Lost in Translation: Computational Approach to Linear B Decryption with T5 Transformer Models

Oluwafemi Adelegan, Steven Lu, Georgiy Sekretaryuk

Abstract

In this paper we apply aspects of the novel language deciphering LSTM model used by MIT researchers to decipher Linear B, a syllabic language related to Ancient Greek, onto well-established encoder-decoder Transformer models. We aim to test the performance of transformer models in Linear B to Greek decipherment by applying methods of data preprocessing, transliteration, as well as improvements to various types of models, such as T5, in order to achieve successful translation of Linear B to Greek. Due to challenges in training a model from scratch in a previously-unused language (Linear B), the success of the model was calculated using Levenshtein distance between the transliterated Greek model output and the transliterated base Greek translation. In the process of testing the T5 model, we achieved a peak averaged Levenshtein distance of 0.3445 for the model performance, which is a very reasonable performance for such a lost language like Linear B. Additional hyperparameter testing and modifications to the model failed to outperform the baseline performance.

Introduction

The goal of this research, inspired by the MIT and Google researcher's paper "Neural Decipherment via Minimum-Cost Flow: from Ugaritic to Linear B", is to decipher lost languages using natural language processing techniques, such as Long Short-Term Memory (LSTM) and attention [2]. While the original paper created a well-performing LSTM model, our research focuses on assessing and comparing the performance of Transformer models pre-trained on mapping Linear B to modern Greek syllables, specifically the T5/MT5 models. The original paper from MIT reported a model performance of 67.3% which is the current highest performance model to decipher Linear B.

A lost language is defined as a language that is not used anymore. Alarmingly, languages are disappearing at an unprecedented rate, with one estimated to go extinct every 3.5 months. Projections suggest that by 2050, nearly 90% of the languages currently in use globally may face extinction, leading to a significant loss of cultural heritage and linguistic diversity. Researchers of Ancient Greek language note that Linear B, which evolved from its older parent language Linear A, comes from the Greek Bronze Age, when it was known as Mycenaean Greece. (The languages are referred to as "linear" due to the use of styluses to carve "lines" into clay tablets, as opposed to other languages which use wedges, like cuneiform.) This era spanned from roughly 1700 B.C. to 1050 B.C and was known as the first advanced Greek civilization, with a writing system using stone tablets, works of art using chiseled stone, and distinct urban organization [3]. Success in helping improve ancient Greek translation would not only be a breakthrough for historians delving into Greek culture during the Bronze Age but also contribute invaluable insights into the evolution and interconnections of languages over time.

Background (literature review or related work)

In deciphering Linear B, we employed syllabic and cognate mapping, an approach informed by understanding language evolution and the relationships between cognates. Syllabic mapping aligns with findings from "Deciphering Undersegmented Ancient Scripts Using Phonetic Priors", emphasizing the significance of phonological systems in language evolution [4]. The study "Cross-Language Distributions of High Frequency and Phonetically Similar Cognates" highlights variations in cognate frequency and similarities between similar languages, reinforcing the need for cognate-focused decipherment strategies [1]. The concept of cognate mapping aligns with "The Theory of Graphs in Linguistics", where the relationship between languages is analyzed through structural similarities [5]. These approaches provide a foundational basis for predicting language evolution, especially in the context of ancient scripts like Linear B.

The primary source of prior natural language processing work related to the topic was a paper by MIT and Google researchers, “Neural Decipherment via Minimum-Cost Flow: from Ugaritic to Linear B”. The original model used by MIT researchers is a custom LSTM model with attention, that is tasked with learning character-to-character mapping between Greek and Linear B [2]. Specifically, the model utilizes a character-based sequence-to-sequence model with a shared universal character embedding space and a residual connection. The inputs into the encoder are the words in the lost language (Linear B), and the inputs into the decoder are the words in the known language (Greek). Under the assumption that any character embedding in any given language is a linear combination of universal embeddings, the model contains a universal embedding matrix, U, a lost language character weight matrix W^x, and a known language character matrix, W^y. Then, the embedding matrix for the lost language can be represented as W^xU, and the embedding matrix for the known language can be represented as W^yU. Thus, via universal embeddings, each phrase in Linear B could be translated to Greek. The architecture can be represented as the following:

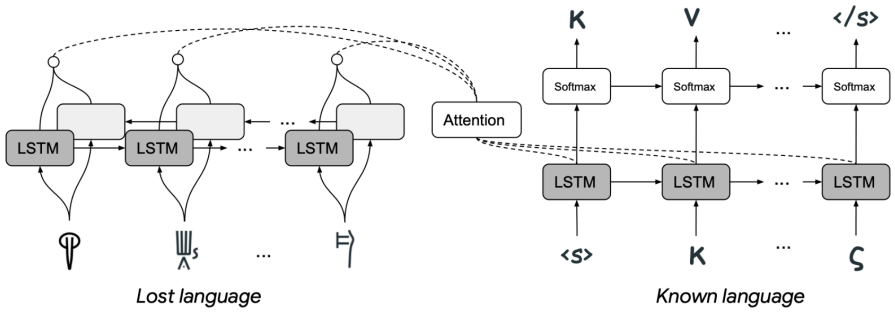


Figure 1: Model Architecture used in MIT research paper. Note that the encoder takes in lost language characters as input, and the decoder takes in known language characters as input. The problem then becomes one of minimum-cost.

The MIT paper also defines a custom loss function with regularization, which serves two goals. First, to avoid expensive insertions and deletions when translating the lost language to the known language. Second, to encourage alignment of characters to be monotonic; for Linear B,

since one character usually aligns with one to two Greek characters, the regularization function aims for the Greek characters corresponding to two consecutive Linear B characters to also be in consecutive positions. Specifically, defining p^t_i to be the position of alignment for word i in Linear B at time t, the regularization term can be expressed as such:

$$\Omega_2(\{p_i^t\}) = \sum_{t=1}(p_i^t - p_i^{t-2} - 1)^2$$

Note that when the positions of the Greek characters are aligned (e.g., the Greek characters corresponding to Linear B character 0 are at 0 and 1, and the Greek characters corresponding to Linear B character 1 are at 2 and 3), the regularization term is equal to 0, so no additional loss is incurred. Moreover, the regularization term is quadratic as to be strictly positive and more heavily punish additional insertions and deletions.

Methods (Design and Implementation)

Pre-Training & Training

First, the original dataset used by MIT researchers was downloaded. The data consisted of two datasets. The first dataset, which was the primary dataset used to train the model, consisted of 919 instances of Linear B phrases translated into their Greek counterparts. The second dataset, the “names” dataset, consisted of the same 919 instances of linear B; however, for **non-name** phrases, the Greek translation was blank. This dataset included 605 instances of names, and 414 instances of blank Greek translations. As this is an extremely low-resource dataset, it was a challenge to try and expand the number of training instances for the model.

Since languages are rarely (if ever) a one-to-one mapping between phrases, each Linear B phrase contains 1 or more translations in Greek. The number of corresponding translations in Greek ranged from only 1 to as many as 5. Each row which contained more than 1 translation from the original Linear B phrase was split into the number of rows equaling the number of translations, such that each row represented one linear B phrase and one translation of the linear B phrase. Thus, the original dataset of 919 instances was expanded to 1,429 instances. Each phrase was then tokenized via separation, with each token representing the transliteration of one character in its representative language. This step is crucial in order for the model to learn to model tokens in the respective language instead of learning English letters.

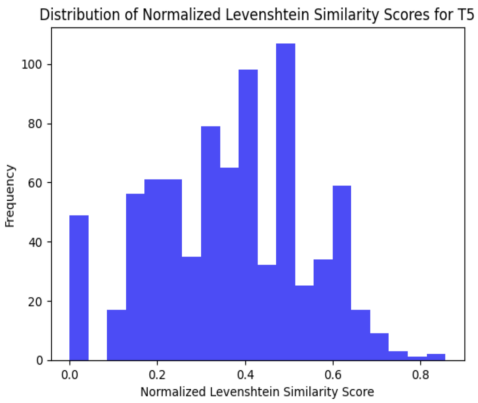
A difficulty that was encountered when attempting to train our own model was the lack of ability to handle Linear B characters. T5 in particular is trained on an English dataset and has no exposure to Linear B characters. Furthermore, from a researcher perspective, no researcher on the team understood either Greek or Linear B, making the problem exponentially more difficult by essentially translating between two unknown languages. Therefore, using a mapping of characters in Linear B to pronunciations in English, each phrase in both Linear B and Greek were transliterated into English.

Upon producing the transliteration, the research team proposed several model architectures. All of the below were tested as masked, then unmasked, then with varied amounts of epochs and learning rates:

1. For our baseline model, we had selected the T5 encoder-decoder transformer model, created by Google researchers. It was selected for its ability to handle a wide variety of NLP problems, including a wide range of text-to-text translation, sequencing, and those tasks where the relationship between input and output sequences is complex. It uses an encoder-decoder structure (that we have not modified).The encoder-decoder structure, in theory, would work well on translation tasks for unknown words, unlike a seq-to-seq model such as BERT, which was the first idea. Furthermore, an encoder-decoder structure's use of deconstructed and reconstructed context vectors should allow the model to process more complex relationships between characters in each language. This may be particularly beneficial in deciphering low-resource languages like Linear B where contextual syllabic nuances are very important to the mapping.
2. The team also proposed a novel use of T5 in which the loss function was customized to serve as a regularized value based on the edit score of the words. This loss function was proposed from the MIT paper referenced above, which we applied to the T5 architecture.

Results & Discussion

Since the language of Linear B is not totally understood, and since translation is not a black-and-white task that should be measured via sheer accuracy (correct/incorrect), it was decided that the accuracy of the translations would be measured via Levenshtein distance, a metric that calculates the distance between two strings by measuring how many changes need to be made to one string to create the other. In order to standardize the metric, the Levenshtein distance was also divided by the maximum length of the two transliterations in order to create a metric between 0 and 1. In addition, since many Linear B words actually had more than one translation into Greek, the minimum standardized Levenshtein distance was used to represent the Linear B word (which represents the score of the **best** translation). We also evaluated our models by masking and unmasking to compare the models performance.



The distribution of the standardized Levenshtein distance for the T5 model was on average left-skewed, peaking at around 0.45 and dropping off dramatically after 0.6. Note that the figure on the right represents the Levenshtein distance for the T5 model with hyperparameters for training set at: LR = 0.001, Epochs = 5. Each of the four model variations was tested and its performance measured in Levenshtein distance. In addition, we tested the number of epochs of training as 10 or 20, and the learning rate as either 0.001 or 0.005. Furthermore, the alpha for the custom loss function, which was set by the MIT researchers as 0.7, was varied between 0.5 and 0.7. The results are as follows, with the full

results in terms of loss and Levenshtein Distance for all values of learning rate, epochs trained, and alpha in Appendix A.

Table 1: Best Configurations and Performance For Each of 4 Model Configurations

Model	Learning Rate	Epochs	Alpha (if Applicable)	Masked? (Y/N)	Num_Beams	NGram Size	Best Average Regularized Levenshtein Distance
T5 baseline	0.0001	20	N/A	Y	8	4	0.3445
T5 with with Masking	0.0001	20	N/A	N	4	4	0.3991
T5 with custom loss (No Making)	0.0001	20	0.5	N	8	2	0.3561
T5 with custom loss (With Masking)	0.0001	20	0.5	N	4	2	0.3559

As evidenced by the graphs above, the best performing model was the unmodified T5 baseline with various hyperparameters. This was in support/not in support of our hypothesis. The modified models with masking and with a custom loss function performed worse than expected, by approximately 1%. The average standardized Levenshtein distance for this model at its best configuration (learning rate of 0.0001, 20 epochs trained at, 8 num_beams, and NGram size of 4) was equal to 0.3445. The loss over epochs (right) and distribution of Levenshtein distance for this configuration (below) was as follows:

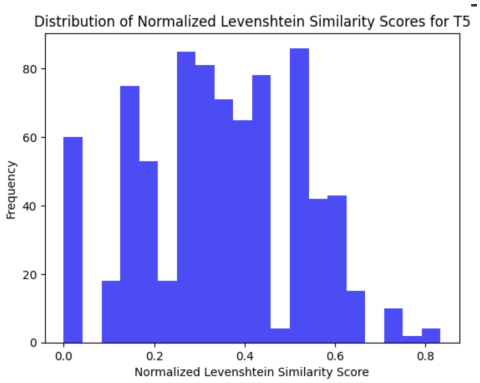


Figure 2: Normalized Levenshtein similarity score distribution for the optimal configuration of the model. Note that the distribution peaks at around 0.3 and is right-skewed; however, there are a significant amount of words (around 60) that were predicted with a normalized Levenshtein distance of 0, implying that these words were predicted with 100% accuracy.

Conclusion/Next Steps

In this project, we attempted to create a competitive model to successfully decipher a lost language, Linear B. The T5 model architecture, which heavily utilizes the encoder-decoder architecture, was originally chosen as a baseline due to its ability to pick up on complex relationships between words and to translate between languages. By applying a regularization term that both discouraged expensive insertions and deletions as well as encouraging a monotonic translation, the T5 baseline model has an average standardized Levenshtein distance of 0.3445 which is a competitive performance considering the limited resource for Linear B as a language. Also, with a custom loss function added to the model, we achieved an average standardized Levenshtein distance of 0.3561, which is in the performance range of the baseline model.

After the models implementation above, a next step would be to implement more changes from the MIT research paper to MT5, in addition to the customized loss function. For example, implementing the minimum cost flow algorithm may, in theory, the model should be able to control structural sparsity and cover the global cognate assignment with higher accuracy.

Furthermore, we would like to attempt to decode Linear A by applying the model to Linear A as input. Linear A has not been properly decoded as of the time of the writing of this paper. The two languages (Linear A and Linear B), while similar in scripts, may not be phylogenetically related (i.e., while the scripts may be related, the languages may differ entirely in structure), which may pose challenges when trying to apply learned structures in Linear B to Linear A.

Works Cited

1. Schepens J, Dijkstra T, Grootjen F, van Heuven WJB (2013) Cross-Language Distributions of High Frequency and Phonetically Similar Cognates. PLOS ONE 8(5): e63006. https://doi.org/10.1371/journal.pone.0063006

2. Luo, J., Cao, Y., & Barzilay, R. (2020). Neural Decipherment via Minimum-Cost Flow: From Ugaritic to Linear B. Retrieved from https://arxiv.org/abs/2010.11054

3. Chadwick, J., 1990. *The decipherment of Linear B*. Cambridge University Press.

4. Luo, J., Hartmann, F., Santus, E., Barzilay, R. and Cao, Y., 2021. Deciphering undersegmented ancient scripts using phonetic prior. *Transactions of the Association for Computational Linguistics*, *9*, pp.69-81.

5. van der Elst, Gaston. "The Theory of Graphs in Linguistics." (1974): 190-192.

Appendix A

Table 2: Performance in terms of Levenshtein Distance for baseline T5 model for all configurations of Learning Rate and Number of Epochs, without Mask.

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Average Standardized Levenshtein Distance (Test)
T5	0.001	10	N	4	2	0.4552
T5	0.001	10	N	4	4	0.4471
T5	0.001	10	N	8	2	0.4277
T5	0.001	10	N	8	4	0.3718
T5	0.001	20	N	4	2	0.3575
T5	0.001	20	N	4	4	0.3842
T5	0.001	20	N	8	2	0.4838
T5	0.001	20	N	8	4	0.3445
T5	0.005	10	N	4	2	0.4013
T5	0.005	10	N	4	4	0.4249
T5	0.005	10	N	8	2	0.3710
T5	0.005	10	N	8	4	0.3739
T5	0.005	20	N	4	2	0.3938
T5	0.005	20	N	4	4	0.4039
T5	0.005	20	N	8	2	0.3982
T5	0.005	20	N	8	4	0.4449

Table 3: Performance in terms of Levenshtein Distance for baseline T5 model for all configurations of Learning Rate and Number of Epochs, with Mask.

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Average Standardized Levenshtein Distance (Test)
T5	0.001	10	Y	4	2	0.4219
T5	0.001	10	Y	4	4	0.4825

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Average Standardized Levenshtein Distance (Test)
T5	0.001	10	Y	8	2	0.4583
T5	0.001	10	Y	8	4	0.4217
T5	0.001	20	Y	4	2	0.4077
T5	0.001	20	Y	4	4	0.4026
T5	0.001	20	Y	8	2	0.4269
T5	0.001	20	Y	8	4	0.4231
T5	0.005	10	Y	4	2	0.3996
T5	0.005	10	Y	4	4	0.4566
T5	0.005	10	Y	8	2	0.4185
T5	0.005	10	Y	8	4	0.4355
T5	0.005	20	Y	4	2	0.4595
T5	0.005	20	Y	4	4	0.5037
T5	0.005	20	Y	8	2	0.4743
T5	0.005	20	Y	8	4	0.4466

Table 4: Performance in terms of Levenshtein Distance for T5 model with customized regularization function for all configurations of Learning Rate, Number of Epochs, Alpha values, and no mask.

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0001	10	N	4	2	0.5	0.5067
T5 with customized loss	0.0001	10	N	4	4	0.5	0.3874
T5 with customized loss	0.0001	10	N	8	2	0.5	0.3984

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0001	10	N	8	4	0.5	0.4139
T5 with customized loss	0.0001	10	N	4	2	0.7	0.4274
T5 with customized loss	0.0001	10	N	4	4	0.7	0.4155
T5 with customized loss	0.0001	10	N	8	2	0.7	0.3834
T5 with customized loss	0.0001	10	N	8	4	0.7	0.4802
T5 with customized loss	0.0001	20	N	4	2	0.5	0.3950
T5 with customized loss	0.0001	20	N	4	4	0.5	0.3833
T5 with customized loss	0.0001	20	N	8	2	0.5	0.3561
T5 with customized loss	0.0001	20	N	8	4	0.5	0.3947
T5 with customized loss	0.0001	20	N	4	2	0.7	0.4336
T5 with customized loss	0.0001	20	N	4	4	0.7	0.3712
T5 with customized loss	0.0001	20	N	8	2	0.7	0.3722
T5 with customized loss	0.0001	20	N	8	4	0.7	0.3647
T5 with customized loss	0.0005	10	N	4	2	0.5	0.3776

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0005	10	N	4	4	0.5	0.4042
T5 with customized loss	0.0005	10	N	8	2	0.5	0.3863
T5 with customized loss	0.0005	10	N	8	4	0.5	0.3745
T5 with customized loss	0.0005	10	N	4	2	0.7	0.3895
T5 with customized loss	0.0005	10	N	4	4	0.7	0.4047
T5 with customized loss	0.0005	10	N	8	2	0.7	0.4386
T5 with customized loss	0.0005	10	N	8	4	0.7	0.4012
T5 with customized loss	0.0005	20	N	4	2	0.5	0.3984
T5 with customized loss	0.0005	20	N	4	4	0.5	0.4361
T5 with customized loss	0.0005	20	N	8	2	0.5	0.4185
T5 with customized loss	0.0005	20	N	8	4	0.5	0.4147
T5 with customized loss	0.0005	20	N	4	2	0.7	0.3905
T5 with customized loss	0.0005	20	N	4	4	0.7	0.3918
T5 with customized loss	0.0005	20	N	8	2	0.7	0.4156

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0005	20	N	8	4	0.7	0.4217

Table 5: Performance in terms of Levenshtein Distance for T5 model with customized regularization function for all configurations of Learning Rate, Number of Epochs, Alpha values, and with masking.

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0001	10	Y	4	2	0.5	0.3935
T5 with customized loss	0.0001	10	Y	4	4	0.5	0.3879
T5 with customized loss	0.0001	10	Y	8	2	0.5	0.4026
T5 with customized loss	0.0001	10	Y	8	4	0.5	0.4804
T5 with customized loss	0.0001	10	Y	4	2	0.7	0.3968
T5 with customized loss	0.0001	10	Y	4	4	0.7	0.4527
T5 with customized loss	0.0001	10	Y	8	2	0.7	0.3916
T5 with customized loss	0.0001	10	Y	8	4	0.7	0.4173
T5 with customized loss	0.0001	20	Y	4	2	0.5	0.3558

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0001	20	Y	4	4	0.5	0.3858
T5 with customized loss	0.0001	20	Y	8	2	0.5	0.3571
T5 with customized loss	0.0001	20	Y	8	4	0.5	0.3539
T5 with customized loss	0.0001	20	Y	4	2	0.7	0.3640
T5 with customized loss	0.0001	20	Y	4	4	0.7	0.3678
T5 with customized loss	0.0001	20	Y	8	2	0.7	0.3934
T5 with customized loss	0.0001	20	Y	8	4	0.7	0.3621
T5 with customized loss	0.0005	10	Y	4	2	0.5	0.4064
T5 with customized loss	0.0005	10	Y	4	4	0.5	0.4420
T5 with customized loss	0.0005	10	Y	8	2	0.5	0.4066
T5 with customized loss	0.0005	10	Y	8	4	0.5	0.3789
T5 with customized loss	0.0005	10	Y	4	2	0.7	0.3755
T5 with customized loss	0.0005	10	Y	4	4	0.7	0.3907
T5 with customized loss	0.0005	10	Y	8	2	0.7	0.382

Model	Learning Rate	Number of Epochs Trained	Masked? (Y/N)	Num_Beams	NGram Size	Alpha Values	Average Standardized Levenshtein Distance (Test)
T5 with customized loss	0.0005	10	Y	8	4	0.7	0.3775
T5 with customized loss	0.0005	20	Y	4	2	0.5	0.4229
T5 with customized loss	0.0005	20	Y	4	4	0.5	0.3994
T5 with customized loss	0.0005	20	Y	8	2	0.5	0.4186
T5 with customized loss	0.0005	20	Y	8	4	0.5	0.4092
T5 with customized loss	0.0005	20	Y	4	2	0.7	0.4108
T5 with customized loss	0.0005	20	Y	4	4	0.7	0.4056
T5 with customized loss	0.0005	20	Y	8	2	0.7	0.4056
T5 with customized loss	0.0005	20	Y	8	4	0.7	0.4151