Nicholas Cichoski

Professor Aarohi Srivastava

Introduction to Natural Language Processing

17 December 2025

Final Project Writeup

Problem Statement

My project seeks to understand whether neural networks can be utilized to solve logical tasks. The goal of my project was to develop a neural network that can translate from encoded messages to decoded messages. The messages were encoded using both a substitution cipher and an Enigma cipher. For context, a substitution cipher is a cipher that assigns letters in the English alphabet to other letters and then writes out a message as normal, substituting each letter for the letter it was mapped to. The Enigma machine is an encryption device developed to encode messages during WWII. The cipher is essentially a substitution cipher that changes as each letter is entered via rotating wheels. Figure 2 charts the path that electricity takes through the machine, swapping the letter nine times after it is pressed. This incredibly complex cipher with over 5 quintillion setting combinations was famously cracked by Alan Turing, along with some of the world's best mathematicians, with the development of "the Bombe," marking an incredible achievement in computer science. It is the historical significance of this cipher that led me to choose it for my project to not only apply neural networks to a famously complex problem, but also highlight the development of the computer science field over the past 80 years.

Approach

I approached the problem with three methods. The first was a letter frequency method, which served as a baseline approach. For this method, I counted the letter frequencies in the encrypted text and ranked them by frequency. Using these rankings, I matched the letters of the encoded text with the ordered list of the most common English letters to map from the encoded scramble to regular English text. The second method I used was an encoder-decoder architecture. This was chosen in hopes that the problem could be solved by the model in a similar structure to how many language translation models utilize an encoder-decoder structure (i.e. "translate" the encrypted message to a decrypted one). This was built using PyTorch tools and follows the basic architecture shown in Figure 3. This model comprises an encoder with multi-head attention and a feed-forward layer, as well as a decoder block featuring masked attention, multi-head attention, and a feed-forward network. To increase the model size, I experimented with using up to eight stacks of encoder and decoder blocks. The third and final method I used was fine-tuning an existing model from Huggingface. I chose the Google/byt5-small model as a base for fine-tuning and tuned it through ten epochs on my training data. This model was chosen as it is a text-to-text model based on byte-level encoding, which is a key feature for solving this task. I initially ran methods two and three on my laptop's CPU, however, due to the intense computational demands, I moved them to Google Colab and ran them on the A100 GPU. All of these models can be found in the linked GitHub repository. The baseline method uses the script baseline.py; the encoder-decoder method uses model.py, and later versions run on Google Colab can be found in pytorch.ipynb; the BYT5 fine-tuning method uses byte5_model.py with later versions run on Google Colab found in fine_tune.ipynb.

Data

The data used was sourced from Project Gutenberg (Joyce, "The King James Version of the Bible", Shakespeare, Tolstoy and Maude). This can be found by clicking the link in the citations and downloading the text version of the books. It included text files of Shakespeare's complete works, the Bible, War and Peace, and Ulysses. The text was processed into a CSV file by removing the Project Gutenberg-specific text at the start and end of each file, then combining every ten lines that were over ten characters in length, and feeding the text into either a substitution cipher or an Enigma cipher. The lines were then scrambled to achieve an even mix of books represented in the training and validation data, and the results were written to a CSV file. The scripts that perform this process are process2.py and enigma2.py, both of which can be found in the linked GitHub repository. The final data sets used contained 27,529 lines of sample text with an average of 94 words per line. Figure 4 shows examples of the original text and the corresponding encrypted text for both the substitution and Enigma ciphers, which were used for training the model.

Preprocessing

The data was preprocessed using Python scripts to encode Project Gutenberg text as detailed in the data section. The process2.py script, which applies a substitution cipher, was written using the random library in Python and other basic Python commands. The script enigma2.py, which applies the Enigma cipher, uses a Python library called enigma.cipher to simulate an Enigma machine on the text and the random library to randomize settings. Within the encoder-decoder model, this text is then split into individual characters using a pre-tokenizer, which is then fed into a word-level tokenizer from the Huggingface tokenizer library. This creates a byte-level tokenization, which is reconstructed with a byte-level decoder from the Huggingface library. These are then embedded using torch.nn.Embedding and added together with positional encoding. The Byt5 model works similarly in its pre-processing. The model will take in encoded data and tokenize it using UTF-8 byte tokenization to handle the data on a character-by-character level. This data is embedded and positionally encoded before being run through the transformer structure. For both the encoder-decoder model and the Huggingface model, it was important tokenize and embed text on the character level, given that the logic of descrambling text occurs on the character level.

Experiments and Results

All three approaches were tested using character-by-character accuracy scores. The encoder-decoder model and the fine-tuned model used an 80-20 split for training and testing data to train and evaluate the models. These results are shown in Table 1. As expected, the baseline approach yielded very poor results, only achieving an accuracy of about 4% for both the substitution and the Enigma ciphers. These results were greatly improved by the encoder-decoder model, which achieved an accuracy of 63% for both the substitution and Enigma ciphers. These results were achieved by 10 epochs of training using 8 stacks of encoder and decoder blocks. While this marks a significant improvement on the baseline approach and early results from the model (which achieved an accuracy of around 35%), it is still a far cry from fully cracking the cipher. Example output from these models can be seen in Figure 5, demonstrating the model's failure to output any reasonably interpretable text. The final approach was fine-tuning Byt5 to the specific task of decoding strings. Despite 10 epochs of fine-tuning and multiple hours spent on the GPUs, the model never achieved an accuracy greater than 5% when trained on the Enigma cipher. As a result, there was no fine-tuning testing using the substitution cipher. Overall, these results were much lower than expected and constrained by time and access to GPUs. As a result, I decided to examine how the model improved by continuing to add encoder and decoder blocks to the model. Figure 6 shows the results from this experiment that trained the encoder-decoder model with 1-8 stacks of the encoder and decoder. There is a clear jump in improvement from adding 2-3 stacks; however, this quickly flatlines as the model size continues to grow, achieving less than a two-percentage point improvement by increasing from four to eight stacks. It is important to note that these models were trained and evaluated on just 10% of the training data for time's sake, and greater performance gains may have been achieved with more training data.

Analysis

These results demonstrate a relative inability for neural networks to adapt to solve logical tasks in a meaningfully efficient manner. With a relatively low accuracy and very poor output from a qualitative perspective, it is clear that the encoder-decoder model would require many more computational resources to achieve a proficient level of accuracy in decoding these messages. Practically, the computational steps required to train a model of such size are multitudes more than any algorithmic solution would take to solve the same problem, making this type of approach an incredibly inefficient way to approach a problem of this type. Additionally, an analysis of the output reveals lots of unintelligible text with loops of text. This indicates that the model is poor at outputting anything meaningful, and the greedy search algorithm provides poor token selection. While an easy fix for this might be to use top-p or top-k selection, this would stray from the goal of the task, which is to output an exact token that matches the decrypted message.

The Byt5 model struggled even more to adapt to this problem, never achieving a much higher accuracy than guesswork would have provided. This poor result is likely because the Byt5 model was too big and trained to too different a task to be quickly fine-tuned into the very specific requirements of the problem. Looking at just output length, the encoder-decoder model did a very good job of outputting the same number of characters that were inputted which is a key feature of the problem, however, the Byt5 model would output sequences far longer than what was input, indicating that the text generation or prompt-response aspects of the original model were not fine-tuned away. With significantly more fine-tuning, these issues may have been stemmed, however, better results likely would have required a model that is smaller and trained on a base task more similar to the deciphering task.

One surprising result of these experiments was the similarity in accuracy when evaluating how well the models learned the substitution and Enigma ciphers. Throughout the project, these two accuracies were consistently very close and almost the same for the largest models. This is a surprising result given that the Enigma cipher is a far more complex way to encode messages. Despite this increased complexity, there was nearly no distinction between neural networks' abilities to solve the problem. This result indicates that the problem complexity may matter far less than it might for an algorithmic approach. This result could be due to the model identifying patterns in both problems equally as well, or, in the worst case, indicate that the model is not really learning to solve these problems in the way it is supposed to. For example, if it is just guessing sequences and not "learning" to interpret the encrypted input, it would guess outputs just as well for any cipher since it is not grasping any real information from it.

While the results from the models were poor, there is some indication that sustained improvements could be achieved with increases to model's sizes. Analyzing results from increasing model size, as shown in Figure 6, reveals that there is some sustained improvement when increasing model size, although the marginal returns from continuing to stack encoders and decoders onto the model are diminishing in this experiment. Critically, these diminishing returns may be mitigated by accompanying increases in the amount of data models were trained on. Looking at the model accuracy for eight encoder-decoder stacks, there is an eight-percentage-point increase in accuracy when training on 10% of the available data versus 100% of it. While there still would likely be diminishing returns to increases in model size at a certain point, increasing the data available to train on as the size increases would likely lead to significant improvements in model performance.

Lastly, on the point of data, it should be noted that the books that make up the training data in these experiments come from vastly different lingual background and times in history. Shakespeare's complete works were written in 16th century English, War and Peace was a 19th century translation from Russian, Ulysses is a notoriously complex combination of multiple languages and writing styles, and the Bible has been translated over thousands of years from multiple different languages. These vast differences lead to different word choices, phrases, and grammatical structures being represented in the data. While this is good for generating a very robust model, it may have been better to train on more homogenized data for the sake of achieving the most accurate model.

Related Work

A 2017 paper called "Learning Enigma with Recurrent Neural Networks" (Greydanus) provides a good point of comparison for my project. In this paper, researchers developed an LSTM that reconstructed polyalphabetic ciphers in order to help solve them, obtaining over 99% accuracy on the task. Unlike my project, which used an encoder-decoder structure to recreate the ciphers in a similar structure to how models learn to translate languages, this project was based on a relatively simple RNN structure; in the researchers' experience with this problem, more complex models learned too slowly or not at all. Another key difference is that the data used for training was a random scramble of letters, then fed through the Enigma cipher. This data set completely removes any context of English words or grammatical structure from the models' training, isolating the ability to learn the cipher patterns from the ability to common English patterns. Lastly, the models in this paper were trained for much longer on much larger data sets, with their final model running for multiple days on a Tesla K80 GPU to achieve 500,000 training steps.

While the experiment in the paper was much more successful, the takeaways were similar to mine, the main takeaway being that it takes a massive amount of time and data for neural networks to adapt to intense logical tasks such as the Enigma cipher. A specifically astute observation that the researchers noted in their paper was that "the Enigma was cracked nearly a century ago using fewer computations than that of a single forward pass through our model" (Greydanus, page 2). While it is possible with enough data to solve very complex logical tasks with neural networks, it is clear from these experiments that this approach is far inferior to algorithmic approaches from a practicality and resource standpoint.

Conclusion

Overall, while the results were not sufficient to successfully decode the Enigma cipher, the project provides key insights into the application of neural networks to tasks like deciphering codes. Critically, to master these tasks, neural networks require large amounts of memory, training data, and training time. While possible, the requirements to achieve a high level of accuracy are impractical in comparison to algorithmic approaches. Additionally, solving these problems through sheer scaling of models yields diminishing marginal returns and is a poor method for achieving better results, especially in the absence of increased training time and data. Cipher complexity also appeared to make very little difference in model accuracy, indicating that models could similarly learn varying degrees of complex tasks. These results support the idea that with enough data and a large enough model to throw at a problem such as deciphering encrypted messages, neural networks are likely to develop a fairly accurate solution. However, these methods are poorly adept at taking on this type of logical challenge and incredibly inefficient.

Future Work

There are many improvements that can be made to this project. One key improvement is reconsidering the data used for the project. While books provide an easy way to quickly accumulate large amounts of text, the non-homogeneity of the data may have been a limiting factor. While I wanted the model to be able to take advantage of common word endings or grammatical phrases to assist in deciphering, the data pulled from may have been too broad to effectively do this. Additionally, instead of attempting to stack more encoder-decoder blocks to improve model performance, looking to make improvements within these blocks, such as adding feed-forward layers or adjusting the dropout rate, may have yielded slightly leaner models that would be more effective. Similarly, finding a smaller, more related model to fine-tune to this problem or significantly changing the training step size likely would have yielded better results.

The most interesting area for future work, though, is finding more efficient ways to train neural networks for these types of logic-based tasks. This project and existing literature demonstrate that given enough time and data, these problems can be solved through the application of natural language processing tools. A major point of interest is finding significant efficiency gains in the computational intensity required for solving these problems. Structuring data in a better manner and strategic decisions in model design may be paths forward to achieving a practical application of neural networks to deciphering problems and similar logic-intensive challenges.

GitHub Repository: https://github.com/ncichosk/NLP/tree/main/Project

Citations

Greydanus, Sam. "Learning the Enigma with Recurrent Neural Networks." arXiv.org, September 7, 2017. https://arxiv.org/abs/1708.07576.

Joyce. "Ulysses by James Joyce." Project Gutenberg, November 27, 2025. https://www.gutenberg.org/ebooks/4300.

"The King James Version of the Bible." Project Gutenberg, April 6, 2024. https://www.gutenberg.org/ebooks/10.

Shakespeare. "The Complete Works of William Shakespeare by William Shakespeare." Project Gutenberg, August 24, 2025. https://www.gutenberg.org/ebooks/100.

Tolstoy, and Maude. "War and Peace by Graf Leo Tolstoy." Project Gutenberg, June 14, 2022. https://www.gutenberg.org/ebooks/2600.

Figures and Tables

Figure 1. An Enigma Machine

Figure 2. Electrical Path Taken in an Enigma Machine

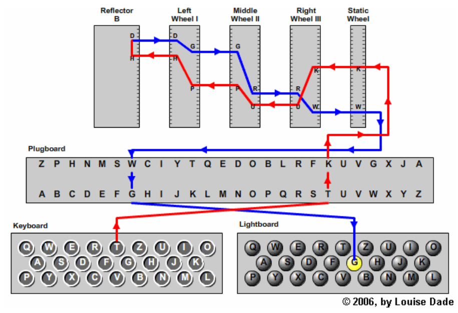


Figure 3. Encoder-Decoder Architecture

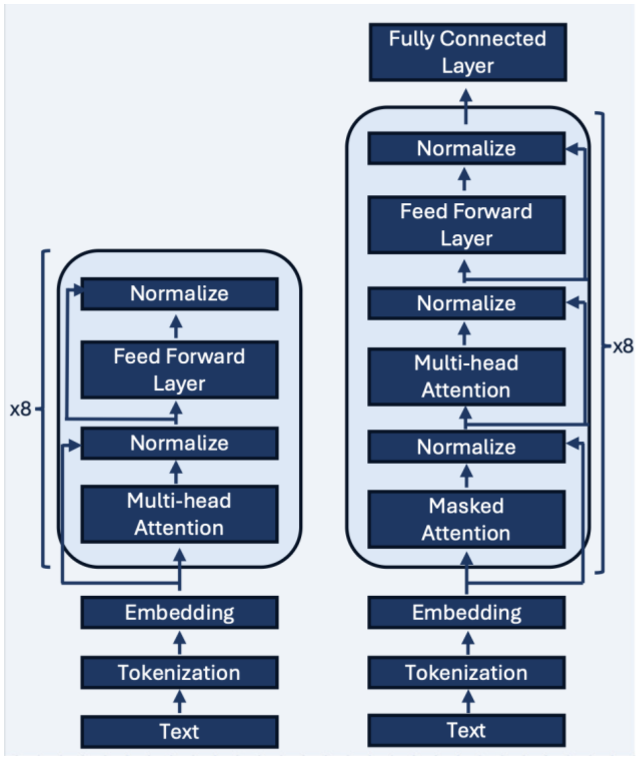
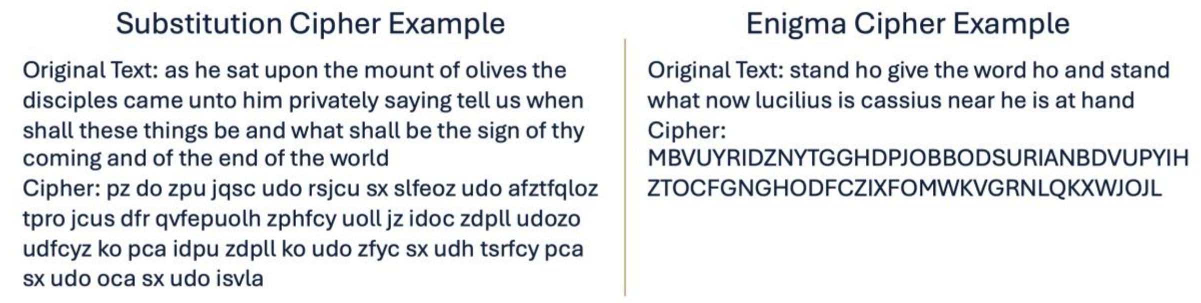


Figure 4. Data Examples



Substitution Cipher Example

Original Text: as he sat upon the mount of olives the disciples came unto him privately saying tell us when shall these things be and what shall be the sign of thy coming and of the end of the world
Cipher: pz do zpu jqsc udo rsjcu sx slfeoz udo afztfqloz tpro jcus dfr qvfepuolh zphfcy uoll jz idoc zdpll udozo udfcyz ko pca idpu zdpll ko udo zfyc sx udh tsrfcy pca sx udo oca sx udo isvla

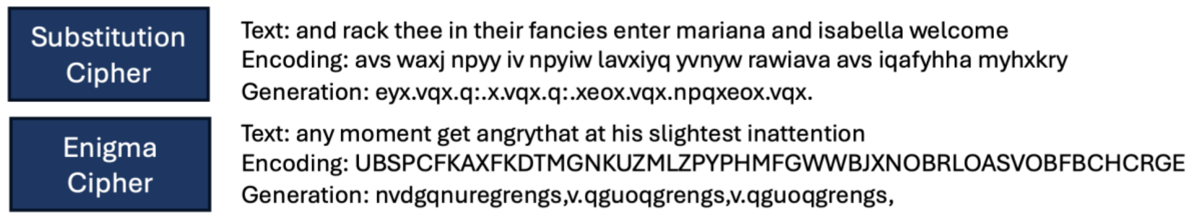
Enigma Cipher Example

Original Text: stand ho give the word ho and stand what now lucilius is cassius near he is at hand
Cipher:
MBVUYRIDZNYTGGHDPJOBBODSURIANBDVUPYIH
ZTOCFGNGHODFCZIXFOMWKVGRNLQKXWJOJL

Table 1. Model Accuracies

Model	Substitution Cipher	Enigma Cipher
Baseline	4.54%	4.09%
Encoder-Decoder	63.11%	63.12%
Byt5-Small	NA	4.45%

Figure 5. Output Text from the Encoder-Decoder Model



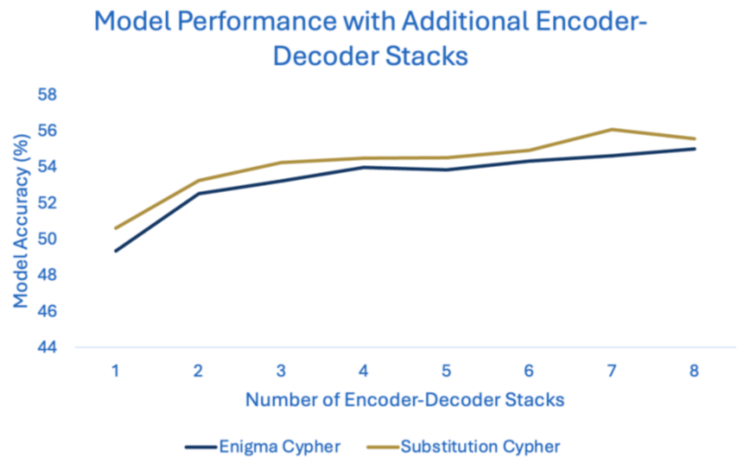
Substitution Cipher

Text: and rack thee in their fancies enter mariana and isabella welcome
Encoding: avs waxj npyy iv npyiw lavxiyq yvnyw rawiava avs iqafyhha myhxkry
Generation: eyx.vqx.q:.x.vqx.q:.xeox.vqx.npqxeox.vqx.

Enigma Cipher

Text: any moment get angrythat at his slightest inattention
Encoding: UBSPCFKAXFKDTMGNKUZMLZPYPHMFGWWBJXNOBRLOASVOBFBCHCRGE
Generation: nvdgqnuregrengs,v.qguoqgrengs,v.qguoqgrengs,

Figure 6. Performance Improvements from Increasing Model Size


Model Performance with Additional Encoder-Decoder Stacks
Model Accuracy (%)
Number of Encoder-Decoder Stacks
Enigma Cypher
Substitution Cypher