Building Chinese Chat Bot with Controlled Sentence Function

Lue Shen*
Department of Mathematics
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
lshenac@connect.ust.hk

Abstract

In this report, three sentence functions, say interrogative, imperative and declarative, are implemented to control the generation of response to a Chinese sentence. The proposed conversation model is able to produce informative responses with controlled sentence functions, which is transformed from a latest research [1]. A data set with 2 million labeled Weibo post-response pairs is loaded for training, and one WeChat API is utilized to construct a Chinese chat bot and conduct experiments on the trained model in public. Results indicate that the conversation model fulfill all the expected requirements: it is capable of generating responses with both controlled sentence function and informative content.

1 Introduction

Sentence function, an elementary linguistic terminology, refers to the purpose of a speaker to speak out a sentence, phrase or clause. It is an answer to the question "why has this been said?". There are four fundamental sentence functions in English, including interrogative, imperative, declarative and exclamative [2]. Similar linguistic phenomenon also appears in Chinese and other languages. Since each sentence function is different from others in language rules, the transformation between any two types of them requires word order changes, syntactic pattern modifications, etc. [3]

As shown in Table 1, there can be multiple responses to the same post and each of them has a different sentence function, say purpose of the speaker. Specifically speaking, An interrogative response is raising a question to retrieve further information from the listener, and it usually ends with a question mark. An imperative response, however, is to make requests on the listener, usually ending with a period or exclamation mark. Exclamatory responses present a strong emotion from the speaker, ending with exclamation mark while declarative sentences simply state an idea with period ending them. Therefore, sentence function, say the purpose of a speaker, can be a significant factor during interactions in conversational systems. On top of that, interrogative and imperative responses are able to avoid stalemates and they are proactive behaviors in dialogues to lead the conversation to go further [4] [5].

Compared to generic generation and manipulation of text [6], sentence function is a global control variable instead of local controlling and this will be more challenging since the entire text global structure needs to be adjusted, i.e. changing word orders and patterns. Another challenge is the compatibility of sentence functions and contents. There are many universal and meaningless responses like "对吗？| Right?", "哦。| I see.", "请便！| Go ahead!" or "我也是。| So do I.". Hence, the conversation model must manage to generate responses with both controllable sentence function and informative content.

*Currently studying MSc in Financial Mathematics, submission date: May 26th, 2019

Preprint. Work in progress.

Table 1: Responses to the same post with 4 different sentence functions

Post		我为什么这么聪明? Why am I so smart?
Responses	**Interrogative**	你遗传了谁的基因? Whose genes do you inherit?
	Imperative	下次小组项目让我抱大腿吧! Please help my project in the future!
	Exclamative	你真是聪明啊! What a smart guy!
	Declarative	那是因为你有一部智慧手机。 That's because you have a smart phone.

In this report, a conversation generation model is proposed to take care of the challenges aforementioned. An encoder-decoder structure with a hidden variable in conditional variational auto encoder (CVAE) is devised to project sentence functions into different subspaces in the hidden space and capture corresponding word patterns within each sentence function. More model details will be covered in the next section. A labeled data set of Weibo post-response pairs are retrieved from Prof. Minlie Huang's homepage, and it takes more than 169 hours (7 days) to complete the training on my personal computer (Intel(R) Core (TM) i9-9900K CPU, NVIDIA GeForce RTX 2080) since the data size is close to 2 million. Experiments using WeChat Python API itchat indicates the effectiveness of the proposed model. Future studies are discussed at the end of this report.

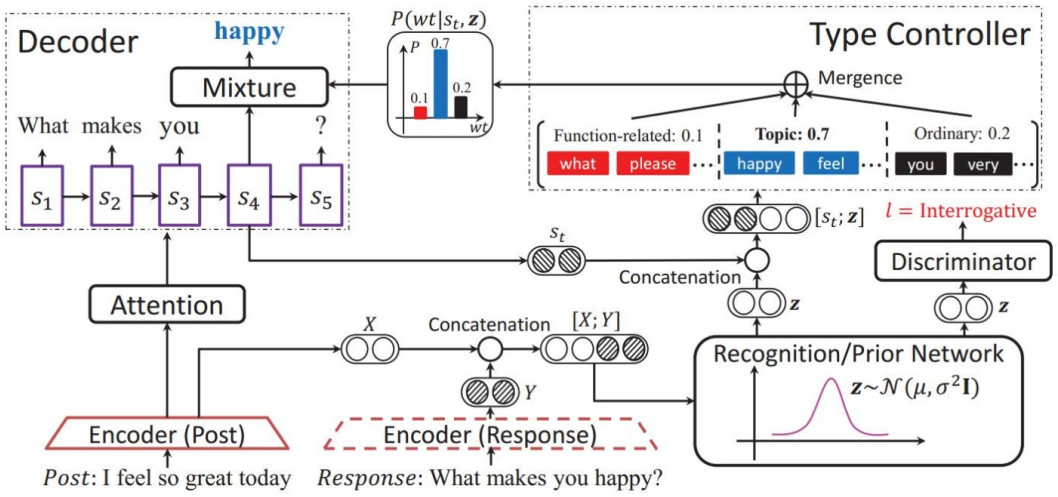

Figure 1: Model Framework

2 Theories

The conversation model is supposed to solve the following problem: given a post sentence $X = x_1 x_2 ... x_n$ and a sentence function l, generate an informative response sentence $Y = y_1 y_2 .. y_m$ in sentence function l, where x_i and y_i are Chinese characters or words at the i-th position within their sentence. Since a controlled hidden variable $\boldsymbol{z}$ is introduced to ensure the consistency of responses with sentence functions, the problem can be defined as an optimization problem:

$$\arg\max_{\theta} P(Y, \boldsymbol{z}|X, l) = \arg\max_{\theta} P(\boldsymbol{z}|X, l) \cdot P(Y|X, l, \boldsymbol{z})$$

Then, the problem can be further divided into two sub tasks: one is to estimate $P(\boldsymbol{z}|X,l)$, while the other is to estimate $P(Y|X,l,\boldsymbol{z})$. The first one can be handled by the recognition and prior network in the model, whereas the latter one can be calculated as $P(Y|X,l,\boldsymbol{z}) = \prod_{i=1}^{m} P(y_i|Y_{<i},X,l,\boldsymbol{z})$ by the decoder. An overview of the model is presented in Figure 1. Specifically speaking,

Data Preprocessing The model will split Chinese sentences, both post and response into characters and words at the beginning. These characters and words are so-called hot vectors in natural language programming. Then each of those characters or words will be converted to a 64-dim numeric vector according to a Baidu Baike word embedding scheme, which is 1.51 GB and covers 1 million Chinese words, three times larger than an ordinary modern Chinese dictionary. For those characters or words that cannot be found in the embedding scheme, a pure 0 64-dimension vector will be used to represent them. Besides, since there is a word limit for Weibo posts and responses and we need to concatenate post and response pairs as an input to recognition and prior network, each sentence is converted to a matrix with a shape of 64×128 by filling empty positions with 0 vectors.

Encoder-Decoder Framework The encoder-decoder framework is essentially to bridge the initial vector representations and hidden vector representations using a gated recurrent unit (GRU). Given a sequence $X = x_1x_2...x_n$, its embedding vector, say $\boldsymbol{E}(X) = e_1e_2...e_n$, can be calculated straightforwardly based on the embedding scheme aforementioned. Then the numeric vectors is plugged into the encoder, the output is generated by Equation 1.

$$\begin{aligned} \boldsymbol{H} &= \boldsymbol{h}_1\boldsymbol{h}_2...\boldsymbol{h}_n \\ \boldsymbol{h}_i &= \boldsymbol{GRU}(e_i, \boldsymbol{h}_{i-1}) \end{aligned} \tag{1}$$

As for the decoder, another GRU is implemented to generate the response sequence $Y = y_1y_2...y_m$ from a hidden representation variable $\boldsymbol{S} = \boldsymbol{s}_1\boldsymbol{s}_2...\boldsymbol{s}_m$. Suppose the intended embedding vectors for Y are $\boldsymbol{E}(Y) = e_1e_2...e_n$, then the rest calculations follows Equation 2. Note that $\boldsymbol{cv}$ is the context vector, which is a dynamic weighted sum of the encoder's hidden representation variable, and the weights α is related to the correlation between decoder's hidden state and the encoder's. There are three word types in response, namely, sentence function-related, topic words and other words. The word type distribution is captured by type controller network, and it identified the best word type to use in each word position in the response sentence in the final word generation process.

$$\begin{aligned} \alpha_k^i &\sim corr(\boldsymbol{s}_i, \boldsymbol{h}_k) \\ \boldsymbol{cv}_i &= \sum_{k=1}^{n} \alpha_k^i \boldsymbol{h}_k \\ \boldsymbol{s}_i &= \boldsymbol{GRU}(\boldsymbol{s}_{i-1}, e_{i-1}, \boldsymbol{cv}_{i-1}, X, l, \boldsymbol{z}) \\ y_i &\sim P(y_i|Y_{<i}, \boldsymbol{s}_i, X, l, \boldsymbol{z}) = P(y_i|y_{i-1}, \boldsymbol{s}_i, X, l, \boldsymbol{z}) \\ &= \sum_{j=1}^{3} P(type_i = j|\boldsymbol{s}_i, \boldsymbol{z}) P(y_i|y_{i-1}, \boldsymbol{s}_i, X, l, \boldsymbol{z}, type_i = j) \end{aligned} \tag{2}$$

Where for $j =$ sentence function-related,

$$P(y_i|y_{i-1}, \boldsymbol{s}_i, X, l, \boldsymbol{z}, type_i = j) = softmax(\boldsymbol{W}_j \cdot \boldsymbol{f}(\boldsymbol{s}_i, \boldsymbol{z}, l))$$

and otherwise

$$P(y_i|y_{i-1}, \boldsymbol{s}_i, X, l, \boldsymbol{z}, type_i = j) = softmax(\boldsymbol{W}_j \cdot \boldsymbol{s}_i])$$

Recognition/Prior Network Recognition network and prior network in the model is designed to learn the sentence function hidden variable $\boldsymbol{z}$ and then use the variable to learn word patterns, say combination order of words, in different sentence function. Recognition network is for training while prior network is for testing. Both of them are so-called conditional variational auto encoder (CVAE)

framework. As a matter of fact, in the training process, the posterior distribution for $P(\boldsymbol{z}|X, l, Y)$ is unknown to us, but we can approximate this distribution with a multivariate Gaussian distribution, say $\hat{P}(\boldsymbol{z}|X, l, Y) \sim N(\mu, \sigma^2 \boldsymbol{I})$, where $\boldsymbol{I}$ is a 2-dim identity matrix. With this distribution approximation assumption, the recognition network can learn the distribution of $\boldsymbol{z}$ with a simple multi-layer perceptron, say $[\mu, \sigma] = \boldsymbol{MLP}_{recognition\ posterior}(X, l, Y)$. As for prior network in testing, it's similar to the recognition one: it also assumes an approximate multivariate Gaussian distribution, say $\hat{Q}(\boldsymbol{z}|X, l) \sim N(\mu, \sigma^2 \boldsymbol{I})$, and uses a multi-layer perceptron to learn the hidden variable $\boldsymbol{z}$, say $[\mu', \sigma'] = \boldsymbol{MLP}_{prior\ posterior}(X, l)$. The loss function for recognition and prior network uses Kullback–Leibler divergence to bridge the gap between them, see Equation 3.

$$L_1 = D_{KL}(\hat{P}(\boldsymbol{z}|X, l, Y)||\hat{Q}(\boldsymbol{z}|X, l)) \tag{3}$$

Discriminator The discriminator is essentially serving as a supervisor for $\boldsymbol{z}$ to embed information about sentence function into a response. This structure will enforce $\boldsymbol{z}$ to learn sentence function's hidden features and leverage the role of the introduced hidden variable in producing responses. It is also a multi-layer perception net, say $P(l|\boldsymbol{z}) = softmax(\boldsymbol{W} \cdot \boldsymbol{MLP}_{dis}(\boldsymbol{z}))$. Its loss function is given in Equation 4.

$$L_2 = -E_{\hat{P}(\boldsymbol{z}|X,l,Y)}[\log P(l|\boldsymbol{z})] \tag{4}$$

Topic Word Selection In order to generate informative responses, topic words of posts must be extracted first and then applied to the generating process. To retrieve topic word information, a relevance score of a topic word, say y, to a given post, say $X = x_1 x_2 ... x_n$, is implemented to capture them during training, make use of them during testing. Although there is no prevailing method to calculate the exact value, it can be approximated by a summation of point-wise mutual information (PMI) [7], see Equation 5. High-score words in a response to the post are added to topic words in training process and the most relevant, say with highest score, topic is selected for a post in testing.

$$Relevance(X, y) \approx \sum_{i=1}^{n} PMI(x_i, y) = \sum_{i=1}^{n} \log \frac{P(x_i, y)}{P(x_i) \cdot P(y)} \tag{5}$$

Type Controller The type controller handles the compatibility problem between controlling sentence function and informative content. The characters or words in responses can be classified into three categories, namely, sentence function-related, topic words and other words. The type controller essentially determines the word type at each decoding position and the type distribution will be used in the decoder for response word generation. It takes decoder's hidden representation $\boldsymbol{s}_i$ and $\boldsymbol{z}$ from the prior network as inputs and estimates the word type as shown in Equation 6.

$$P(type_i|\boldsymbol{s}_i, \boldsymbol{z}) = softmax(\boldsymbol{W} \cdot \boldsymbol{MLP}_{type}(\boldsymbol{s}_i, \boldsymbol{z})) \tag{6}$$

Total Loss Function The total loss for the conversation model can be calculated as a weighted sum of all the networks within it. For Kullback–Leibler divergence from recognition/prior network, however, is multiplied by a coefficient α, which increase from 0 to 1 gradually during training, to avoid vanishing hidden variables in RNN encoder-decoder [8]. The total loss function is express in Equation 7.

$$L = \alpha L_1 + L_2 + L_3 \tag{7}$$

3 Process

3.1 Data Description

All the data are downloaded here. A preview of the training data is shown in Figure 2. There are three training/validation files, say post, response and label. The training data set size is 1963382, 0.6 million for each label, and the validation data set size is 24034. Every post has a corresponding

response, and every response has a corresponding sentence function label, one of 100, 010 and 001. Labels are generated by a self-attentive classifier with 0.78 testing accuracy.

3.2 Word Embedding

The word embedding scheme in this model is constructed from Chinese news, Baidu Baike and Chinese novels, it can be found here. In data preprocessing and post preprocessing, hot vectors, say Chinese characters or words, and 64-dim numeric vectors are converted to each other under this scheme. For instance, the code snippet below shows the data preprocessing conversions. For those words falling out of the scheme, their embedding vector will be regarded as zero vectors.

```
    print("Loading word vectors...")
    vector_model_file = 'word2vec/news_12g_baidubaike_20g_novel_90g_embedding_64.bin'
    vector_model = gensim.models.KeyedVectors.load_word2vec_format(vector_model_file, binary=True)

    embed = []
    for word in vocab_list:
        if word in vector_model.vocab:
            vector = np.array(vector_model[word], dtype=np.float32)
        else:
            vector = np.zeros((FLAGS.embed_units), dtype=np.float32)
        embed.append(vector)
    embed = np.array(embed, dtype=np.float32)
```

main.py

3.3 Model Settings

The model settings are contained in a global object FLAGS, see the code snippet below. The first three lines determine the model mode, whether training or testing, whether using GUI mode or WeChat API mode for testing. The vocabulary size determines the size of word pool for generating responses. Note that the number of hidden units are only referring to GRU in encoder-decoder framework while other network is either a multiple times of this number or simply hard coded in their scripts. The batch size is the number of training pairs fed for one batch optimization iteration. One full iteration is completed when all the batches are used to train the model once. And the training set will then be reshuffled to generate new batching scheme for next iteration. A detailed model settings can be found in Figure 3.

```
tf.app.flags.DEFINE_boolean("is_train", True, "Set to False to inference.")
tf.app.flags.DEFINE_boolean("gui_mode", False, "Interaction in GUI mode")
tf.app.flags.DEFINE_boolean("wechat_api_mode", False, "Interaction in WeChat API")
tf.app.flags.DEFINE_integer("symbols", 40000, "vocabulary size.")
tf.app.flags.DEFINE_integer("topic_symbols", 10000, "topic vocabulary size.")
tf.app.flags.DEFINE_integer("full_kl_step", 80000, "Total steps to finish annealing")
tf.app.flags.DEFINE_integer("embed_units", 64, "Size of word embedding.")
tf.app.flags.DEFINE_integer("units", 256, "Size of hidden units.")
tf.app.flags.DEFINE_integer("batch_size", 128, "Batch size to use during training.")
tf.app.flags.DEFINE_string("data_dir", "Data", "Data directory")
tf.app.flags.DEFINE_string("train_dir", "Train", "Training directory.")
tf.app.flags.DEFINE_integer("per_checkpoint", 1000, "How many steps to do per checkpoint.")
tf.app.flags.DEFINE_integer("inference_version", 0, "The version for inferencing.")
tf.app.flags.DEFINE_boolean("log_parameters", True, "Set to True to show the parameters")
tf.app.flags.DEFINE_string("inference_path", "", "Set filename of inference, default isscreen")
tf.app.flags.DEFINE_string("num_keywords", "2", "Number of keywords extracted from responses")
```

main.py

3.4 Training Evaluation

Apart from the log loss of every network, perplexity [9] and accuracy are also implemented to evaluate the training performance. Both accuracy and perplexity are derived from log loss, say $acc = \exp(-L)$, and $ppl = \exp(D_{KL})$. Since these information is ignored at the very beginning of this project and the training cost is relatively expensive, only metrics for the first 1000 steps are collected in the results.

3.5 Chat Bot Applications

With the trained model, a Chinese chat bot can be built. In this project, two applications are created, one is GUI-based chat bot and the other is WeChat-based chat bot. GUI-based chat bot is realized

straightforwardly after loading the trained model into the memory and starting to ask for inputting Chinese sentences. However, this kind of chat bot application has quite strong programmer smell, and it is not easy to be commercialized as a product. A Python package itchat provides a perfect solution by using WeChat web API. Generally speaking, this package provides a QR code to login a WeChat account. After login, a function will be executed repeatedly reading an input message from one of the account's friends and returning a message. Therefore, a Wechat Chinese chat bot is realized by using the our trained model to process text messages. [2]

4 Results

4.1 Model Convergence

As shown in Figure 11, the total training loss reduces drastically in the first 200 iterations, and the learning rate becomes quite slow after the initial convergence and the prediction accuracy even starts to oscillate at 0.125%, which is very low. Similar phenomenon can also be found in the decoder network. A more obvious convergence is found in the perplexity of the recognition and prior network, which drops from more than 35000 to 100-level and decreases slowly afterwards, see Figure 5. By contrast, the log loss of discriminator network appears to be random but the reality is after 2000 iterations, its log loss for both training and validation set is close to 0, say about 0.03, which means the discriminator has an accuracy of 97% to distinguish sentence functions from one to another. As a result, the controlled sentence function is accomplished at early stage of training, say 2000 iterations, and the result 78000 iterations are mainly used to improve decoder at a very slow speed, say the informative content of the generated responses

4.2 Training Results

The training time for first 1000 iterations is shown in Figure 4. The vertical line at the beginning is caused by the huge initial value for the time record variable. Except that, it is clear the processing time for every full iteration, say to fit the 2 million training data set, is very stable, approximately 6 or 7 seconds. However, the memory cost and the computational cost is extremely significant, 100% usage of CPU and 95% usage of 16GB memory at post stage, see Figure 13. The accuracy of recognition and prior network is 92% in the end while the discriminator network has an accuracy of 98%. However, the informative content examination is not conducted rigorously, and user experience on applications serves as an alternative, see next section.

4.3 Application Test Results

The application user experience reflects that most of the responses are informative and grammatically correct, but most of them are not appropriate, say it is replying to something else. In Figure 14, left hand side speaker is performed by the chat bot, and the right hand side is the tester. The responses still cannot avoid universal answers like "我也是| So do I", but its content does refer to some topic like "电影| Movie". Figure 15 is the back-end processing for the chat bot in WeChat. Figure 16 shows the interactions between the chat bot and the tester. It is obvious that the topic relevance and the informative content still has a large room to improve, but the sentence function and word patterns are implemented well enough already.

5 Discussions

5.1 Training Limitations

The training process is conducted on TensorFlow packages not the GPU version though. The reason for not implementing tensorflow-gpu for training is because the number of parameters in this whole model is too large for a GPU with 8GB memory to carry out. An error will be presented after the first few iterations, see Figure 17. Even using pure CPU TensorFlow, 16GB memory is still not enough at late training stage. Hence, a more reliable machine or platform should be considered for further studies. Besides, although the model is proved to be efficient in the research [1], only one model

[2] WeChat demostration can be scheduled by contacting my email.

parameter setting is carried out in this report, so the model may not be tuned to its most appropriate setting. This should also be investigated in the future, but note that the computational cost would be much higher.

5.2 Evaluation Limitations

Although a few different models regarding generating responses are studied in background research, none of them is implemented as a comparison to the conversation model proposed in this report. It is difficult to evaluate how good the model is. In this report, only the metrics for training and validation are used to evaluate the conversation model, but the content, grammar, and appropriateness are only judged at a few cases, rather than a large testing set. The evaluation process can be promoted once more Chinese-speaker judges join. So far as I know, this evaluation based on understanding could only be conducted manually. Therefore, a more rigorous evaluation scheme can be formulated to better prove the appropriateness of the conversation model.

5.3 Multi-Turn Conversation Model

One presumption of our model is that each input sentences are independent and only one-turn dialogue is handled. However, in practice, most conversations are multi-turn, hence the context of responses should also be taken into consideration. This model cannot realize the context recognition so far, but more advanced networks or a combination of networks are possible to implement the multi-turn feature. Advancements of this model can be further discussed in the future.

6 Conclusions

In this report, a conversation model with sentence function control variable is introduced based on a latest paper. It is expected to solve the problem of compatibility between sentence function controlling and informative contents. A large Weibo post-response data set is implemented for training. The training process converges quickly in the first few hundreds iterations, but the total training time lasts 169 hours (7 days) for a high-end personal computer. The training and validation results indicate that the networks for controlling sentence function obtain high accuracy while the networks for topic and decoder still have a large room to improve. Besides, two chat bot applications are implemented on WeChat and GUI successfully. More appropriate model can be trained and evaluated by overcoming hardware limitations and introducing more rigorous evaluation scheme. Advancements to enable multi-turn conversation feature can be studied in the future.

Acknowledgement

This project is completed on my own, and I have 100% contribution to the project: as discussed early this semester, Prof. Yao acceded to my one-person team proposal. The source code is retrieved from here, and I updated it from Python 2.7 to Python 3.6, also fixed a few bugs and added my own ideas in these scripts. Last but not least, thanks Miss Mandy Xu (Baidu Inc.) for her model advice, thanks Miss Peggy Wang (University of Tianjin) and Mr. Leheng Chen (HKUST) for their model testing.

References

[1] Pei Ke, Jian Guan, Minlie Huang, and Xiaoyan Zhu. Generating informative responses with controlled sentence function. 2018.

[2] Laurie Rozakis. *The complete idiot's guide to grammar and style*. Penguin, 2003.

[3] George Yule. *The study of language*. Cambridge university press, 2016.

[4] Xiang Li, Lili Mou, Rui Yan, and Ming Zhang. Stalematebreaker: A proactive content-introducing approach to automatic human-computer conversation. *arXiv preprint arXiv:1604.04358*, 2016.

[5] Zhou Yu, Ziyu Xu, Alan W Black, and Alexander Rudnicky. Strategy and policy learning for non-task-oriented conversational systems. In *Proceedings of the 17th annual meeting of the special interest group on discourse and dialogue*, pages 404–412, 2016.

[6] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

[7] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[8] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[9] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

Appendices

6.1 Figures

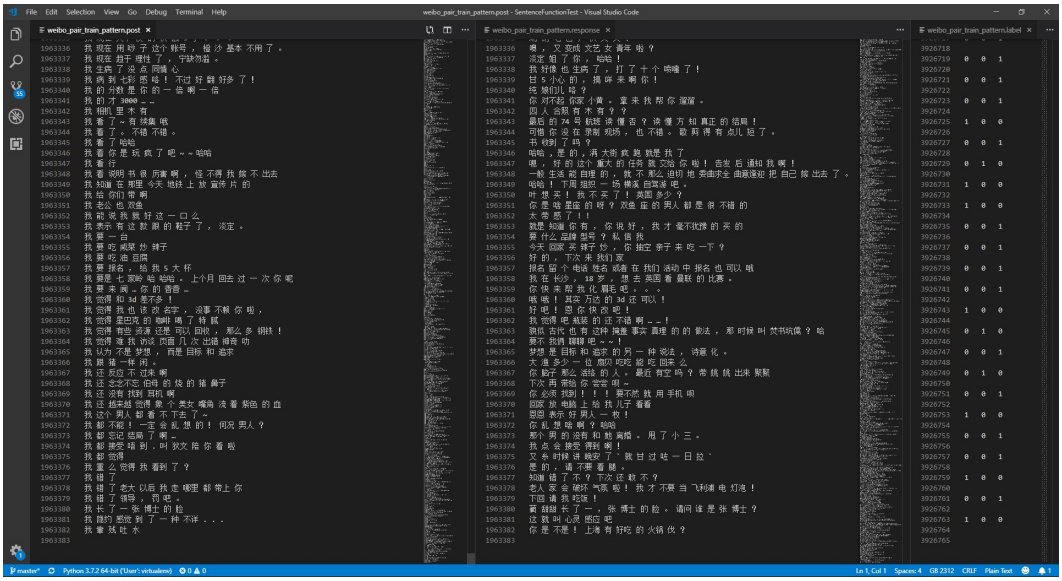

Figure 2: Training Data Preview

```
encoder/bidirectional_rnn/fw/gru_cell/gates/kernel:0: (320, 512)
encoder/bidirectional_rnn/fw/gru_cell/gates/bias:0: (512,)
encoder/bidirectional_rnn/fw/gru_cell/candidate/kernel:0: (320, 256)
   encoder/bidirectional_rnn/fw/gru_cell/candidate/bias:0: (256,)
encoder/bidirectional_rnn/bw/gru_cell/gates/kernel:0: (320, 512)
encoder/bidirectional_rnn/bw/gru_cell/gates/bias:0: (512,)
encoder/bidirectional_rnn/bw/gru_cell/candidate/kernel:0: (320, 256)
encoder/bidirectional_rnn/bw/gru_cell/candidate/bias:0: (256,)
recog_net/muvar/weights:0: (1024, 256)
recog_net/muvar/biases:0: (256,)
prior_net/fc1/weights:0: (512, 256)
prior_net/fc1/biases:0: (256,)
prior_net/muvar/weights:0: (256, 256)
prior_net/muvar/biases:0: (256,)
discriminator/pattern_fc1/weights:0: (128, 128)
discriminator/pattern_fc1/biases:0: (128,)
discriminator/pattern_logits/weights:0: (128, 3)
discriminator/pattern_logits/biases:0: (3,)
attention_keys/weights:0: (512, 512)
dec_start/dec_start_fc1/weights:0: (704, 512)
dec_start/dec_start_fc1/biases:0: (512,)
dec_start/dec_start_fc2/weights:0: (512, 512)
dec_start/dec_start_fc2/biases:0: (512,)
decoder/gru_cell/gates/kernel:0: (1280, 1024)
decoder/gru_cell/gates/bias:0: (1024,)
decoder/gru_cell/candidate/kernel:0: (1280, 512)
decoder/gru_cell/candidate/bias:0: (512,)
attention_construct/weights:0: (1024, 512)
decoder/output_projection/l_fc1/weights:0: (640, 640)
decoder/output_projection/l_fc1/biases:0: (640,)
decoder/output_projection/l_fc2/weights:0: (640, 3)
decoder/output_projection/l_fc2/biases:0: (3,)
decoder/output_projection/w_fc2/weights:0: (512, 40000)
decoder/output_projection/w_fc2/biases:0: (40000,)
decoder/output_projection/dl_fc2/weights:0: (512, 40000)
decoder/output_projection/dl_fc2/biases:0: (40000,)
decoder/output_projection/d2_fc2/weights:0: (704, 40000)
decoder/output_projection/d2_fc2/biases:0: (40000,)
Reading model parameters from Train
```

Figure 3: Model Description

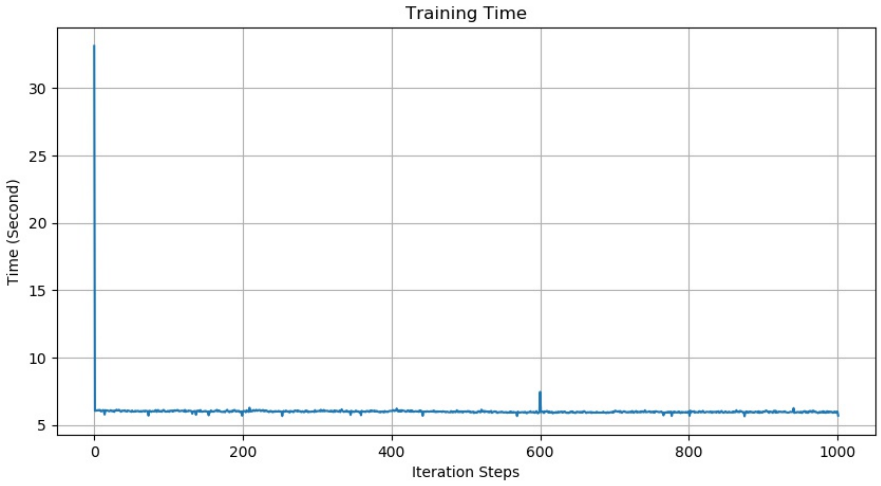

Figure 4: Training Time in First 1000 Training Iterations

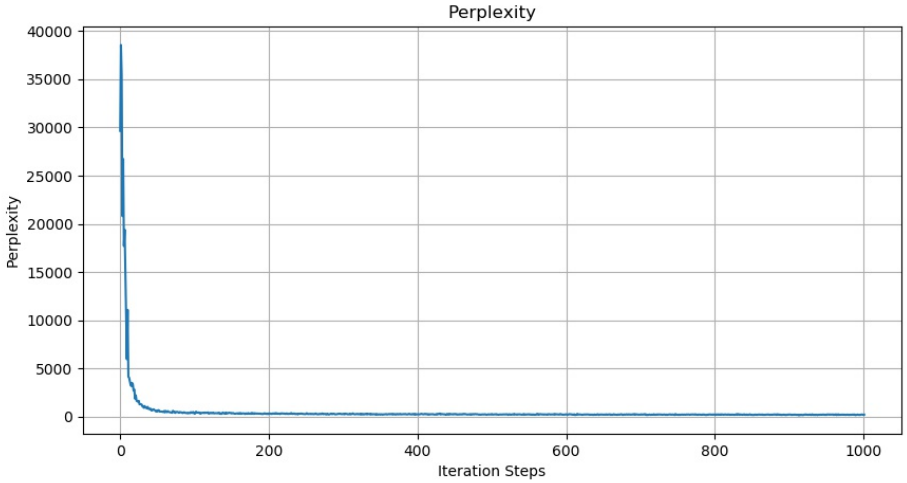

Figure 5: Perplexity in First 1000 Training Iterations

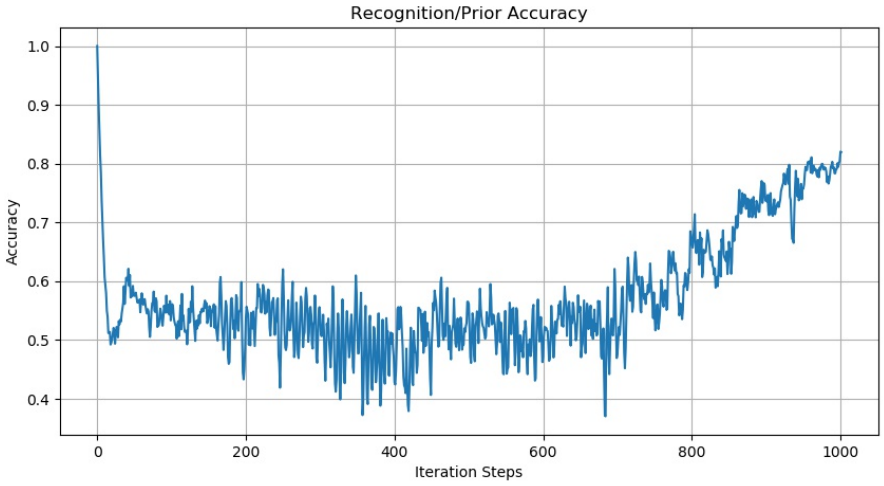

Figure 6: Recognition/Prior Network Accuracy in First 1000 Training Iterations

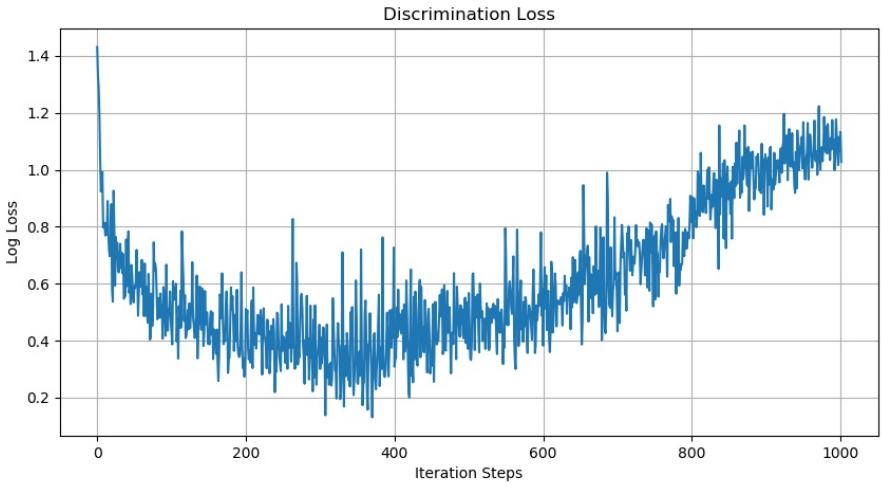

Figure 7: Discriminator Network Log Loss in First 1000 Training Iterations

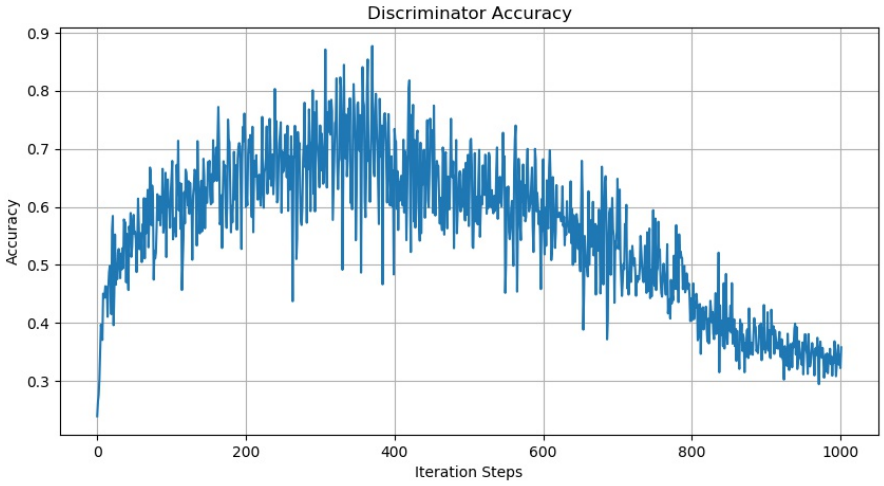

Figure 8: Discriminator Network Accuracy in First 1000 Training Iterations

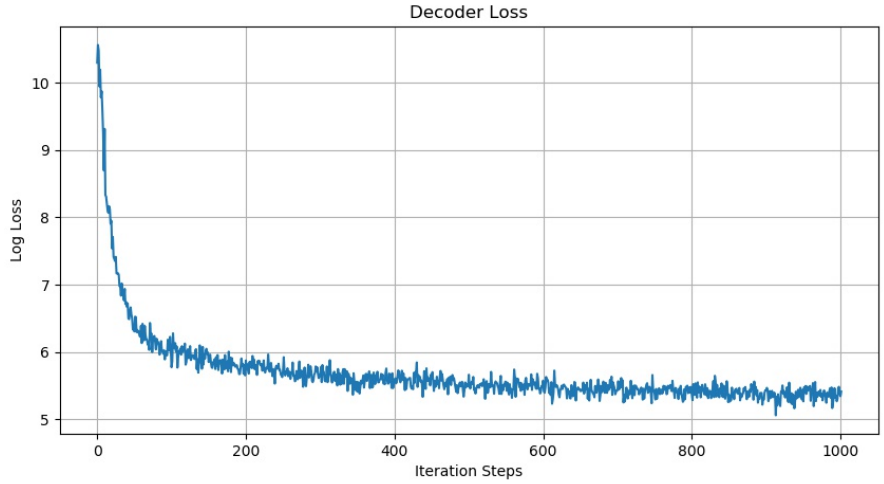

Figure 9: Decoder Network Log Loss in First 1000 Training Iterations

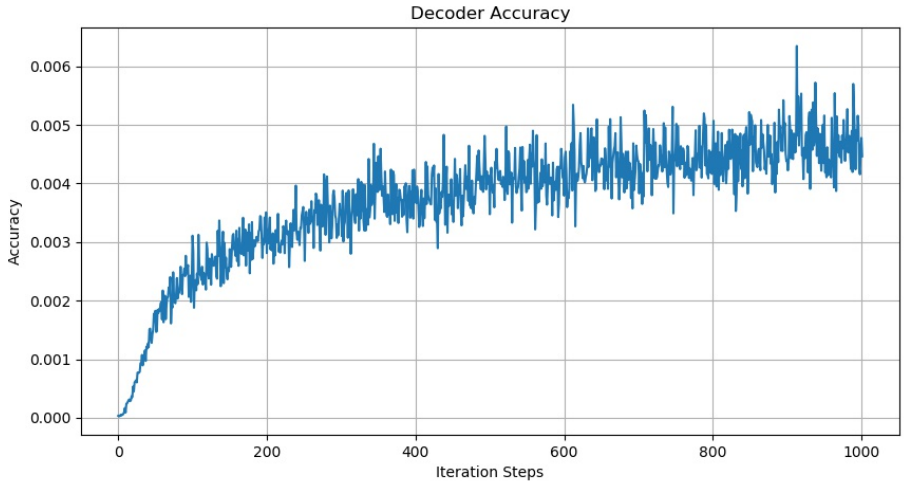

Figure 10: decoder Network Accuracy in First 1000 Training Iterations

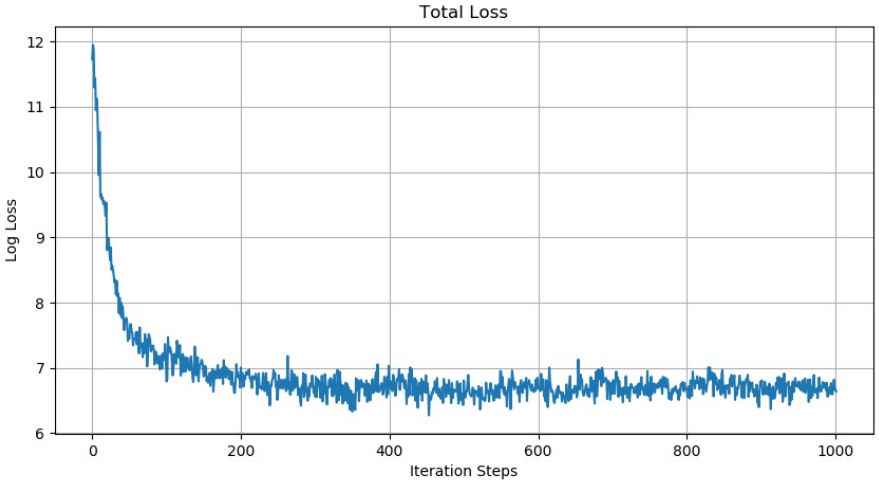

Figure 11: Model Log Loss in First 1000 Training Iterations

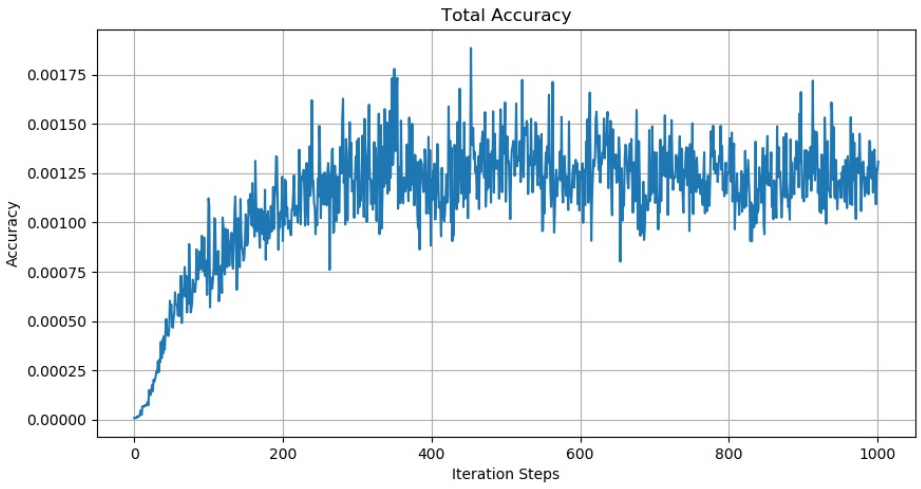

Figure 12: Model Accuracy in First 1000 Training Iterations

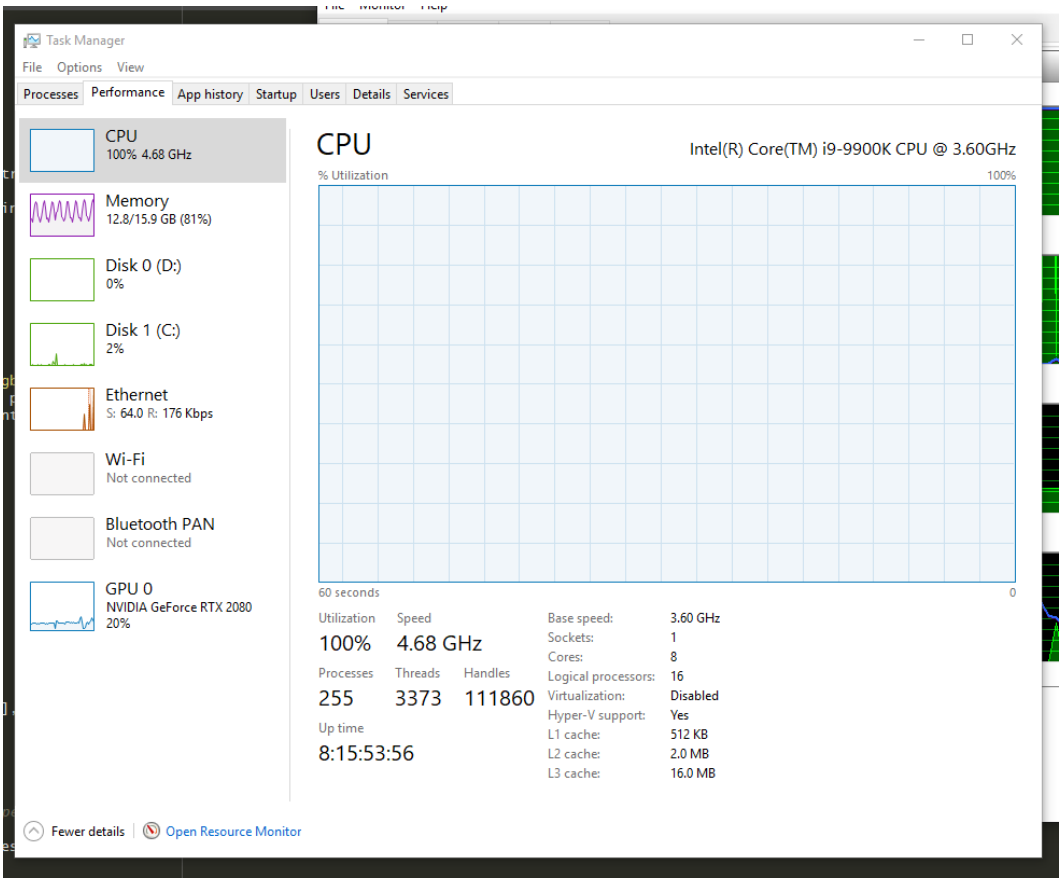

Figure 13: Computational Performance

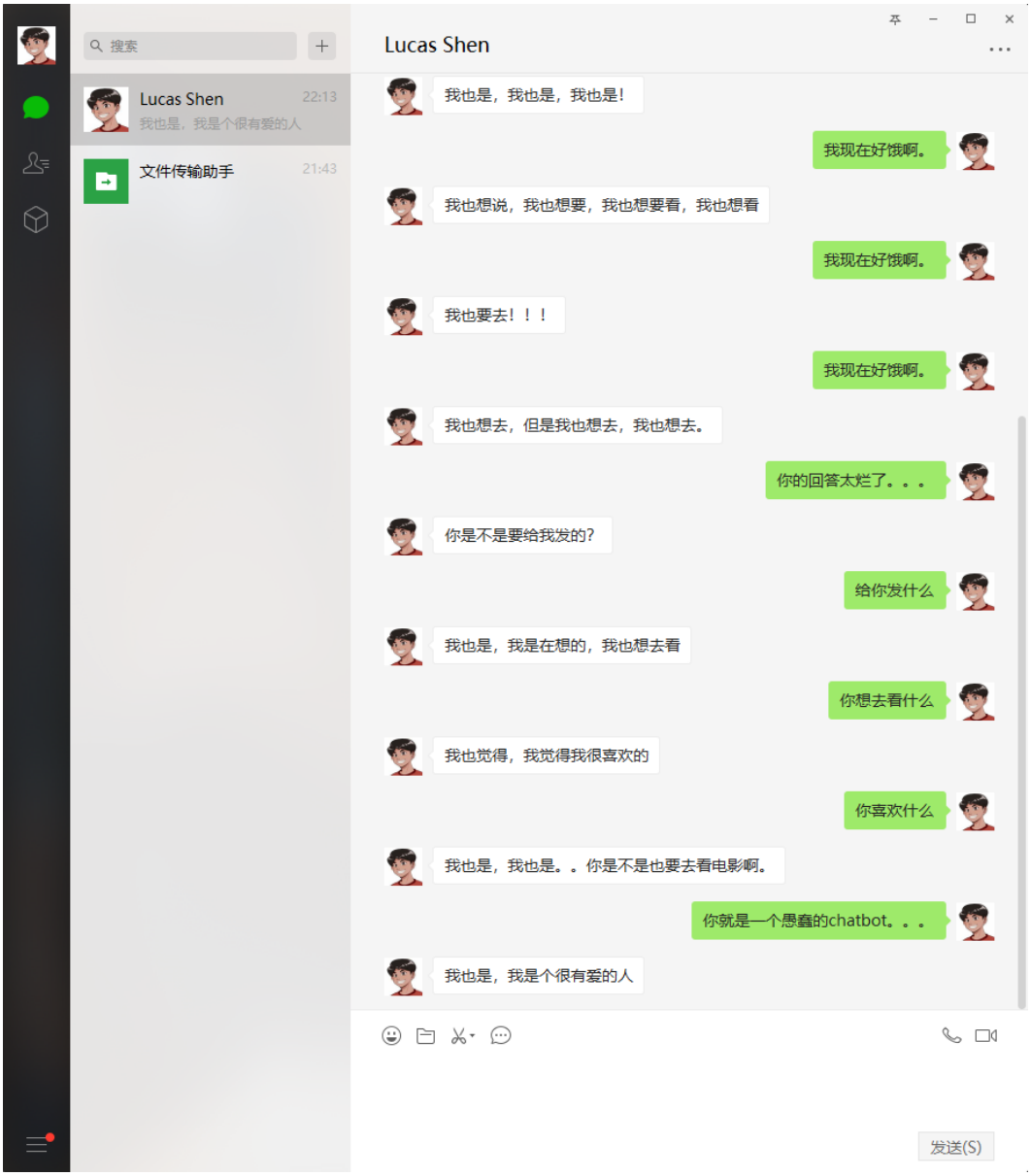

Figure 14: WeChat Application

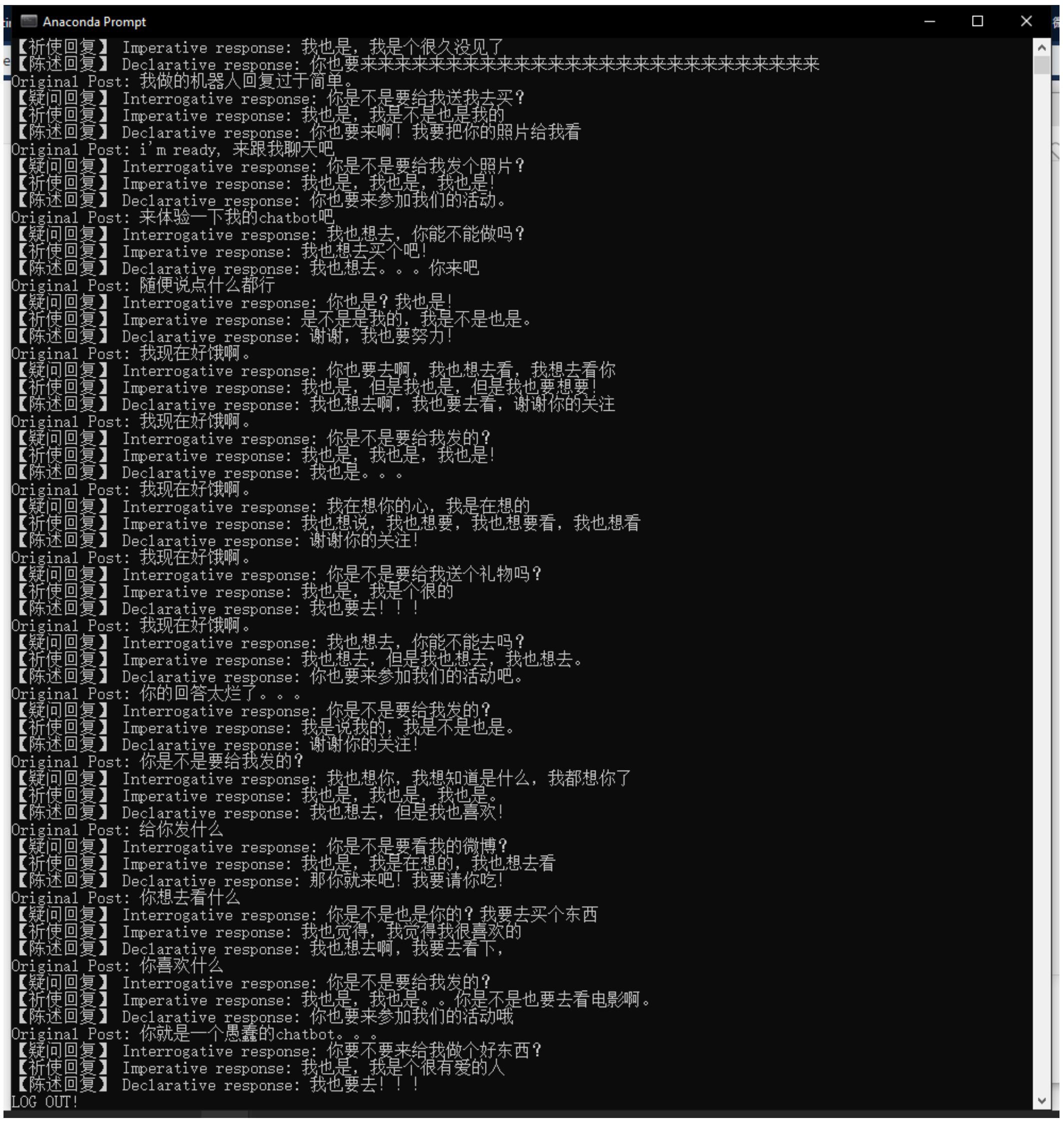

Figure 15: WeChat Application Backend

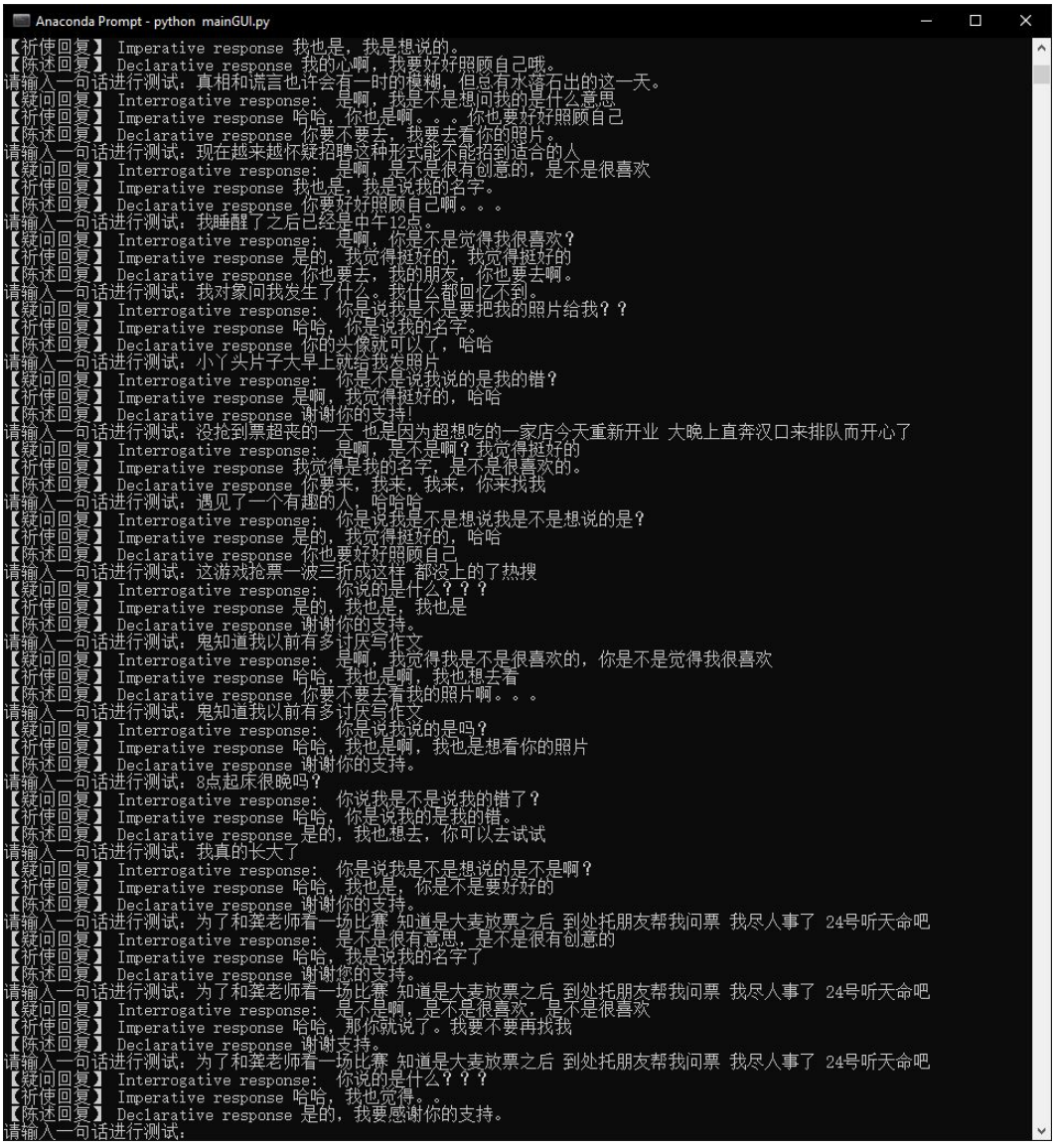

Figure 16: GUI Application

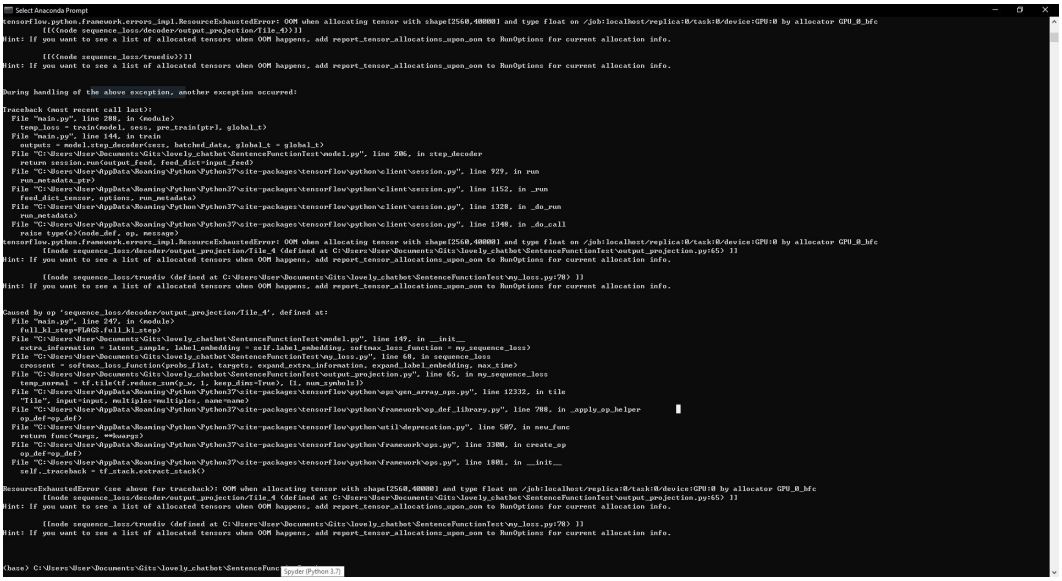

Figure 17: Error Using TensorFlow GPU