	1024-512	0.5604	0.6407	72.99%	57.27%

Tab. 5 – 6 ED - Validation loss and accuracy of various classifier architectures

On daily dialog, classifier architecture does not seem to have significant influence on model performance. All models perform very well, with high accuracy scores and low loss. The only notable detail is that all single layer architectures over-fitted within the first 2 epochs.

As such, we decided to proceed with two-layer architectures only, more specifically the 128-64 variant with best results overall, and the 512 + 256 variant to keep architectures across datasets similar enough to compare. Results of classifier evaluation can be seen in Tab. 5 – 7.

Model variant	Neuron Counts	Validation loss		Validation accuracy	
		1st epoch	5th epoch	1st epoch	5th epoch
Single Layer	512	0.3274	0.4032	90.92%	85.20%
Single Layer	1024	0.3034	0.3169	90.09%	89.02%
Single Layer	2048	0.3199	0.2912	91.34%	90.92%
Two Layers	**128 +64**	**0.3468**	**0.2789**	**91.24%**	**91.55%**
Two Layers	256 + 128	0.4390	0.2829	88.28%	90.80%
Two Layers	512 + 256	0.3028	0.2922	90.22%	91.11%
Two Layers	1024 + 512	0.3053	0.2786	91.47%	91.30%

Tab. 5 – 7 DD - Validation loss and accuracy of various classifier architectures

Dropout

We also experimented with adding dropout to limit over-fitting for our models.

In the case of a two layer classifier, we positioned the dropout layer between the first and second linear layer. In the case of a single-layer model, dropout was positioned between the hidden and output layers.

Despite our expectations, implementing dropout has not lead to betterment of results - in fact, in the two layer use-case, model performance decreased. As such we abandoned this approach and proceeded forward without a dropout layer. The single layer ED variant seems to benefit to some degree from using dropout, as such, we kept the layer for this configuration.

Detailed results for ED models can be seen in Tab. 5 − 8, for the DD models in Tab. 5 − 9. All results have been averaged over multiple runs.

Model variant	Dropout	Validation loss		Validation accuracy	
		1st epoch	5th epoch	1st epoch	5th epoch
512-256	**none**	**0.5349**	**0.5581**	**72.31%**	**76.08%**
512-256	0.1	0.5640	0.5546	73.43%	68.05%
128-64	**none**	**0.5145**	**0.5459**	**74.44%**	**74.58%**
128-64	0.1	0.5562	0.5830	72.14%	72.59%
512	none	0.5211	0.5271	74.51%	75.13%
512	**0.1**	**0.5445**	**0.5362**	**72.28%**	**72.89%**

Tab. 5 − 8 ED - Comparison of performance with dropout present and absent

Model variant	Dropout	Validation loss		Validation accuracy	
		1st epoch	5th epoch	1st epoch	5th epoch
128-64	**none**	**0.3468**	**0.2789**	**91.34%**	**91.55%**
128-64	0.1	0.3362	0.3348	89.99%	89.10%

Tab. 5 − 9 DD - Comparison of performance with dropout present and absent

5.3.2 Training setup

We moved onto the full model with just the most promising classifier architectures - the 512-256 and 128-64 double layer setups and the single layer 512 neuron setup for empathetic dialogues, and the 512-256 and 128-64 double layer setups for daily dialog.

As previously mentioned, there are three loss functions in the full model - the reply generation cross-entropy loss, the binary classification cross-entropy loss, and the emotional context cosine similarity loss.

Both binary and categorical cross entropy yield results in the $\langle 0, 1\rangle$ value interval. However, cosine similarity as implemented in TensorFlow has a $\langle -1, 1\rangle$ value range. To prevent cosine similarity from seemingly lowering the compound loss beyond what it should be, we normalize the cosine similarity at each step using min-max normalization.

The three losses are transformed into one compound loss via linear combination, with the main categorical cross-entropy loss remaining at full value and the supplementary losses added after multiplication with corresponding factors alpha and beta. We experimented with setting these two values to various combinations from the $\langle 0.3, 0.7\rangle$ range. A balanced 0.5 - 0.5 approach seems to lead to the best prediction results.

We also spent some time experimenting with various training setups:

1. Training the encoder and classifier first (as if independent model) using just classification loss, with decoder frozen, then training the decoder only with the 2 remaining losses.
2. Training the encoder and decoder first (as if baseline model), inserting pre-trained classifier and training without emotion classification loss.
3. Taking the baseline encoder, training just decoder and classifier with all losses

on.

4. Training the whole model simultaneously.

We do not record the results of the first three experiments, as they have proven ineffective (scores comparable to full simultaneous training - setup three) or outright harmful to performance (setup one - worsened replies overall, setup two - emotion classification accuracy remains at random chance the whole way through). All recorded results are thus from the last setup - training all model layers simultaneously.

Building upon the knowledge gained from previous setups, we used AdamW as our optimizer and 1e-3 as the learning rate. In addition to Top-5 Accuracy to we also monitored the Binary Accuracy of the classifier via a second added accuracy metric.

5.3.3 Results - Empathetic Dialogues

On empathetic dialogues, for the 512-256 two layer architecture, we can observe a steady emotion classification accuracy increase over the course of all 5 epochs. In terms of prediction accuracy and overall loss, however, the models begins to over-fit around second epoch. This can be seen in Fig. 5 – 7. Top-5 accuracy is comparable

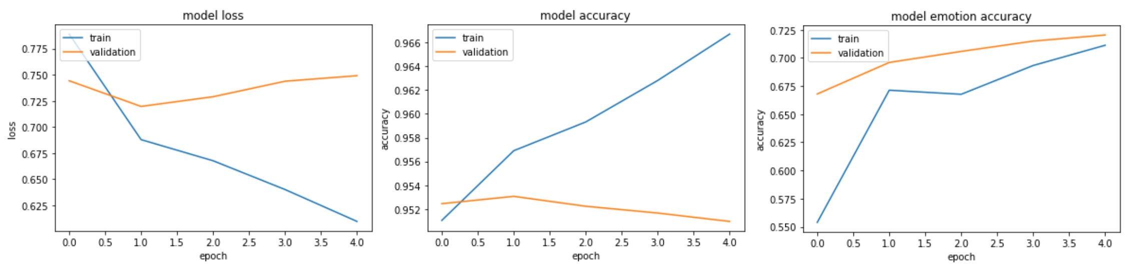


Fig. 5 – 7 ED 512-256 - model loss and accuracy over 5 epochs

to the baseline model, and while loss seems higher, it is important to note this is baseline loss in addition to emotion loss and cosine similarity loss.

BLEU results for this model (for both validation and test sets) can be seen in Tab. 5 – 10.

Set	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Validation	11.13	4.92	2.94	2.11	5.27
ED Baseline	12.37	5.79	3.52	2.54	6.06
Testing	11.02	4.64	2.79	2.04	5.12
ED Baseline	11.82	5.43	3.22	2.29	5.69

Tab. 5 – 10 BLEU performance of ED 512-256 model

For the 128-64 architecture, the emotion accuracy behavior is more erratic - the model experienced a sudden accuracy drop in epoch 4 but seems to have recovered from it in epoch 5. Overall, model over-fits slightly later, at third epoch instead of the second. The overall top-5 accuracy remains comparable to previous architecture. Full training process can be seen in Fig. 5 – 8.

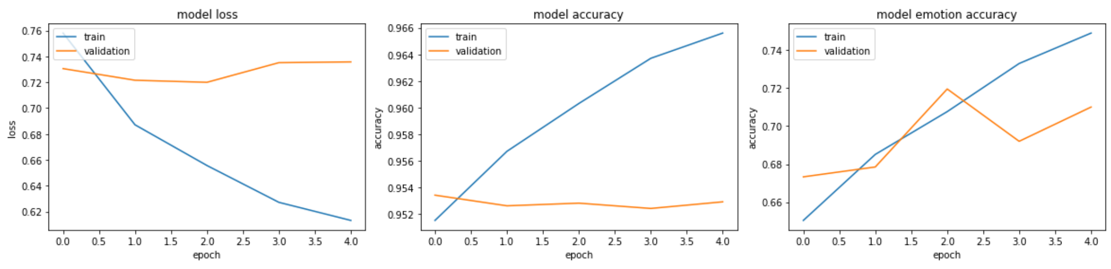


Fig. 5 – 8 ED 128-64 - model loss and accuracy over 5 epochs

The real difference, however, can be seen in BLEU scores. This architecture performs better in terms of BLEU both on validation and test sets, and noticeably so. The AVG BLEU increase compared to the larger classifier is nearly 0.6 on both validation set and test set (Tab. 5 – 11).

The last model for empathetic dialogues was the model with a single layer 512 classifier. This simpler classifier was slower to learn accurately predicting target emotions, and overall achieved lower accuracy. It also starts over-fitting around epoch 2 and slowly worsens, as shown in Fig. 5 – 9, although the decrease is subtler than for the previous setups.

Set	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Validation	12.57	5.40	3.15	2.21	5.83
ED Baseline	12.37	5.79	3.52	2.54	6.06
Testing	12.06	5.18	3.02	2.14	5.60
ED Baseline	11.82	5.43	3.22	2.29	5.69

Tab. 5 – 11 BLEU performance of ED 128-64 model

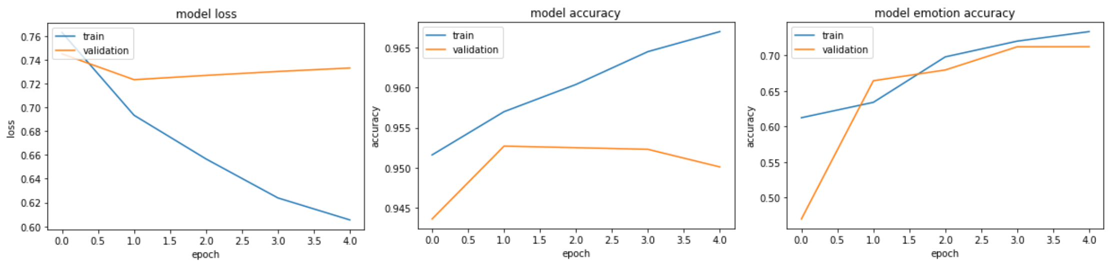


Fig. 5 – 9 ED 512 - model loss and accuracy over 5 epochs

In terms of BLEU scores, this model does not outperform the 128-64 setup, but it is an improvement over the model 512-256 variant. Average BLEU is almost as good as the most successful setup from this category, lagging behind by 0.13 on test set and 0.16 on validation set (Tab. 5 – 12).

Set	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Validation	11.90	5.15	3.04	2.23	5.57
ED Baseline	12.37	5.79	3.52	2.54	6.06
Testing	11.60	5.13	3.01	2.15	5.47
ED Baseline	11.82	5.43	3.22	2.29	5.69

Tab. 5 – 12 BLEU performance of ED 512 model

As can be seen, complicating the model architecture leads to BLEU score decrease across the board. This may mean worse prediction capabilities, but it may just as likely mean higher creativity in the model. The emotion mirroring loss may lead the model to display empathetic behaviours even in part of the original conversation where it wasn't present. As the authors of the article that served as our inspiration [50] also detected a BLEU drop with their architecture compared to their fine-tuned model, we suspend our judgement until human evaluation is performed.

5.3.4 Results - Daily Dialog

For daily dialogs, we considered two model variants - a 128-64 double layer variant and a 512-128 double layer variant.

Emotion classification accuracy was significantly higher for both variants than for any ED variant - up to 20 percent difference, comparing binary classification and a multi-class classification with 7 classes. This, however, has proven to be more of a weakness for these model than a strength - while the mirroring emotions approach worked well enough when only mirroring a positive or negative sentiment generally, with specific emotions the enforced mirroring leads the model to predict output more noticeably different from the golden reply.

The first evaluated architecture, the two layer 128-64 classifier model, achieved slightly lower emotional accuracy, but at negligible levels. Emotion accuracy improved for 3 full epochs before declining (Fig.5 − 10).

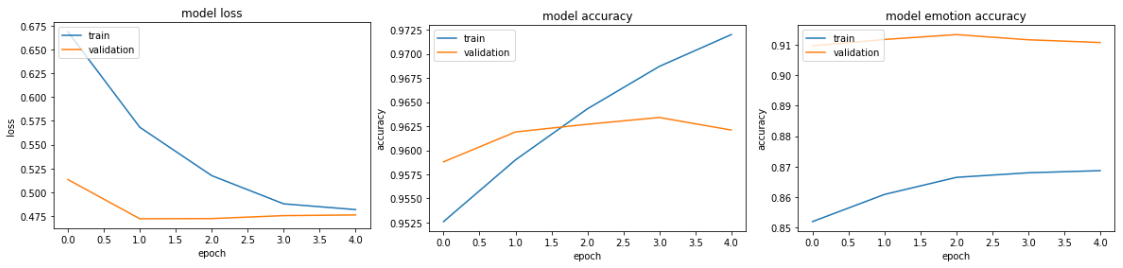


Fig. 5 − 10 DD 128-64 - model loss and accuracy over 5 epochs

The real truth of the situation, however, is revealed in terms of the BLEU score, which is a significant drop from both baseline and the best performing DD model - DD prepend-2. BLEU scores for this model can be seen in Tab.5 − 13.

The 512-128 classifier architecture performed slightly better. Emotion classification

Set	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Validation	6.43	2.89	1.80	1.37	3.12
DD Baseline	9.15	4.95	3.41	2.62	5.03
Testing	6.55	2.96	1.84	1.38	3.18
DD Baseline	9.52	4.96	3.37	2.61	5.12

Tab. 5 − 13 BLEU performance of DD 128-64 model

Set	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Validation	8.24	3.93	2.63	2.03	4.21
DD Baseline	9.15	4.95	3.41	2.62	5.03
Testing	7.14	3.30	2.12	1.63	3.55
DD Baseline	9.52	4.96	3.37	2.61	5.12

Tab. 5 − 14 BLEU performance of DD 512-256 model

climbed to over 90 percent within the first epoch and improved slightly for 2 more epochs. The whole model also keeps improving for a similar length of time before over-fitting, as seen in Fig. 5 − 11.

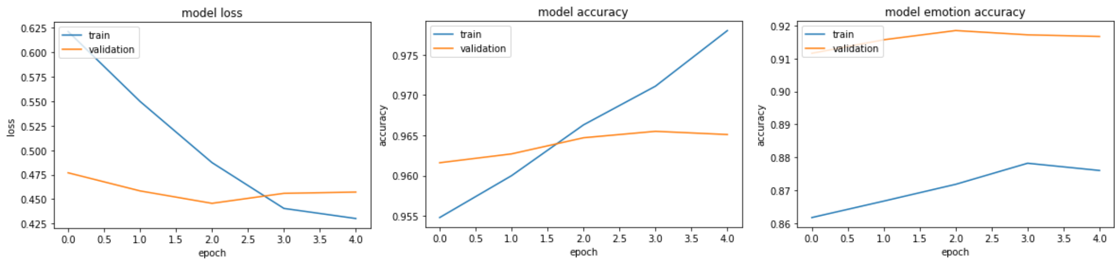


Fig. 5 − 11 DD 512-256 - model loss and accuracy over 5 epochs

In terms of BLEU score, this model performs better, but still noticeably weaker than all other DD and ED setups. The scores can be observed in Tab.5 − 14.

It can be concluded that this particular architecture does not mesh well with the daily dialogs dataset, perhaps because of the multi-class nature of the emotion labeling.

5.4 Implementing history

One downside of the T5 model is that the architecture does not natively support multi-turn generation. That is to be expected - the model was not originally introduced with chatbots in mind. But while single-turn generation does seem to produce acceptable results, deeper understanding of the context could help the model sound more life-like and engaging in its replies.

For that purpose we tried to implement a history-aware version of the best decoder-expressive model for each dataset. Instead of feeding the model only a single previous utterance each turn, we give it the past N utterances, with N being a hyper-parameter we called *past size*.

These past N utterances are then encoded sequentially to avoid the issue of maximum input size for T5 (512) and to prevent out of memory errors during training. After encoding, all past N context vectors are concatenated along the first dimension to form the final context vector of size $(batch\_size, N * input\_length, hidden\_dim)$. This expanded context vector is then passed to the decoder in place of the standard one. The emotion classifier branch continues to receive only the last encoded user utterance. Full schema of the model architecture can be seen in Fig.$5-12$

As the model can take both single turn input - 2D vector of size $(batch\_size, input\_length)$ and multi-turn input - 3D vector of size $(batch\_size, input\_length, hidden\_dim)$, the encoder had to be wrapped in a sub-classed outer layer which takes take of input processing during both training and generation.

To enable weight loading from compatible models without history, the encoder wrapper is not created at initialization, but after, by calling a $remake\_encoder()$ function. This is callable only once - after the model is restructured, an internal switch is flipped and the data flow is rerouted accordingly.

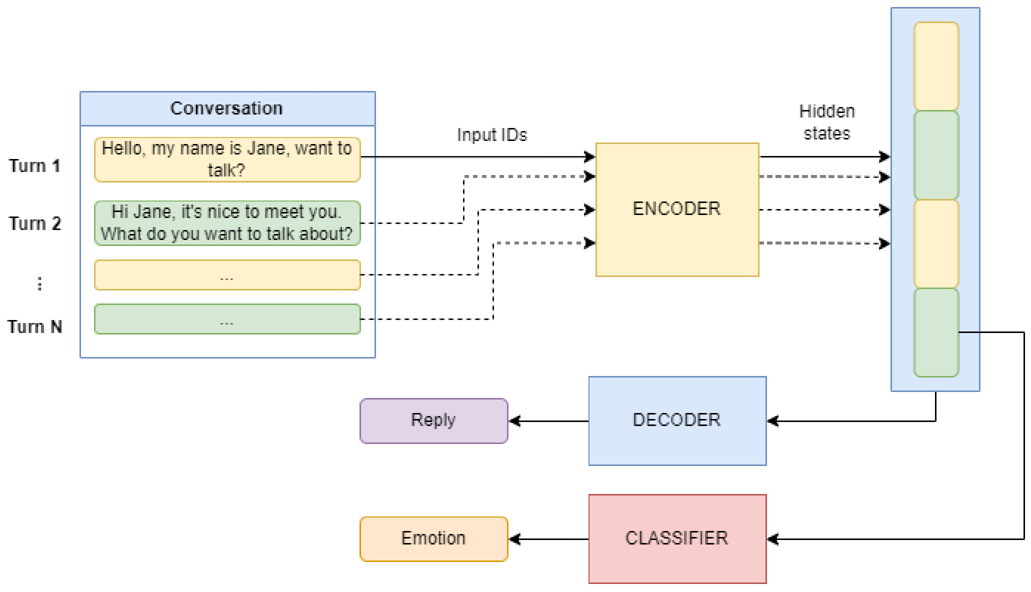

Conversation
Turn 1
Hello, my name is Jane, want to talk?
Turn 2
Hi Jane, it's nice to meet you. What do you want to talk about?
⋮
...
Turn N
...
Input IDs
ENCODER
Hidden states
Reply
DECODER
Emotion
CLASSIFIER

Fig. 5 − 12 Architecture of history-aware model

5.4.1 Training setup

For both daily dialogs and empathetic dialogues, we took the most successful decoder-expressive model and used it as the starting point for the history aware version.

Therefore, most training parameters remain the same as for these models, with only two exceptions:

- To accommodate larger vector sizes (which caused out of memory errors), batch size has been lowered to 8.
- To save time and because these models over-fit extremely quickly, epoch count went to down to 4 (and probably could have been lowered further).

As for past size N, we set it to 4 for daily dialog and 3 for empathetic dialogues datase, as this dataset has shorter conversations on average.

5.4.2 Results

While the experimental setups do learn over the course of the training period, both validation and test BLEU scores indicate implementing history in this way may actually be somewhat detrimental to the model. While the models still answer coherently, the answers do not align as much with the golden reply as before history was implemented.

BLEU results from models with history can be seen in table 5 − 15.

Set	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
ED - Validation	11.89	4.70	2.84	2.09	5.38
ED - Testing	10.66	4.18	2.53	1.87	4.81
DD - Validation	5.04	2.18	1.37	1.13	2.43
DD - Testing	5.75	2.39	1.52	1.24	2.73

Tab. 5 − 15 BLEU scores of history models

Adding dialogue history in this way seems to have negative effect on the DD model in particular, resulting in average BLEU drop of over 2, making this model noticeably inferior by this metric.

In the case of the ED model, BLEU also drops, but less dramatically. Compared to the weakest decoder-expressive model, validation BLEU actually grew slightly, even as testing BLEU dropped by roughly 0.3 points.

This is still an improvement over a different historical architecture we considered - hierarchically processing the encoded inputs with a second decoder, instead of concatenating them. This variant did not learn at all within our experiments, producing scrambled output.

That being said, implementing history does not seem to benefit the decoder-aware model architecture we experimented with within the course of this work.

6 Evaluation

In this section we evaluate individual model performance when compared to each other and to various published models. We do so on the basis of the automated metric BLEU, both BLEU-1 to BLEU-4, and average BLEU.

We also evaluate our own models using subjective evaluation, providing results from a user survey. The respondents were asked to rank our models in terms of reply relevance, reply fluency, and correct emotional valence.

6.1 Automated evaluation results

This sections provides a summary of achieved test BLEU scores across models and datasets. We also compare the achieved scores across different chatbots from various pieces of previous research.

For empathetic dialogues models, the highest average BLEU score was achieved by the fine-tuned baseline, closely followed by the prepend-2 model and the 128-64 classifier model (Tab. 6 – 1). The BLEU differences across these 3 models are all under 0.10 points. The model which performed most poorly was the history-extended classifier, with a score dip under 5 - the only model to fall under that line for this dataset.

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Baseline	**11.82**	**5.43**	**3.22**	**2.29**	**5.69**
Prepend-2	11.56	5.39	3.23	2.31	5.62
512-256 classifier	11.02	4.64	2.79	2.04	5.12
128-64 classifier	12.06	5.18	3.02	2.14	5.60
512 classifier	11.60	5.13	3.01	2.15	5.47
512-256 concat classifier	10.66	4.18	2.53	1.87	4.81

Tab. 6 – 1 ED models evaluation overview - test set

As for daily dialog models, the achieved scores were overall lower than on the empathetic dialogues dataset. This applied to all four calculated BLEU scores. The differences across model architectures were also more notable, with dips as high as a full point for some models.

On test set, the highest achieved average BLEU score was 5.51, achieved by the Prepend-2 model. Following that was baseline with 5.12. After that, BLEU scores took a noticeable dip, with decoder-expressive models scoring 2 points lower. The best decoder-expressive model on this dataset achieved just 3.55 average BLEU (Tab. 6 – 2)

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	AVG BLEU
Baseline	9.52	4.96	3.37	2.61	5.12
Prepend-2	**10.45**	**5.33**	**3.55**	**2.72**	**5.51**
512-256 classifier	7.14	3.30	2.12	1.63	3.55
128-64 classifier	6.55	2.96	1.84	1.38	3.18
512-256 concat classifier	5.75	2.39	1.52	1.24	2.73

Tab. 6 – 2 DD models evaluation overview - test set

We suspect the reason behind this score drop to be up to a combination of factors:

- While the Daily Dialog dataset has emotional labels, a majority of conversations is emotionally neutral or carries low emotional charge.
- The switch to multi-class classification may not be as well-suited to emotional mirroring as binary classification.

6.2 Comparison to previous research

Comparing to previous research in the field of empathetic chatbots, our models achieved respectable results.

We offer several empathetic models for comparison, from among those who trained on the same datasets as us and also reported the AVG BLEU metric. This is a necessary precaution - comparing BLEU scores across models trained on different dataset carries next to no information value.

For empathetic dialogues, comparison can be seen in Tab.6 − 3.

Model	AVG BLEU
MoEL [46]	2.90
MIME [46]	2.98
TopicPrepend-1 [55]	4.17
EmoPrepend-1 [55]	4.36
ED 512-256 Concat (ours)	**4.81**
ED 128-64 (ours)	**5.60**
ED Prepend-2 (ours)	**5.62**
CAiRE [45]	7.03
DD MT+FT [62]	8.1

Tab. 6 − 3 Comparison of AVG BLEU scores on ED test set

For daily dialog, comparison can be seen in Tab. 6 − 4

Model	AVG BLEU
DD 512-256 Concat (ours)	**2.73**
DD 512-256 (ours)	**3.55**
EmpTransfo [63]	3.99
EP-bot [64]	4.23
EmpTransfo + topic [63]	4.51
DD Prepend-2 (ours)	**5.51**

Tab. 6 − 4 Comparison of AVG BLEU scores on DD test set

It should be noted that several models within this comparison have up to 10x the number of parameters, indicating sheer model size to be a reliable tactic for improving chatbot performance.

Additionally, we generated our output for BLEU calculation using greedy search, which traditionally leads to worse results than beam search or sampling techniques. We had to do so as a result of time and computational constraints.

6.3 Human evaluation

6.3.1 Methodology

We decided to proceed forwards with the 5 highest-ranking models in terms of average BLEU scores, as well as our baselines. These models were as follows:

- Empathetic Dialogues baseline
- Daily Dialog baseline
- Empathetic Dialogues prepend-2 model
- Daily Dialog prepend-2 model
- Empathetic dialogs decoder-expressive models with the 512-256 and 128-64 classifiers
- Daily Dialog decoder-expressive model with 128-64 classifier.

Each model was used to generate a response to 30 different, randomly selected prompts from within its test set. The settings used for generation were beam search with 6 beams in 3 separate beam groups, with a maximum length of 60 tokens and no trigram or higher repeat allowed. The responses were generated via the web application interface.

Human evaluation was conducted via a Google Forms survey presented to volunteers from within our educational institution.The volunteers were not informed as to the number of models being evaluated, nor was any information regarding which model generated which output provided to them.

As this adds up to 210 evaluation samples in total, we decided to divide each group of 30 responses into 10 categories of 3 conversations each. Each respondent was then asked to randomly sort themselves into one of 10 test groups based on a d10 dice roll.

Based on the test group selected, they were presented with a different set of 21 conversations, 3 from each evaluated model. This way, all 210 data samples could be used in our research, without having to subject our volunteers to a significantly longer survey.

Every conversation sample was graded based on 3 criteria:

- **Response relevance** - whether the chatbot response seems relevant to the topic discussed in the presented prompt.
- **Response fluency** - whether the chatbot response is grammatically correct and easy to read and understand.
- **Response empathy** - whether the chatbot response correctly detects the emotional sentiment present and addresses it in ad adequate manner.

All criteria were graded on a $\langle 1, 5\rangle$ Likert quality scale (very poor, poor, acceptable, good, very good).

6.3.2 Results

We received results from 61 respondents in total over the course of 7 days. Each test group had at least one randomly sorted respondent. The most numerous of our test groups had 14 respondents, least numerous had 2. The achieved scores for models were averaged across test groups and conversations.

In terms of all 3 criteria ED models performed slightly better than their DD counterparts. This is most visible in achieved empathy scores. For both ED and DD models, at least one model variation improved upon baseline scores across all metrics.

The 512-256 decoder-expressive model was unexpectedly the winner for Empathetic Dialogues. This model had previously achieved a comparatively low score of 5.12 AVG BLEU, which was half a point lower than the 128-64 architecture. However, this model received significantly higher scores for relevance and fluency in human, as well as an increase in empathy score. The full results can be seen in Tab. 6 – 5.

Model	Relevance	Fluency	Empathy
ED Baseline	3.17	3.66	3.48
ED 512-256	**3.84**	**4.01**	**3.88**
ED 128-64	3.36	3.68	3.58
ED Prepend-2	3.69	3.92	3.79

Tab. 6 – 5 Human evaluation results - ED models

For Daily Dialog, the winner was the Prepend-2 model, which improved upon baseline in both relevance and empathy. Full results in this category can be seen in Tab. 6 – 6.

Model	Relevance	Fluency	Empathy
DD Baseline	3.31	3.81	3.30
DD 512-256	3.01	3.66	3.12
DD Prepend-2	**3.57**	**3.88**	**3.65**

Tab. 6 – 6 Human evaluation results - DD models

Overall, while BLEU score slightly decreased with architectural modifications, human results indicate an improved quality of user experience for chatbot models that actively undergone empathy training.

7 The chatbot interface

This section will describe the process and results of creating a GUI chatbot interface to connect our models to.

We decided to implement our chatbot interface as a web-based python application using the **Flask** API. In comparison to its contender Django, Flask has several advantages that made it better suited to our needs:

- It is generally agreed to be more suitable for small scale projects.
- It is very easy to build a quick prototype with it, as it is intuitive and easy to learn.
- It offers easier routing of URL functions.
- It depends on no external libraries.
- It provides flexible, granular control over the workflow.

7.1 Application design

7.1.1 Main communication module

The application consists of a central chat window element upon white background. This window consists of a header, conversation feed, the input field region and an export conversation button. Under the main chat window is a simple generation menu (Fig. $7-1$).The design is responsive, with relative element and text sizing, allowing for maximization in browser as well as easy use on mobile.

Chatbot messages are displayed on the left side in accordance with conventional messaging client design, in white on grey background. User replies are on the right side as dark text on cream. At load, there is already one message 'from the chatbot' present - a prompt to start the conversation.

The input field does not allow for paragraph breaks - however, multi line text is

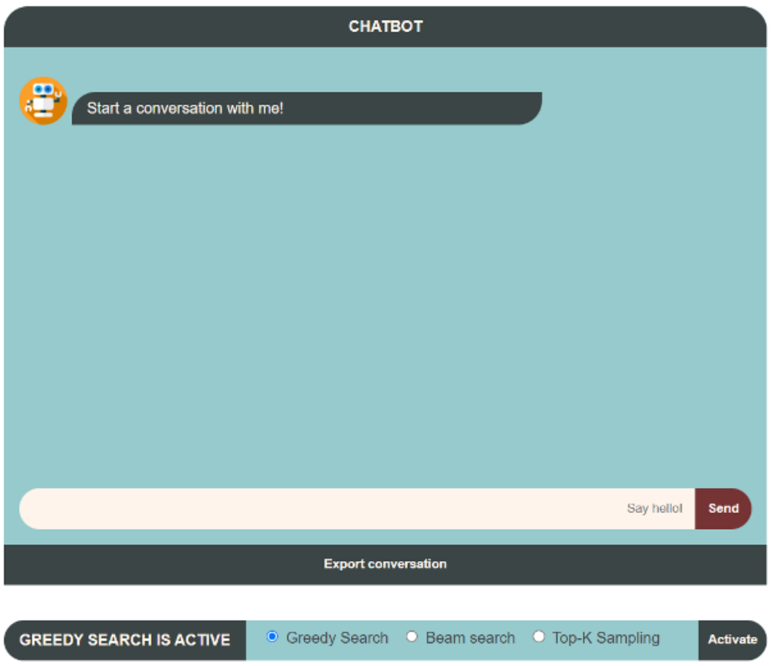


Fig. 7 – 1 Chatbot interface upon startup

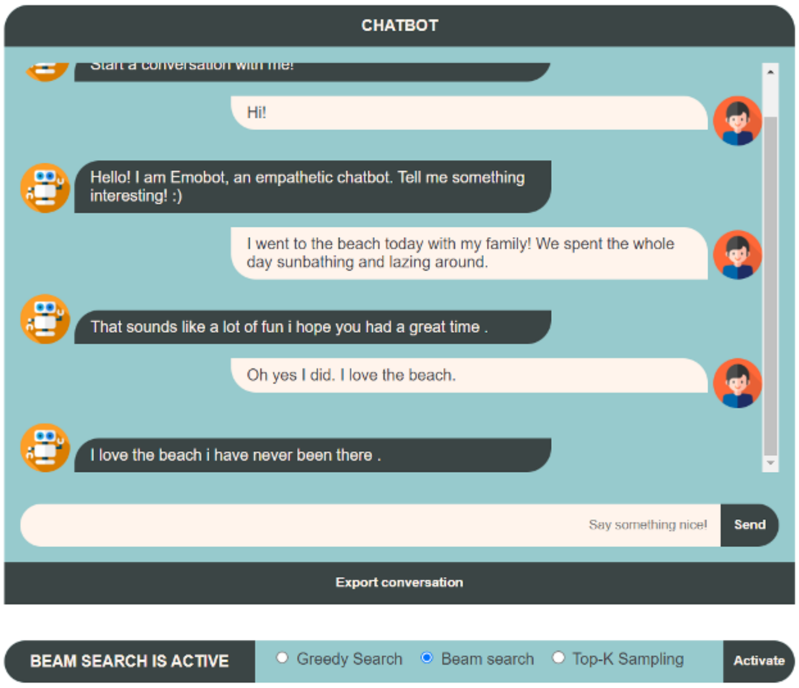
possible. Similarly, emojis and multimedia input are not implemented, as our models have not been trained to process those kinds of data.

The input fields starts enabled by default. To prevent the user from interrupting the chatbot while it generates a response, the input field gets temporarily blocked while the model generates. This is visually distinguished from normal mode both by placeholder text and the button appearance.

Should the conversation length exceed the size of the conversation feed, it automatically becomes scrollable. Mimicking a real messaging client, the scroll-bar position is kept at the bottom automatically unless changed by the user, always displaying the newest messages (Fig. 7 – 2).

7.1.2 Typing indicators

To make the chatbot appear more human-like and personable, a typing indicator is shown while the model generates a reply. This behaviour can be seen in Fig. 7 – 3.


CHATBOT

Start a conversation with me!

Hi!

Hello! I am Emobot, an empathetic chatbot. Tell me something interesting! :)

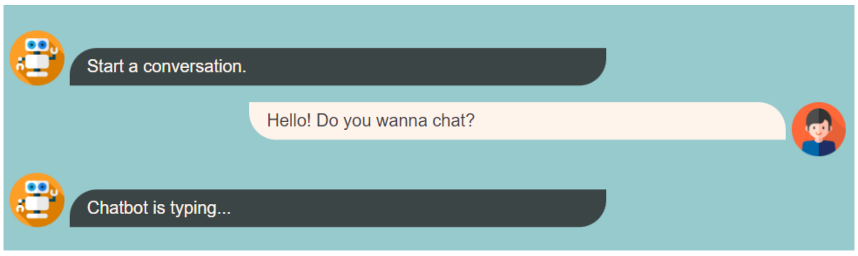
I went to the beach today with my family! We spent the whole day sunbathing and lazing around.

That sounds like a lot of fun i hope you had a great time .

Oh yes I did. I love the beach.

I love the beach i have never been there .

Fig. 7 – 2 Chatbot interface keeps user scrolled at the bottom


Start a conversation.

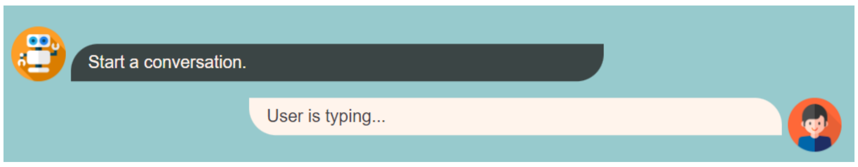
Hello! Do you wanna chat?

Chatbot is typing...

Fig. 7 – 3 Chatbot is typing

Additionally, the user also has a typing indicator, as shown in Fig. 7 – 4. The indicator appears upon first symbol being entered into the input field, and remains until the message is sent. Backtracking and erasing the message completely also temporarily hides the typing indicator.

In case the user is scrolled up in conversation history, typing indicators are not triggered.


Start a conversation.

User is typing...

Fig. 7 – 4 User is typing

7.1.3 New message notifications

The application has several different ways of notifying the user to a new message, including:

- flashing title
- sound effect
- message notification pop-up

Different combinations of these notification features get triggered in different use cases.

1. If the user has the tab active, and is scrolled to the bottom of the message feed, new message simply appears in place of the typing indicator and a sound effect is played.
2. If the user is present in the tab, but scrolled up in dialogue history, no typing indicator is shown, the sound effect is played, and a notification pop-up appears at the top of the main chatbot container. Either scrolling back down manually or clicking at the notification dismisses the panel. Clicking the notification also scrolls the user down to the new message.
3. If the user is not present in the tab at all (if the window is minimized or a different tab has focus), the application employs a flashing title card akin to Facebook Messenger and the soud effect is played. The title settles back to its neutral form once user opens the tab once more.

7.1.4 Exporting the conversation

The application has been additionally equipped with the ability to download conversations. Download is triggered by pressing the ’Export conversation’ button. Doing so opens file dialog allowing user to select the file name and repository.

By default, file name has been set to ’conversation’ and the conversation gets exported as a simple text file, with each utterance starting on a new line and labeled either ’USER:’ or ’BOT:’ depending on who said it.

7.1.5 Generation menu

The application also has a simple generation menu, in which the user can choose between 3 main reply generation styles - greedy search, which generally provides short, simple but relevant responses, beam search, which provides a bit more complicated replies at the cost of a few seconds longer wait, and Top-K sampling, with longer, more involved replies with varying levels of relevance. The application is set to greedy search by default.

Generation style can be changed at any point and as many times as the user wants by selecting the corresponding radio button and clicking Activate. Generation style currently active is displayed on the left.

8 Conclusion

In this thesis, we have developed an emotion-aware chatbot that enhances conversational experiences by recognizing and appropriately responding to users' emotions.

We have explored different approaches to integrating emotional information into model training, including fine-tuning on two different emotional datasets, providing externally generated emotional information, and modifying the architecture of the model itself. We experimented with both single-turn and multi-turn conversations.

We have evaluated our models in terms of classifier accuracy, BLEU-1 to BLEU-4, and AVG BLEU scores, achieving respectable results. We have also evaluated our models via a volunteer survey, asking 61 respondents to grade our models on a scale of 1 to 5 on three main criteria - relevance, fluency and empathy. Here, we have shown that our models improve upon the fine-tuned baseline by a noticeable degree across all three categories.

Lastly, we have created a web application to serve as a graphical interface for the most successful of our chatbot models, as judged by the survey respondents. This application allows the user to comfortably interact with the model, change generation styles, and export conversations.

Overall, our work contributes to the growing field of emotional intelligence in artificial intelligence research and provides a practical application of such technology in human-robot interactions. Future work can expand on the current model to include more advanced emotional recognition and response capabilities, making the chatbot more effective in enhancing conversational experiences. In particular, we believe both model scaling and a successful move towards multi-turn conversation would lead to improved dialogue quality.

Bibliography

[1] Goodrich, M.A. and Schultz, A.C., 2008. *Human–robot interaction: a survey.* Foundations and Trends® in Human–Computer Interaction, 1(3), pp.203-275.

[2] Ramesh, K., Ravishankaran, S., Joshi, A. and Chandrasekaran, K., 2017. *A survey of design techniques for conversational agents.* In International conference on information, communication and computing technology (pp. 336-350). Springer, Singapore.

[3] Damiano, L. and Dumouchel, P., 2018. *Anthropomorphism in human–robot co-evolution.* Frontiers in psychology, 9, p.468.

[4] Costa, P., 2018. *Conversing with personal digital assistants: on gender and artificial intelligence.* Journal of Science and Technology of the Arts, 10(3), 59-72.

[5] Bates, M., 2019. *Health care chatbots are here to help.* IEEE pulse, 10(3), pp.12-14.

[6] Davenport, T. and Kalakota, R., 2019. *The potential for artificial intelligence in healthcare.* Future healthcare journal, 6(2), p.94.

[7] Ahmed, A., Ali, N., Aziz, S., Abd-Alrazaq, A.A., Hassan, A., Khalifa, M., Elhusein, B., Ahmed, M., Ahmed, M.A.S. and Househ, M., 2021. *A review of mobile chatbot apps for anxiety and depression and their self-care features.* Computer Methods and Programs in Biomedicine Update, 1, p.100012.

[8] Amiri, P. and Karahanna, E., 2022. *Chatbot use cases in the Covid-19 public health response.* Journal of the American Medical Informatics Association, 29(5), pp.1000-1010.

[9] Adamopoulou, E. and Moussiades, L., 2020. *Chatbots: History, technology, and applications. Machine Learning with Applications*, 2, p.100006.

[10] Adamopoulou, E. and Moussiades, L., 2020. *An overview of chatbot technology.* In IFIP International Conference on Artificial Intelligence Applications and Innovations (pp. 373-383). Springer, Cham.

[11] Anandarajan, M., Hill, C., Nolan, T., Anandarajan, M., Hill, C. and Nolan, T., 2019. *Text preprocessing. Practical text analytics: Maximizing the value of text data*, pp.45-59.

[12] Sutskever, I., Vinyals, O. and Le, Q.V., 2014. *Sequence to sequence learning with neural networks.* Advances in neural information processing systems, 27.

[13] Williams, R. J., and Zipser, D. (1989). *A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.* Neural Computation, 1(2), 270–280. doi:10.1162/neco.1989.1.2.270

[14] Murphy, K.P. and Massachusetts Institute Of Technology, 2012. *Machine learning : a probabilistic perspective.* Cambridge (Ma): Mit Press.

[15] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. *Learning phrase representations using RNN encoder-decoder for statistical machine translation.* arXiv preprint arXiv:1406.1078

[16] Zarrieß, S., Voigt, H. and Schüz, S., 2021. *Decoding methods* neural language generation: a survey. Information, 12(9), p.355.

[17] Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning.* MIT press.

[18] Cho, K., Van Merriënboer, B., Bahdanau, D. and Bengio, Y., 2014. *On the properties of neural machine translation: Encoder-decoder approaches.* arXiv preprint arXiv:1409.1259.

[19] Masini, R.P., Medeiros, M.C. and Mendes, E.F., 2021. *Machine learning advances for time series forecasting.* Journal of Economic Surveys.

[20] Hochreiter, S. and Schmidhuber, J., 1997. *Long short-term memory. Neural computation*, 9(8), pp.1735-1780.

[21] Bahdanau, D., Cho, K. and Bengio, Y., 2014. *Neural machine translation by jointly learning to align and translate.* arXiv preprint arXiv:1409.0473.

[22] Luong, M.T., Pham, H. and Manning, C.D., 2015. *Effective approaches to attention-based neural machine translation.* arXiv preprint arXiv:1508.04025.

[23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. *Attention is all you need.* Advances in neural information processing systems, 30.

[24] Lin, T., Wang, Y., Liu, X. and Qiu, X., 2022. *A survey of transformers.* AI Open.

[25] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. and Liu, P.J., 2020. *Exploring the limits of transfer learning with a unified text-to-text transformer.* J. Mach. Learn. Res., 21(140), pp.1-67.

[26] Ganesh, P., Chen, Y., Lou, X., Khan, M.A., Yang, Y., Sajjad, H., Nakov, P., Chen, D. and Winslett, M., 2021. *Compressing large-scale transformer-based models: A case study on bert.* Transactions of the Association for Computational Linguistics, 9, pp.1061-1080.

[27] OpenAI (2022). *ChatGPT: Optimizing Language Models for Dialogue.* [online] OpenAI. Available at: https://openai.com/blog/chatgpt.

[28] Openai, 2023. *GPT-4 Technical Report.* [online] Available at: https://cdn.openai.com/papers/gpt-4.pdf.

[29] Qiu Xipeng, Sun TianXiang, Xu Yige, Shao Yunfan, Dai Ning, Huang Xuanjing. *Pre-trained models for natural language processing: A survey.* Sci. China Technol. Sci., 63 (10) (2020), pp. 1872-1897, 10.1007/s11431-020-1647-3

[30] Casola, S., Lauriola, I. and Lavelli, A., 2022. *Pre-trained transformers: An empirical comparison.* Machine Learning with Applications, 9, p.100334.

[31] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V. and Zettlemoyer, L., 2019. *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.* arXiv preprint arXiv:1910.13461.

[32] Shaw, P., Uszkoreit, J. and Vaswani, A., 2018. *Self-attention with relative position representations.* arXiv preprint arXiv:1803.02155.

[33] Huang, C.Z.A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A.M., Hoffman, M.D., Dinculescu, M. and Eck, D., 2018. *Music transformer.* arXiv preprint arXiv:1809.04281.

[34] Chung, H.W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S. and Webson, A., 2022. *Scaling instruction-finetuned language models.* arXiv preprint arXiv:2210.11416

[35] Picard, R.W., 2000. *Affective computing.* MIT press.

[36] Chuah, S. and Yu, J., 2021. *The future of service: The power of emotion in human-robot interaction.* Journal of Retailing and Consumer Services, 61, p.102551.

[37] Shank, D., Graves, C., Gott, A., Gamez, P. and Rodriguez, S., 2019. *Feeling our way to machine minds: People's emotions when perceiving mind in artificial intelligence.* Computers in Human Behavior, 98, pp.256-266.

[38] Klein, J., Moon, Y., and Picard, R., 2002. *This computer responds to user frustration: Theory, design, and results.*Interacting with Computers, vol. 14, pp. 119–140.

[39] Bickmore, T. and Picard, R., 2005. *Establishing and maintaining long-term human-computer relationships.* ACM Transactions on Computer-Human Interaction, 12(2), pp.293-327.

[40] Martinovski, B., and Traum, D. 2003. *Breakdown in human- machine interaction: the error is the clue.* In Proceedings of the ISCA tutorial and research workshop, 11–16.

[41] Zhou, L., Gao, J., Li, D. and Shum, H.Y., 2020. *The design and implementation of xiaoice, an empathetic social chatbot.* Computational Linguistics, 46(1), pp.53-93.

[42] Dongkeon Lee, Kyo-Joong Oh and Ho-Jin Choi, *The chatbot feels you - a counseling service using emotional response generation.* 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), Jeju, Korea (South), 2017, pp. 437-440, doi: 10.1109/BIGCOMP.2017.7881752.

[43] Nidhin Harilal, Rushil Shah, Saumitra Sharma, and Vedanta Bhutani. 2020. *CARO: An Empathetic Health Conversational Chatbot for People with Major Depression.* In Proceedings of the 7th ACM IKDD CoDS and 25th COMAD (CoDS COMAD 2020). Association for Computing Machinery, New York, NY, USA, 349–350. https://doi.org/10.1145/3371158.3371220

[44] K. -J. Oh, D. Lee, B. Ko and H. -J. Choi, *A Chatbot for Psychiatric Counseling in Mental Healthcare Service Based on Emotional Dialogue Analysis and Sentence Generation.* 2017 18th IEEE International Conference on Mobile Data Management (MDM), Daejeon, Korea (South), 2017, pp. 371-375, doi: 10.1109/MDM.2017.64.

[45] Lin, Z., Xu, P., Winata, G.I., Siddique, F.B., Liu, Z., Shin, J. and Fung, P., 2020, April. *Caire: An end-to-end empathetic chatbot.* In Proceedings of the AAAI conference on artificial intelligence (Vol. 34, No. 09, pp. 13622-13623).

[46] Majumder, N., Hong, P., Peng, S., Lu, J., Ghosal, D., Gelbukh, A., Mihalcea, R. and Poria, S., 2020. *MIME: MIMicking emotions for empathetic response generation.* arXiv preprint arXiv:2010.01454.

[47] Ma, Y., Nguyen, K.L., Xing, F.Z. and Cambria, E., 2020. *A survey on empathetic dialogue systems.* Information Fusion, 64, pp.50-70.

[48] Goel, R., Susan, S., Vashisht, S. and Dhanda, A., 2021, September. *Emotion-aware transformer encoder for empathetic dialogue generation.* In 2021 9th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW) (pp. 1-6). IEEE.

[49] A. Ghandeharioun, D. McDuff, M. Czerwinski and K. Rowan, *EMMA: An Emotion-Aware Wellbeing Chatbot.* 2019 8th International Conference on Affective Computing and Intelligent Interaction (ACII), Cambridge, UK, 2019, pp. 1-7, doi: 10.1109/ACII.2019.8925455.

[50] Zaranis, E., Paraskevopoulos, G., Katsamanis, A. and Potamianos, A., 2021. *EmpBot: A T5-based Empathetic Chatbot focusing on Sentiments.* arXiv preprint arXiv:2111.00310.

[51] Wang, Y.H., Hsu, J.H., Wu, C.H. and Yang, T.H., 2021, January. *Transformer-based empathetic response generation using dialogue situation and advanced-level definition of empathy.* In 2021 12th International Symposium on Chinese Spoken Language Processing (ISCSLP) (pp. 1-5). IEEE.

[52] Papineni, K., Roukos, S., Ward, T., and Zhu, W.J. (2002). *BLEU: A Method for Automatic Evaluation of Machine Translation.* In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL) (pp. 311-318)

[53] Post, M., 2018. *A call for clarity in reporting BLEU scores.* arXiv preprint arXiv:1804.08771.

[54] Goodman, J. (2001). *A bit of progress in language modeling.* Computer Speech & Language, 15(4), 403-434.

[55] Rashkin, H., Smith, E.M., Li, M. and Boureau, Y.L., 2018. *Towards empathetic open-domain conversation models: A new benchmark and dataset.* arXiv preprint arXiv:1811.00207.

[56] Li, Y., Su, H., Shen, X., Li, W., Cao, Z. and Niu, S., 2017. *Dailydialog: A manually labelled multi-turn dialogue dataset.* arXiv preprint arXiv:1710.03957.

[57] opus.nlpl.eu. (n.d.). *OpenSubtitles.* [online] Available at: https://opus.nlpl.eu/OpenSubtitles-v2018.php [Accessed 5 Apr. 2023].

[58] Dziri, N., Kamalloo, E., Mathewson, K.W. and Zaiane, O., 2018. *Augmenting neural response generation with context-aware topical attention.* arXiv preprint arXiv:1811.01063.

[59] Kooten, P. van (n.d.). *contractions.* [online] PyPI. Available at: https://pypi.org/project/contractions/ [Accessed 5 Apr. 2023].

[60] Rawat, A. (n.d.). *wordtodigits.* [online] PyPI. Available at: https://pypi.org/project/wordtodigits/ [Accessed 5 Apr. 2023].

[61] Zolotov, V. and Kung, D., 2017. *Analysis and optimization of fasttext linear text classifier.* arXiv preprint arXiv:1702.05531.

[62] Shuster, K., Ju, D., Roller, S., Dinan, E., Boureau, Y.L. and Weston, J., 2019. *The dialogue dodecathlon: Open-domain knowledge and image grounded conversational agents.* arXiv preprint arXiv:1911.03768.

[63] Zandie, R. and Mahoor, M.H., 2020. *Emptransfo: A multi-head transformer architecture for creating empathetic dialog systems.* arXiv preprint arXiv:2003.02958.

[64] Yoo, S. and Jeong, O., 2021. *EP-Bot: empathetic chatbot using auto-growing knowledge graph.*

Appendices