SAPIENZA
UNIVERSITÀ DI ROMA

An AI Framework for Linear B Translation into Ancient Greek and English

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Engineering in Computer Science

Alessio Maiola
ID number 1933744

Advisor
Prof. Aristidis Anagnostopoulos

Academic Year 2024/2025

An AI Framework for Linear B Translation into Ancient Greek and English
Master Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: maiola.1933744@studenti.uniroma1.it

Dedicated to
Luigi Ricci

Acknowledgments

Abstract

Contents

Chapter 1

Overview of the Work

This thesis introduces a pipeline for automating aspects of Linear B translation by combining probabilistic cognate matching, lightweight linguistic classifiers, and a prompted post-processing step. The main components of the pipeline are:

1. A cognate matching model, following Jiaming Luo [12].
2. Auxiliary classifiers that inform and constrain the decipherment: part of speech detection, noun type classification, and inflection detection.
3. A prompt-engineering post-processor that proposes an Ancient Greek reconstruction and produces an English translation.

1.1 Chapters Overview

Chapter 2 introduces the historical context of Aegean societies and their writing systems. Both Linear A and Linear B are outlined, together with the main stages of the Linear B decipherment.

Chapter 3 develops cognate matching, a core ingredient in computational decipherment. We formalize cognates and discuss matching principles grounded in proto-root similarity and phonological/morphological regularities.

Chapter 4 presents the auxiliary tasks devised for this work and their models (part of speech detection, noun type classification, and inflection detection) together with the feature representations, training setup, and evaluation.

... PALLE PALLE ...

1.2 Data Collection

The data used in this work has been collected separately for Linear A, Linear B, and Ancient Greek.

1.2.1 Linear A Fragments

The Linear A fragments were collected from the corpora available at `https://sigla.phis.me` via web scraping. Together with the fragments, site and document-level metadata were organized into a CSV with the following columns.

Field descriptions.

- **Site**: Archaeological site name (e.g., Arkhanes, Haghia Triada).
- **Number of Documents**: Total number of documents from that site in the SigLA corpus (site-level count; repeated on each row for the site).
- **Document Name**: Canonical document identifier used by SigLA (e.g., HT 117a, ARKH 1b); side labels a/b denote tablet faces.
- **Link**: Canonical URL to the SigLA document page.
- **Type**: Document class (e.g., Tablet, Roundel).
- **Number of Signs**: Count of sign impressions recorded for the document (integer).
- **Number of Words**: Count of word tokens as segmented in the source (integer; can be 0 when only non-lexical marks are present).
- **Location**: Findspot within (or coincident with) the site when available; otherwise repeats the site name.
- **Period**: Archaeological period shorthand as in SigLA (e.g., LM I, LM IB).
- **Motif**: Iconographic motif when applicable (common on roundels/seals; often empty for tablets), e.g., bull.
- **Width (cm)**: Maximum preserved width in centimetres (cm).
- **Height (cm)**: Maximum preserved height in centimetres (cm).
- **Depth (cm)**: Thickness in centimetres (cm).

Notes. Missing values are left empty as in the source. Dimensions refer to the preserved object and follow SigLAs measurements (decimal separator .).

SigLA provides two parallel segmentations of each document: (i) a sign-by-sign view and (ii) a word/sequence view. The word view is not fully comprehensive: some isolated symbols are omitted and therefore never appear in the words export. For this reason, I keep the original sign segmentation as the authoritative layer, and use the word/sequence file to reconstruct identified sequences.

Signs CSV (per-sign records). Additional columns:

- **Sign Number**: Running index of the sign within the document (starts at 1).
- **Sign**: SigLA code of the sign (e.g., `ta`, `*118`, `AB16`, `[?]`).
- **Function**: Category of the sign (e.g., Syllabogram, Logogram) as given by SigLA.

Words/Sequences CSV (per-word records). Additional columns:

- **Sequence Number**: Running index of the sequence within the document (starts at 1).
- **Sequence**: Hyphen-separated syllabogram string exactly as in SigLA (e.g., `a-su-mi-*118`).
- **Complete**: Boolean flag from SigLA indicating whether the token is complete (True) or fragmentary/uncertain (False).
- **Length**: Number of syllabograms in the sequence (hyphen token count).

The algorithm used to reconstruct full documents, by aligning the sign and word sequences, is described in Algorithm 1.

Algorithm 1: Align sequences w_j to stream (s_t): on full match emit w_j; else emit unmatched syllabograms.

Input : For each document d: ordered $\text{SIGN\_STREAM}_d = (s_1, \ldots, s_m)$ by sign_number; ordered $\text{WORDS}_d = (w_1, \ldots, w_n)$ where $w_j = (w_{j,1}, \ldots, w_{j,k_j})$ are syllables split by "-".
Output: docs[d]: space-separated token stream.

foreach *document* d **do**
  out $\leftarrow \langle\rangle$; buf $\leftarrow \langle\rangle$; $j \leftarrow 1$; $i \leftarrow 1$
  for $t \leftarrow 1$ **to** m **do**
    if $j \le n \ \wedge \ s_t = w_{j,i}$ **then**
      if $i = k_j$ **then**
        emit w_j as a complete word (syllables joined by "-") into out; buf $\leftarrow \langle\rangle$; $i \leftarrow 1$; $j \leftarrow \min(j{+}1, n)$
      else
        append s_t to buf; $i \leftarrow i{+}1$
      end if
    else
      if buf $\neq \langle\rangle$ **then**
        emit elements of buf into out in order; buf $\leftarrow \langle\rangle$; $i \leftarrow 1$
      end if
      emit s_t into out
    end if
  end for
  if buf $\neq \langle\rangle$ **then**
    emit elements of buf into out in order
  end if
  docs[d] $\leftarrow$ concatenate tokens of out with a single space
end foreach

1.2.2 Linear B Documents

The Linear B corpus was similarly webscraped from the LiBER (Linear B Electronic Resources) archive `https://liber.cnr.it/tablet/list`. However, the LiBER archive

provides only a word-level segmentation, with no underlying sign-by-sign layer. Because many Linear B documents are fragmented and carry annotations and uncertain signs, the data required cleaning to standardize the representation of extracted syllabograms. The cleaned data was organized into three CSV files, as in the previous case.

The data of Linear B documents was organized as follows: the first CSV contains the document-level information, using the following fields.

Signs CSV (per-sign records). Part of Speech Additional columns:

- **Sign Number**: Position of the sign within the document (starts at 1).
- **Sign**: Cleaned Linear B symbol (syllabogram / logogram / numeral / uncertain), e.g., pu, `M`, `1`, `[?]`.

Words/Sequences CSV (per-word records). Additional columns:

- **Sequence Number**: Position of the word within the document (starts at 1).
- **Sequence**: Hyphen-separated cleaned symbols, e.g., `e-ri-sa-ta`, `M`, `1`.
- **Complete**: True/False flag from LiBER indicating whether the token is complete.
- **Length**: Number of syllabograms in the sequence.

The full webscraping and processing of Linear B data is detailed in Figure 1.1.

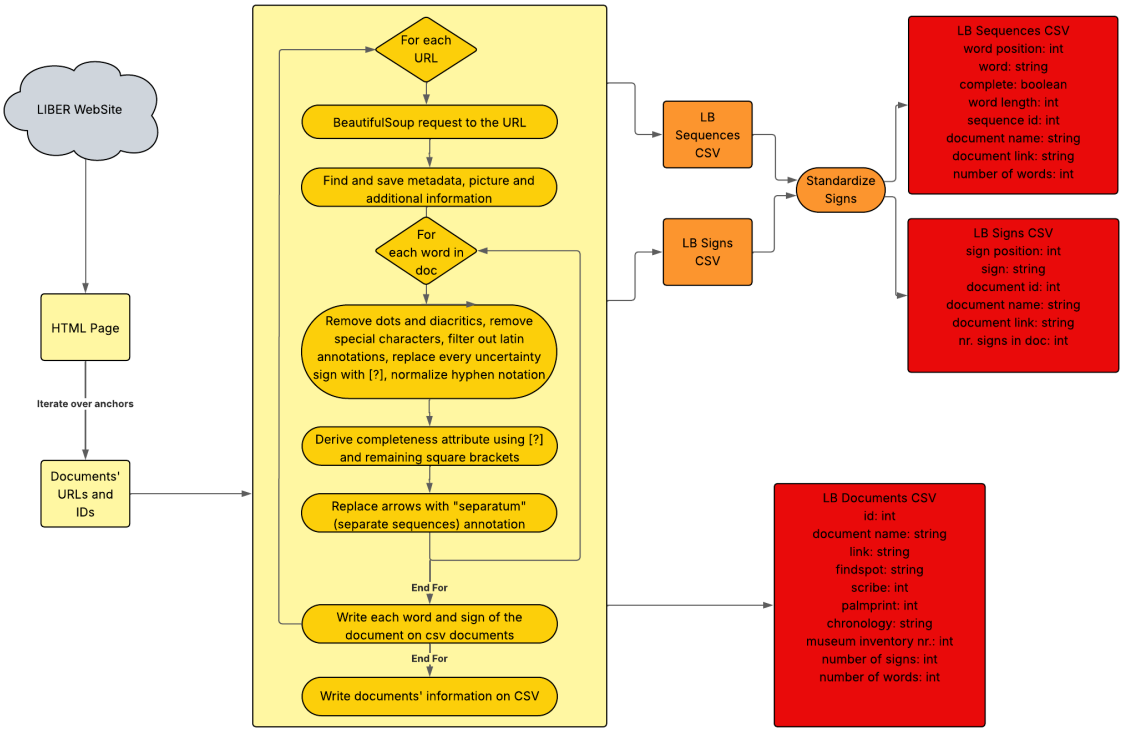


Figure 1.1. Webscraping and processing of Linear B data from the LiBER archive.

1.2.3 Ancient Greek

For Ancient Greek we required a broad lexicon, preferably skewed toward earlier forms that preserve Mycenaean reflexes more than later Classical usage. To that end, the Homeric epics (the Iliad and the Odyssey) were collected and preprocessed to a normalized orthography, described in Section 3.2.

The resulting high-coverage type list serves the brute-force cognate matching stage (Section 3.2.4), where the entire Linear B corpus is compared against normalized Homeric forms to surface plausible cognate candidates.

1.3 Related Work

The work presented in this thesis builds upon and extends prior research in historical linguistics, computational decipherment, and natural language processing.

For the historical and linguistic background of Aegean scripts, I relied on foundational texts, such as Minoan Civilization by Spyros Alexiou [1], Aegean Linear Script(s) by Ester Salgarella [14], and The Decipherment of Linear B by John Chadwick [7].

In the realm of computational decipherment, the most relevant prior work consisted of contributions on the text infilling task by Katerina Papavassileiou [13] and the cognate matching task by Jiaming Luo [12]. Notably, in both cases the models used to encode Linear B sequences are based on bidirectional recurrent neural networks (BRNNs), which are also used in this work. For the text infilling task, I also assessed the effectiveness of transformer-based architectures, noting a significant drop in performance compared to RNNs, mostly due to the scarcity of training data.

Lastly, the prompt-engineering approach to translation and reconstruction is inspired by the advantages of large language models (LLMs) and their demonstrated ability to understand and translate Ancient Greek. By supplying the information that is strictly necessary for translation and reconstruction, the Linear B corpus can be automatically translated into English.

1.3.1 Cognate Matching and Manual Decipherment

Cognate matching has been used in several prior works to support the decipherment of ancient scripts. Ventris and Chadwick themselves relied on simple cognate identification at the start of their decipherment of Linear B. Automating cognate detection is a crucial step toward scaling decipherment efforts to larger corpora and more complex scripts. Luo proposed a language-agnostic neural model for cognate matching, which I tested and extended in this thesis (Section 3.3.2).

The models output was used to generate a list of candidate cognate pairs, which were provided as input to an LLM. Alongside these, I supplied additional linguistic information useful for grammatical, logical, and syntactical analysis. The LLM was then prompted to produce an Ancient Greek reconstruction and translate it into English, using structured processing instructions, typical of manual decipherment, and a few examples. The steps of this process are described in detail in Section **??**.

Chapter 2

The Aegean Linear Scripts

Linear A and Linear B were writing systems used during the Bronze Age, primarily on the island of Crete, with some discoveries also made on the Greek mainland.

2.1 Historical Context

Around 2000 BCE, the already established Minoan civilization on the island of Crete began constructing large, complex architectural buildings commonly referred to as "palaces." These edifices served not only as administrative and economic centers, but also played important religious and ceremonial roles within Minoan society.

The founders of these palatial complexes were undoubtedly powerful landowners. Minoan society was highly organized and capable of mobilizing substantial manpower for major construction projects, such as leveling the hilltops at Knossos and Phaistos and erecting monumental palaces. [1]

Hence, this highly structured society began to feel the need for a form of administrative writing to record transactions, compile inventories, and manage other aspects of economic and bureaucratic activity.

The first form of writing developed by this society was a logographic script known as Minoan Hieroglyphics, or Cretan Hieroglyphics, attested between 2100 and 1700 BCE. The earliest and most archaic script was composed entirely of logographic symbols, which superficially resembled Egyptian hieroglyphs. It was later abandoned in favor of a linear script known as Linear A, employed between 1800 and 1450 BCE. The two systems initially coexisted for over a century, but in the following years, Linear A gradually replaced the former and became the sole writing system in use. [14]

Notably, the latest attestations of Cretan Hieroglyphs date to around 1700 BCE, when a catastrophe struck the island of Crete. All the palaces in the island's three main centers, Knossos, Phaistos, and Malia were destroyed. However, this did not lead to a cultural shift, as the palaces were promptly rebuilt, marking the passage from the Proto-palatial to the Neo-Palatial phase in Minoan history. [2]

This second phase of palace construction is the one that has survived to the present day, particularly at sites such as Knossos, Phaistos, Malia, and Zakros.

In 1450 BCE, a major catastrophe struck, probably caused by the eruption of the Thera volcano. It triggered devastating earthquakes and a tidal wave that

swept the north coast of Crete. As a result, the main centers of Minoan civilization, Phaistos, Aghia Triadha, Malia, the mansions of Tylissos and Ammisos, as well as the eastern cities of Gournia and Zakros, were reduced to ruins. Knossos also suffered significant damage, often accompanied by widespread fires. [3]

In 1400 BCE, Crete began losing its central cultural role, and the focus shifted to mainland Greece, particularly the Peloponnese. The palace of Knossos was destroyed, while major fortified citadels (fortresses) were built in places like Mycenae and Tiryns. [4]

During this period, a new linear writing system emerged. Although visually similar to Linear A, it encoded a different language: an archaic form of Ancient Greek. Its name is Linear B, and it was used from 1400 to around 1100 BCE on Crete and the Greek mainland. The Mycenaean civilization, which flourished during this period, is characterized by its extensive use of Linear B for administrative purposes, particularly in palace economies. However, the destruction in Crete should not be interpreted as a Mycenaean military takeover, but rather as a transformative phase of socio-political and cultural adaptation. [15]

Table 2.1. Chronological framework of LA and LB [14]

Chronology	Crete			Mainland		
High Dating	**Pottery Phase**	**Cultural Phase**	**Scripts**	**Pottery Phase**	**Cultural Phase**	**Scripts**
1900–1800	MM II	Proto-Palatial	CH; LA	MH III	–	–
1800–1700	MM III		CH; LA	MH III		–
1700–1600	LM IA	Neo-Palatial	LA	LH I	Early Mycenaean	LA
1600–1450	LM IB		LA	LH IIA		?
1450–1400	LM II	Final-Palatial	LA?	LH IIB		?
1400–1375	LM IIIA1		LB	LH IIIA1	Late Mycenaean	LB
1375–1300	LM IIIA2	Post-Palatial	LB	LH IIIA2		LB
1300–1200	LM IIIB		LB	LH IIIB		LB
1200–1050	LM IIIC		–	LH IIIC		LB

2.2 The main sites

The main sites where Linear A documents have been found are located on the island of Crete. These include Knossos, Phaistos, Aghia Triada, Zakros, Khania, Tylissos, and Malia.

Figure 2.1. Sites of Linear A fragments in Crete.[1]

Linear A was more widespread, covering completely Crete and the Aegean Islands and reaching the Greek mainland. The main attestations of Linear A on the Greek mainland are very limited and generally considered sporadic and isolated. At Mycenae, a few Linear A inscriptions have been found, likely as a result of commercial or cultural exchanges with Crete. Similarly, some fragmentary finds have been uncovered at Tiryns, probably also related to trade or contacts with Minoan Crete.

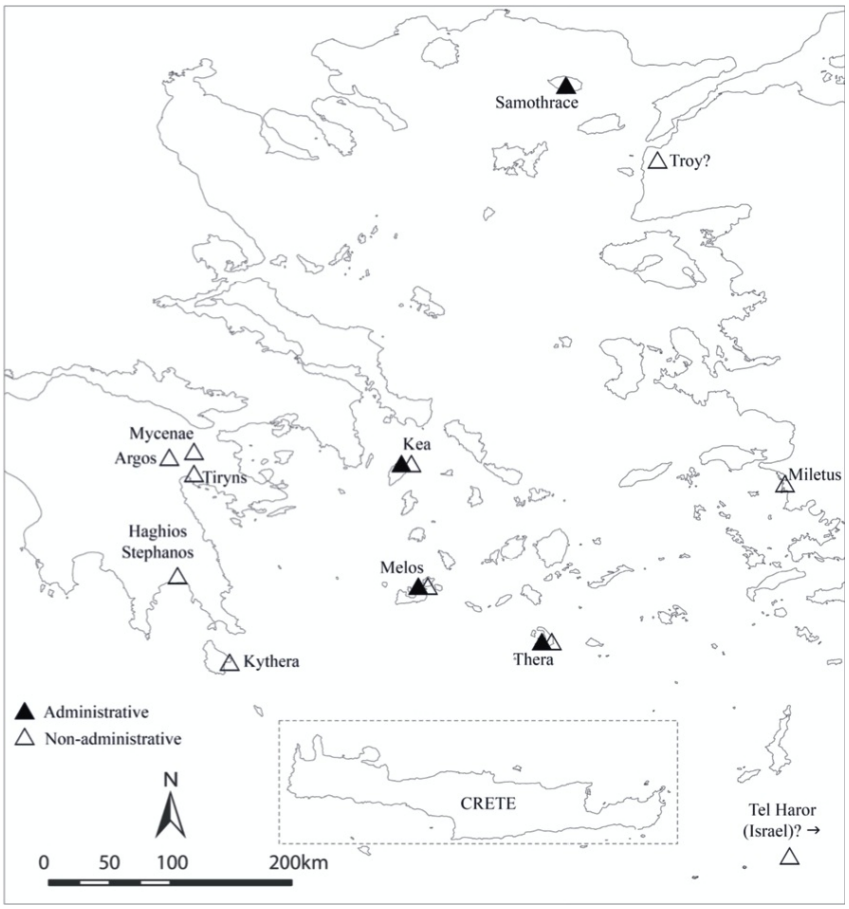


Figure 2.2. Sites of Linear A fragments in the Greek mainland.[2]

In contrast, Linear B is extensively attested on the Greek mainland, particularly in the Peloponnese, reflecting its administrative function during the Mycenaean period. Major sites where Linear B documents have been found include Mycenae, Tiryns, Pylos, Thebes, and Athens. Additionally, significant findings of Linear B tablets have been made in Crete, especially at Knossos and Khania.

The corpora of the two writing systems are relatively small, with Linear A consisting of approximately 1,400 documents, while Linear B comprises around 6,000 documents. Another notable difference is that Linear A was more widely used for non-administrative purposes, particularly in religious contexts, whereas the number of non-administrative Linear B documents is considerably more limited.

[1]Figure 2.1 prepared by Yannis Galanakis and Ester Salgarella.
[2]Figure 2.2 prepared by Yannis Galanakis and Ester Salgarella.

[14]

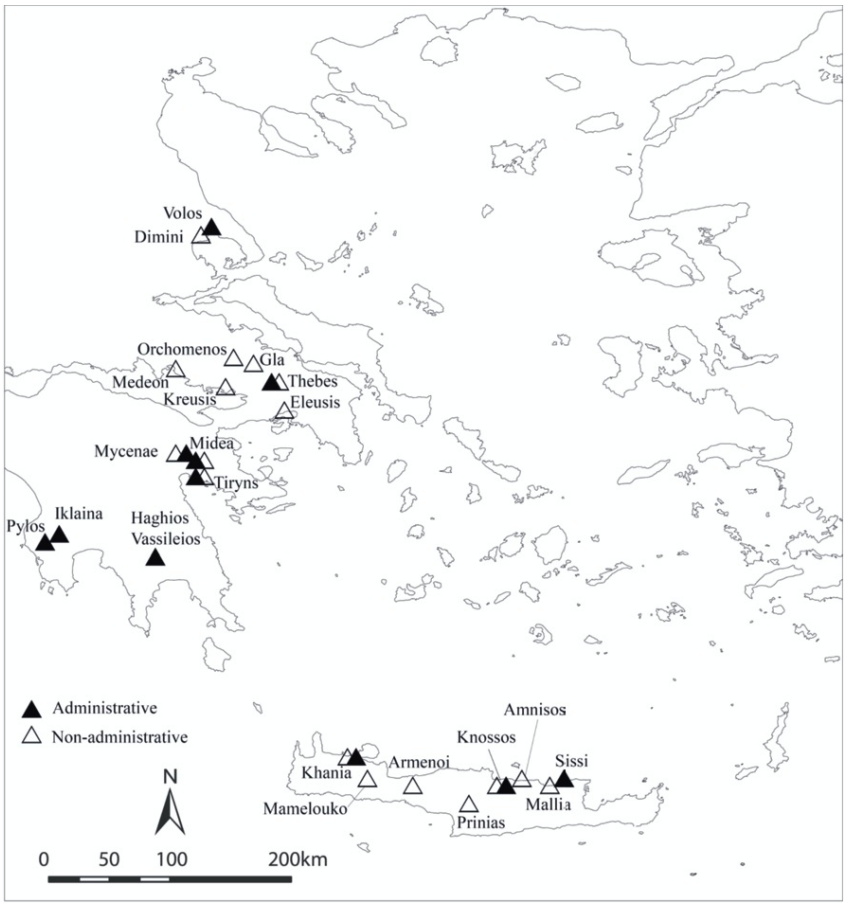


Figure 2.3. Sites of Linear B fragments in Crete and the Greek mainland.[3]

2.3 Linguistic features

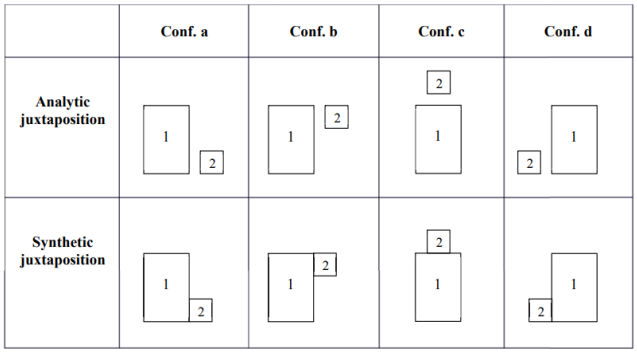
The two writing systems are characterized by similar structural features, reflecting the connection between Linear A and Linear B and the derivation of the latter from the former.

The primary similarity between the two scripts lies in their syllabic structure, which constitutes a defining feature of both writing systems. Both Linear A and Linear B are syllabic scripts, meaning that each symbol represents a syllable rather than an individual letter or a full word. In addition to syllabic signs, both systems incorporate a set of logograms: symbols representing entire words or concepts.

This logographic component is particularly prominent in Linear A, where a significant number of signs are used to denote specific objects, actions, or concepts, often associated with administrative and religious contexts. By contrast, Linear B employs a more restricted set of logograms, reflecting its primary function in

[3]Figure 2.3 prepared by Yannis Galanakis and Ester Salgarella.

administrative record-keeping. Notably, the logograms used in Linear A were generally not inherited by Linear B, with a single exception: the logogram for "wool" (MA+RU), which is attested in both scripts. However, the principles governing the formation of logograms remained unchanged, as in both scripts they are formed by juxtaposing or combining two or more signs, either horizontally or vertically.



(a) Logogram construction criteria.

(b) LA logogram for wool.

Figure 2.4. Logograms in Linear A and Linear B.

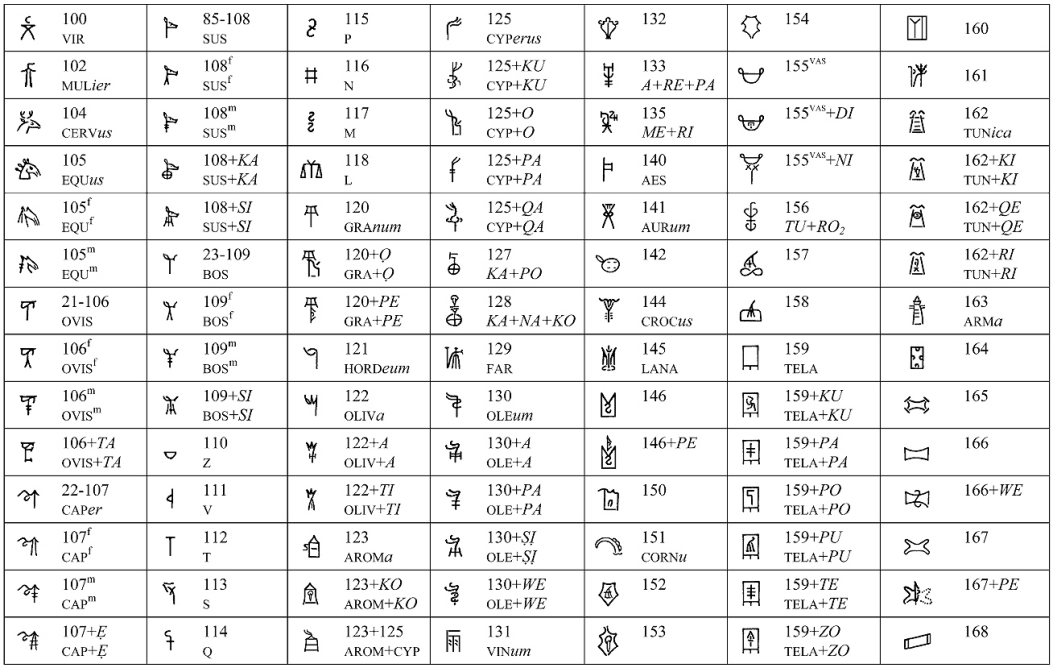


Figure 2.5. Linear B logograms (symbols 100-168).

Figure 2.5 illustrates how logograms in Linear B can also incorporate syllabograms. In these cases, the syllabogram is referred to as an adjunct and typically serves to qualify or specify the meaning of the logogram. Moreover, the use of adjuncts is significantly more frequent in the Knossos corpus than on the Mainland, suggesting a possible continuity with Linear A, where isolated signs with semato-

graphic value appear more commonly. [16]

Furthermore, a substantial portion of the Linear A syllabary is shared with Linear B, with approximately 72% of Linear A signs being identical to those used in Linear B. This overlap also illustrates continuity in symbol creation and in the assignment of phonetic values between the two systems.

Figure 2.6. All Linear B syllabograms with the associated phonetic values.

As observed in Figure 2.6, signs referring to the same vowel exhibit recurring patterns, a characteristic feature of syllabic scripts also evident in Linear A. One

of the most debated assumptions regarding the relationship between Linear A and Linear B is the principle of homomorphy and homophony. This principle posits that signs which are visually similar (homomorphy) in both scripts also share the same phonetic value (homophony), representing the same syllable. [14]

This observation has led to the widely accepted conclusion that Linear A encodes a language fundamentally different from Linear B, with the latter used to represent an archaic form of Ancient Greek. Consequently, although scholars are able to phonetically transcribe Linear A inscriptions, the language remains undeciphered and its meaning unknown.

2.4 The decipherment of Linear B

Ever since the discovery of the first Linear B tablets in 1900 by Sir Arthur Evans at Knossos, the script has been a subject of intense scholarly interest. Evans himself introduced the classification of Aegean scripts that is still used today. He also made the earliest attempts to decipher Linear B, though without success.

The breakthrough in understanding Linear B came after World War II, following major discoveries at the site of Pylos in 1939, which uncovered a large number of tablets and inscriptions. A key figure in the decipherment of Linear B was Michael Ventris, a British architect and amateur linguist. Ventris, in collaboration with philologist John Chadwick, succeeded in deciphering the script in 1952, demonstrating that it encoded an early form of Ancient Greek.

2.4.1 The knowledge before the decipherment

Before the decipherment of Linear B, scholars attempted to understand the script by studying it in isolation and by comparing it with known languages. One of the earliest hypotheses focused on the similarity between the Linear B syllabary and the Cypriot syllabary, a syllabic script used from about the eleventh to the fourth centuries BCE. The latter, which was also used to write an archaic form of Greek, shared a significant number of signs with Linear B.

However, this resemblance proved misleading in several respects. Although many signs appeared visually similar, as can be observed in Figure 2.7, their phonetic values often differed between the two systems. Moreover, the scripts treated grammatical suffixes differently. In Linear B, grammatical suffixes were frequently omitted, whereas this was not the case in the Cypriot syllabary, where the syllabogram "se" was regularly employed to indicate word endings.

Because "se" appeared regularly in the Cypriot syllabary but was rare in Linear B, early researchers concluded that Linear B could not represent the Greek language. In particular, Arthur Evans was convinced that the Minoan civilization was entirely distinct from the Mycenaean Greek world. This assumption, reinforced by Evans's authority and influence, contributed to delaying the recognition of the script's true linguistic nature. [7]

An example of this is the word "anthropos" (Ancient Greek: ἄνθρωπος), which appears in Linear B as "a-to-ro-qo" (), while in Cypriot it is spelled with the "se" suffix as follows: "a-to-ro-po-se" ( , written from right to left).

Linear B	Cypriot	Value in Cypriot
		ta
		*lo*
		*to*
		*se*
		*pa*
		*na*
		*ti*

Figure 2.7. Similarity between Linear B and Cypriot syllabary.

Evans had already established that most documents were administrative records. However, any attempt to decipher the script was hindered by unreliable underlying assumptions, while many aspects of the language remained unknown.

The most significant contribution came from Dr. Alice Kober, an American linguist. She was the first to approach the script methodically in order to uncover the nature of the underlying language. She sought to answer fundamental questions, such as whether the language was inflected or whether specific forms were used to indicate gender and number.

Through her careful analysis, she was able to identify grammatical patterns, including distinctions between masculine and feminine nouns, as well as examples of inflection. These results were achieved through a systematic study of repeated sign groups and their contexts, which she meticulously documented in her notebooks. Her work challenged Evans's assumptions about the non-Greek nature of the language and paved the way for Ventris' later decipherment. The outcome of her research was a set of "triplets", groups of three related words differing only in their endings, which provided critical evidence of an inflected language structure and were later referred to as "Kober's triplets". Some examples of these triplets are shown in Figure 2.8. [8]

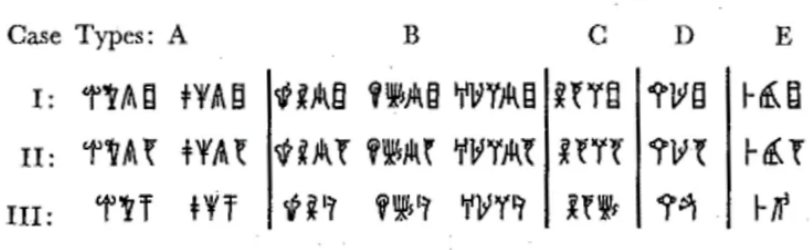


Figure 2.8. Kobler's triplets examples.

2.4.2 The decipherment by Ventris and Chadwick

The initial phase of the decipherment of Linear B involved the identification of numerical signs and easily recognizable logograms. This preliminary work was largely accomplished by Sir Arthur Evans, who successfully identified the most frequently occurring logograms and reconstructed the basic features of the numerical system. Notably, the Linear B script lacked a symbol for zero, but included fractional signs and was based on a decimal structure.

Building on these foundations, scholars began to assign tentative phonetic values to individual syllabograms through contextual analysis. By examining recurring patterns and the placement of signs within administrative texts, it became possible to propose the phonetic values of certain symbols. For instance, words such as "total" and "sons," which regularly appeared in similar tabular contexts, offered valuable clues for these early phonetic assignments.

However, the construction of a comprehensive and reliable grid of syllabograms was not feasible until the publication of the Pylos tablets in 1951. This newly available corpus significantly expanded the body of evidence, enabling more systematic linguistic analysis.

Michael Ventris began his work on the decipherment in 1950, initially by circulating a questionnaire among scholars to gather views on the possible nature of the language encoded in Linear B and its potential relationship to Linear A or the Cypriot syllabary.

Realizing that scholarly opinion was divided, Ventris adopted a combinatorial and structural approach. He explored positional patterns of signs within words, aiming to deduce their possible function or phonetic value. By analyzing the frequency and distribution of certain symbols, particularly those likely to represent vowels, he was able to identify a subset of syllabograms corresponding to pure vowels.

A key breakthrough came from the identification of inflectional endings, many of which were made possible by the foundational work of Alice Kober. Kober had demonstrated, through rigorous tabulation of sign groups, that the language encoded by Linear B exhibited an inflectional structure. Ventris incorporated these findings into his research and began constructing a syllabic grid, updating it continually in light of new phonological rules and morphological patterns.

For example, he correctly identified the syllabogram  as representing the syllable si, based on its frequent use in noun declensions and verb conjugations. Its function mirrored that of the Ancient Greek suffix σι, which appears in both oblique noun forms and verbal endings.

Additionally, place names proved particularly helpful, especially when accompanied by adjectival derivatives. These offered valuable comparisons with known Greek toponyms and morphological structures.

As Ventris refined his hypotheses and incorporated increasingly sophisticated linguistic deductions, especially regarding inflectional patterns and suffixes, he was able to assign phonetic values to a majority of the signs. Through this methodical approach, he successfully constructed a nearly complete syllabary grid. Most of his assignments proved to be correct, with only minor exceptions that were later adjusted through collaborative efforts with John Chadwick and subsequent scholarly review. [9]

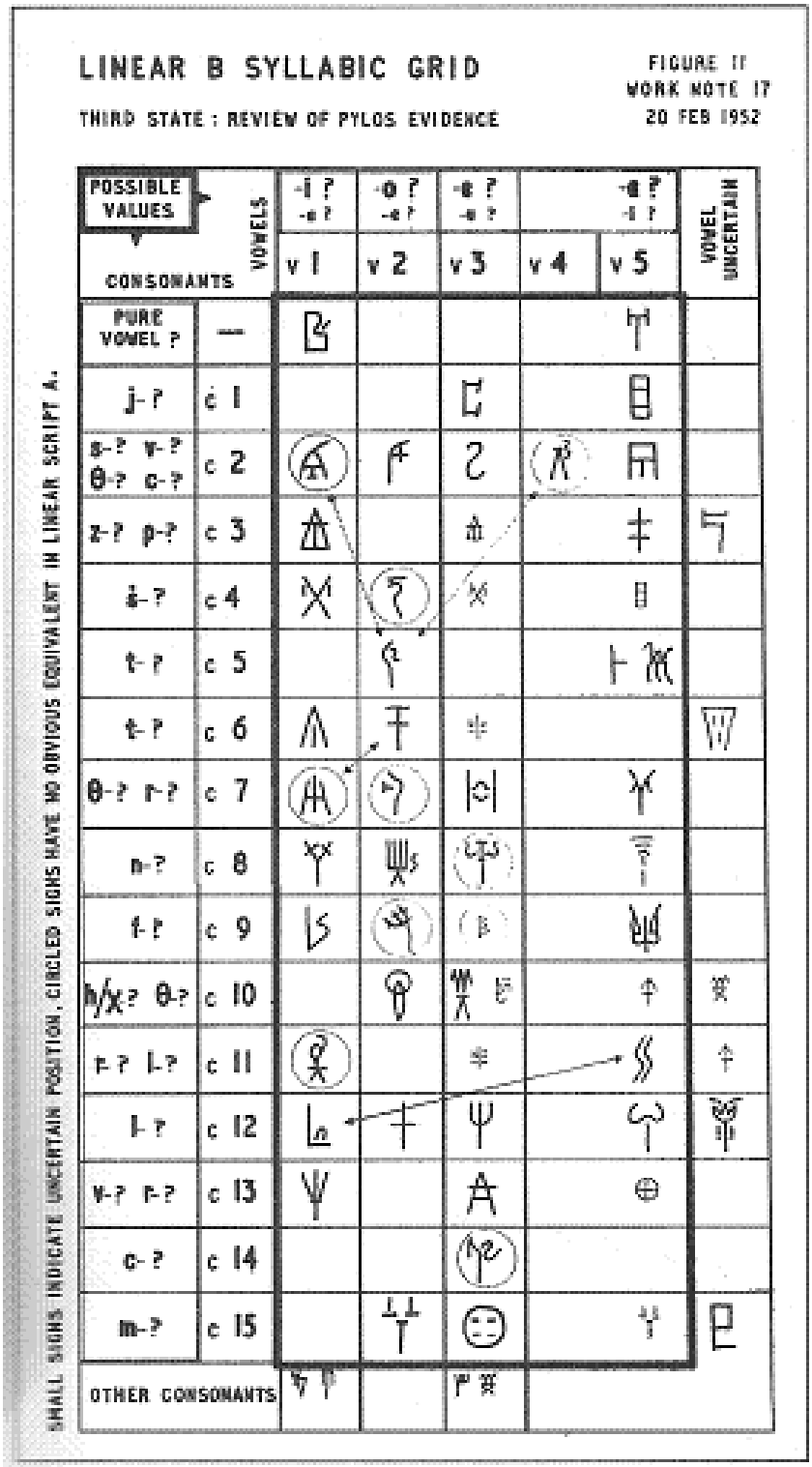


Figure 2.9. Ventris' syllabary grid.

Together, Ventris and Chadwick continued to investigate how the script encoded the phonology of the Mycenaean Greek language, recognizing that Linear

B imposed structural constraints on how sounds were represented. The syllabary, like many early writing systems, lacked a one-to-one correspondence with spoken Greek. To better understand these adaptations, they analyzed the principles governing the script's orthography, namely how Greek words were transcribed within the limitations of the Linear B system.

The following rules summarize the most significant phonological and orthographic conventions that Ventris and Chadwick identified in the course of their research:

1. The script distinguishes five vowels: *a, e, i, o, u*. Vowel length, however, is not indicated.

2. The second component of diphthongs ending in *-u*, such as *au, eu, ou*, is represented explicitly.

3. In diphthongs ending in *-i* (e.g., *ai, ei, oi, ui*), the second component is generally omitted, except:

   - when it occurs before another vowel, in which case it is represented as *y*;
   - in the initial syllable *ai*, where the full diphthong is retained.

4. Glides occurring between a front vowel and a following vowel are indicated:

   - the glide after *i* is written as *j*;
   - the glide after *u* is written as *w*.

   These sounds are typically omitted in Greek alphabetic spelling.

5. The script represents twelve consonants:

   - **j**: used only to represent diphthongal *i* or as a glide (see point 3);
   - **w**: corresponding to the archaic Greek digamma (*ϝ*), pronounced as in English *w*;
   - **d, m, n, s**: approximately as in later Greek and English;
   - **r**: corresponding to Greek *l* and *r*;
   - **z**: corresponding to Greek *z*; its exact phonetic value in Mycenaean Greek remains uncertain;
   - **p, t, k**: representing both plain and aspirated stops, as the script does not distinguish aspiration;
   - **q**: representing a series of labio-velar stops (*kw, gw, khw*), which later disappeared from Greek but were preserved in Latin (e.g., *quis*, *unguem*).

6. The script does not represent aspiration; thus, aspirated consonants (*ph, th, kh*) are not distinguished from their unaspirated counterparts.

7. The consonants *l, m, n, r, s* are omitted when they occur:

   - at the end of a word;

- before another consonant.

For example, *po-me* = *poimen* (ποιμήν) "shepherd"; *ka-ko* = *khalkos* (χαλκός) "bronze"; *pa-te* = *pater* (πατήρ) "father".

8. Initial *s-* is generally omitted before another consonant.

9. In consonant clusters involving a stop + *w*, both consonants are represented, with an intervening vowel that is either taken from the following syllable or supplied as a default. However, *r* preceding *w* is typically omitted.

10. Stop consonants (*d, k, p, q, t*) occurring before another consonant are typically written with the vowel of the following syllable (less often the preceding syllable). For example:

    - *ku-ru-so* = *khrusos* (χρυσός) "gold";
    - *A-mi-ni-so* = *Amnisos* (Ἀμνισός).

    Special orthographic solutions are used to represent final consonant clusters, as in:

    - *wa-na-ka* = *wanax* (ϝάναξ) "king".

These conventions reveal the extent to which the Linear B script adapted to the phonology of Mycenaean Greek, despite being constrained by a syllabic system originally designed for a different language. [10]

The decipherment of Linear B was finally confirmed one year later, when in 1953 a new tablet found in Pylos could be translated only by using Ventris' grid. This independent verification of the decipherment on material not previously available to Ventris provided undeniable evidence of his success. [11]

The work of Ventris and Chadwick, by combining structural analysis with comparative linguistics, thus not only unlocked the meaning of Linear B but also illuminated the phonological landscape of the earliest attested form of the Greek language.

Chapter 3

The cognate matching task

As explained in Section 2.4, the decipherment of Linear B required a collective effort that involved several scholars and extended over multiple decades.

The main difficulties in any decipherment process concerning ancient languages arise from two factors: the absence of parallel texts that could guide the interpretation, and the extreme scarcity of surviving documents. Even for domain experts, decipherment therefore demands encyclopedic linguistic and historical knowledge, combined with an enormous amount of manual work that is often prohibitive in terms of time and resources. Moreover, the challenges encountered during the decipherment of one language are rarely reusable for others, since much of the required work and insights are closely tied to the specific features of that language and cannot be easily generalized [12]. For example, the rules mentioned at the end of Section 2.4.2 are highly specific to the context of Linear B and its unique characteristics, and do not even apply to closely related scripts, such as Cypriot or possibly Linear A (for which no certainty exists, as the language remains undeciphered).

For these reasons, the introduction of computational methods and machine learning techniques has the potential to provide crucial assistance. Nevertheless, the central obstacle remains the limited amount of training data. This scarcity of examples makes it essential to design models that are able to learn effectively from small datasets and to generalize from minimal evidence.

In this respect, it is important to note that not all machine learning architectures are equally suitable for such low-resource conditions. Transformers, for example, are a class of neural networks that have achieved remarkable results in modern natural language processing tasks, such as machine translation and text generation. However, their success is strongly dependent on access to very large training corpora. In the absence of such data, as is the case for ancient scripts, the application of transformers becomes impractical. This explains why alternative architectures, which are better suited to scenarios where training data is scarce, are currently preferred in the computational study of ancient languages.

3.1 Identifying Cognates

Cognates are words in different languages that share a common etymological origin. Identifying cognates is a crucial task in historical linguistics, as it allows researchers

to trace the evolution of languages and to understand their relationships. In the case of Linear B, the identification of cognates provided valuable correspondences between the syllabic script and the phonetic representations of Ancient Greek, which proved instrumental in the decipherment process. Cognate matching was indeed a key aspect of Ventris' work, as he relied on clearly identifiable cognates to assign phonetic values to Linear B signs. However, the task of identifying cognates is not always straightforward, since it requires a deep understanding of the historical and linguistic context of the languages involved. In addition, several phonological transformations had taken place over time, including the differentiation of aspirated consonants and the loss of certain sounds once present in Linear B, such as the digamma (ϝ) and the labio-velar consonant *q*.

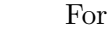
Below are some examples of cognates identified between Linear B and Ancient Greek:

- 𐀀𐀍𐀪𐀵 (a-e-ti-to) → ἀέθιστος, ἐθίζω
  This example shows how a Linear B sequence corresponds directly to a recognizable Greek adjective and verb.
- 𐀀𐀨𐀜𐀗 (a-di-nwa-ta) → Ἀδινϝᾶτας
  An instance where the Linear B syllabogram "nwa" reflects the presence of the digamma.
- 𐀁𐀅𐀴 (e-ma-ha) → Ἑρμᾶς, Ἑρμαῖ, Ἑρμαίας
  Illustrates a clear lexical link to Greek terms related to Hermes.
- 𐀏𐀜𐀲 (ko-no-so) → Κνωσός
  A straightforward toponym, the name of the main Minoan palace site.
- 𐀷𐀜𐀲𐀎𐀇 (wo-no-qe-wa) → Φονοκέϝας, Φονοκέϝαν
  Preserves traces of the digamma and labio-velar consonants.
- 𐀷𐀖𐀬𐀩𐀖𐀩 (wo-ro-ki-jo-ne-jo) → ϝοργιονείον, Ὀργιονεῖος, Ὀργίωνες
  A more complex example, relating to anthroponyms and associations.
- 𐀔𐀋𐀆𐀁 (qi-si-pe-e) → ξίφεις, ξίφη, ξίφεα
  Reflects the Linear B representation of the Greek word for "sword."
- 𐀕𐀈𐀏𐀍𐀖 (re-u-ko-to-ro) → Λεῦκτρον, Λεῦκτροι
  A toponym, attested both in Linear B and later Greek sources.
- 𐀟𐀈𐀟𐀗 (qo-u-qo-ta) → Βουβώτας
  Demonstrates the phonetic evolution of labio-velar sounds.
- 𐀎𐀔𐀜𐀡𐀙 (qe-qi-no-me-na) → γεγινώμενα, γιγνόμενα
  A verbal form that shows continuity in Greek morphology.
- 𐀓𐀖𐀃𐀍𐀩 (po-ro-wi-to-jo) → πλωϝιστοῖο
  Example of a genitive form with preservation of the digamma.
- 𐀓𐀪𐀬 (po-ti-pi) → Πόρτις, Πόρτιφι
  Reflects a proper name with different Greek attestations.

-  (a-ri-qa) → Ἄρισβας
  Example of an anthroponym preserving older consonantal values.
-  (ko-no) → Σκοῖνος
  Shows the addition of a consonant in Ancient Greek.

3.2 Datasets

In this section, the process of gathering and preparing the datasets used in this study is described. It is worth noting that I made several choices in order to create a clean and consistent dataset. The main choices are the following:

- All diacritics, accents, breathings, and iota subscripts were removed from Ancient Greek forms, resulting in a simplified text. For example ἀέθιστος is represented as αεθιστος.
- All instances of uppercase letters were converted to lowercase in order to normalize the data. For instance Κνωσός is represented as κνωσος. Note also that Linear B does not distinguish uppercase and lowercase.
- All instances of punctuation were removed.
- Linear B words were represented using their Latinized transcription. Therefore, the dataset contains the word as ko-wo instead of the original Linear B characters.
- The instances of digamma were inserted in a suitable position also in the Greek form, despite their disappearance from Classical Greek. For example, the word κόρος is represented as κορϝος in the dataset. In some cases, the version without digamma is also included, as in κουρος.
- The only additional symbol employed, besides the standard Greek alphabet, is "h", used to represent the aspiration conveyed by the syllabogram  (ha). For instance,  (a-pi-ha-ro) is rendered as αμφιηαλος.

The sources of all the data included in the final version of the dataset are the following:

- **Luo's dataset** [12]: a collection of cognates between Linear B and Ancient Greek, compiled by Jiaming Luo. It contains 919 Linear B words together with their proposed Ancient Greek correspondences.
- **Chris Tselentis' *Linear B Lexicon*** [17]: a lexicon comprising 1338 Linear B entries with Ancient Greek equivalents. A substantial portion of these entries overlap with Luo's dataset.
- **Ventris and Chadwick's Vocabulary** [18]: a digitized compilation based on the original notes and lexical work of Michael Ventris and John Chadwick,

later supplemented with commentary by other scholars. The resource, available as a CSV file at `https://linear-b.kinezika.com/lexicon.html`, comprises 2747 unique words. It is organized as a vocabulary, offering definitions and interpretative remarks on the terms, and thus represents an extended digital derivative of their foundational work.

3.2.1 Prompt engineering

Some tasks were automated using prompt engineering techniques with Gemini 2.0 Flash and Gemini 2.5 Flash, which proved effective for text processing and data manipulation. Whenever these techniques are employed, I explicitly indicate their use and summarize the prompt instructions that were critical to the task. In general, the prompts followed these principles:

- **Clarity and specificity:** Clear, unambiguous instructions to reduce variance and align outputs with task requirements [6].
- **Iterative refinement:** Prompts were refined based on model outputs to improve quality across iterations.
- **Contextualization:** Task-relevant context (e.g., field definitions, examples) was included to guide disambiguation.
- **Structured reasoning:** Prompts encouraged stepwise reasoning for complex tasks (e.g., breaking a problem into sub-steps), leading to more coherent outputs [19].
- **Structured formatting:** Outputs were requested in explicit schemas (XML/JSON, bullet/numbered lists) to ensure machine-readable, post-processable results.
- **Salience cues (including UPPERCASE for emphasis):** Key requirements were emphasized to prioritize the most important aspects and reduce omission errors [5].

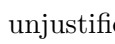
When necessary, I adjusted the model's generation settings to favor determinism: the `temperature` was set low (e.g., 0.1–0.3) to reduce randomness and increase repeatability, and `top-k` was fixed at 1 (greedy selection).

3.2.2 Luo's Dataset

Luo's dataset is the one on which the creator of the NeuroDecipher model (introduced later) tested its performance. The model achieves excellent results on this dataset, reaching an accuracy of 84.7% of cognates correctly matched [12].

However, the dataset is not without limitations. The main issue I identified upon closer inspection is that some proposed Greek cognates are not attested in extant Ancient Greek sources, but rather tentative transliterations of Linear B forms or artificially modified versions of Greek correspondences. While this may artificially improve the cognate matching task, it does not reflect a realistic linguistic scenario

and makes the dataset unsuitable for use in an automatic translation pipeline. A few illustrative examples are given below:

-  (qo-o) is correctly associated with βους, as also confirmed by Tselentis' lexicon [17]. However, the dataset additionally lists κfoος, which does not correspond to any attested Ancient Greek form.
-  (to-no) is linked to θρονος, likewise confirmed by Tselentis' lexicon [17], but the dataset also includes θορνος, an unattested form that results from an unjustified inversion of letters. This error extends to  (to-ro-no-wo-ko), where the listed cognate is θορνοfοργος. I corrected this instead to θρονοfοργος and θρονοfεργος, both plausible formations obtained by combining θρόνος ("throne, chair") with the productive suffix derived from ἔργον ("work, deed"). The resulting compound restores the original sense of the term as "chair-maker" or "craftsman of thrones."
- In several cases, the dataset conflates distinct gendered forms by grouping them under a single entry. For example,  (ne-wa), corresponding to νεfα or νεα, feminine, and (ne-wo), corresponding to νεfος or νεος, masculine, were all grouped under , despite both variants being independently attested in the Linear B corpus.

The adjustments described above were applied to Luo's dataset in order to enhance its reliability and linguistic accuracy. This revised version was then used both to measure the performance of the NeuroDecipher model on a more realistic dataset and as the foundation for constructing the final comprehensive dataset, which also integrates additional material from Tselentis' Linear B Lexicon and from the digitized vocabulary of Ventris and Chadwick. The resulting revised dataset comprises 976 Linear B entries paired with their respective Ancient Greek correspondences.

3.2.3 Tselentis' Dataset

Tselentis' dataset represents a valuable resource for the study of Linear B, as it comprises a comprehensive lexicon of Linear B terms and their corresponding Ancient Greek forms. It serves as a crucial reference point for validating and enriching the cognate pairs identified in Luo's dataset.

The main drawback of Tselentis' lexicon is that it was only available as a PDF document, which made it necessary to manually transcribe the entries into a more usable format. After using an online OCR tool to extract its content into a CSV file, followed by targeted cleaning, a structured file containing all the fields in the lexicon was obtained.

Nevertheless, the data still required further processing to extract the Greek and Linear B forms in accordance with the normalization criteria outlined at the start of this section. Several parsing mistakes, along with inconsistencies in accents, diacritics, and formatting, had to be corrected to ensure accuracy and consistency. To streamline this process, Prompt Engineering techniques were employed with Gemini 2.0 Flash, guided by explicit processing directives.

These techniques enabled the automated correction of recurrent errors and inconsistencies, significantly accelerating data preparation. The processed dataset was then reviewed manually to ensure its quality and reliability before integration into the final comprehensive dataset.

For transparency and repeatability, I detail here the precise directives provided to Gemini for dataset processing.

PROCESSING RULES FOR LINEAR B TO GREEK COGNATES:

0. CRITICAL: DO NOT MAKE ANY MODIFICATION TO GREEK COGNATES OR LINEAR B SEQUENCES IF THE MODIFICATION IS NOT MENTIONED IN THE FOLLOWING RULES! DO NOT CHANGE THE INPUT IN ANY POSSIBLE WAY AND ONLY APPLY THE GIVEN MODIFICATION RULES! NO FANTASY JUST BLINDLY OBEY!

1. SPLITTING MULTIPLE WORDS: When Linear B field contains multiple words separated by "/", create separate JSON objects for each word, matching with the corresponding Greek cognate in the same position.

2. HANDLING PARENTHESES: For Linear B words with parenthetical elements like "po-ni-ke-(j)a", create two separate entries (one with and one without the parenthetical element, like "po-ni-ke-ja" and "po-ni-ke-a").

2.1. HANDLING PARENTHESES FOR GREEK COGNATE: if a word is presented with optional greek characters with parenthetical elements like "αιξμά(ν)ς", include both variants with and without the letter in parentheses.

2.2. HANDLING PARENTHESES FOR GREEK COGNATE: if a word is presented within parentheses, include it regardless, like "Αιθαλεύσι(Αιθαλεύς)".

3. MULTIPLE TRANSLATIONS: If a Linear B word has multiple possible Greek cognates, include all of them as an array within the same JSON object.

4. REMOVING DIACRITICS: Remove all accents, breathing marks, and other diacritics from Greek cognates.

5. HANDLING "ha" SIGN: When "ha" appears in Linear B, ensure the corresponding Greek cognate includes "h".

6. DIGAMMA CONVERSION: Convert every instance of digamma "F" to lowercase "f" in Greek cognates.

7. CRITICAL: ALLOWED CHARACTERS: Use ONLY these characters in Greek cognates: fhαβγδεζηθικλμνξοπρσστυφχψω. DO NOT USE ANY OTHER CHARACTER FOR ANY REASON!

8. DISALLOWED CHARACTERS: Drop cognates containing disallowed characters, but preserve valid cognates found within parentheses or other markers.

9. IN SOME VERY RARE AND PARTICULAR CASES some cognates may be considered as DUBIOUS, IF AND ONLY IF THEY CONTAIN A LIKELY WRONG TRANSLITERATION AND A CORRECT MATCH IS ALREADY PRESENT. Put them in the "dubious" field, another optional array field in the JSON object.

10. if white spaces are present between syllables separated by - in the linear b sequence, remove them.

11. DO NOT USE PARENTHESES IN THE LINEAR B SEQUENCES OR IN THE GREEK COGNATE.

Applying these directives reduced OCR noise and errors, preserving valid cognate pairs while preventing format drift that would hinder downstream parsing. These directives, together with a number of examples and some input and output definitions, allowed me to automate most of the manual work that the data needed in order to be ready to use.

3.2.4 Brute-Force Cognate Extraction

To enlarge the dataset, I implemented and applied a brute-force, syllabogram-aware matcher over a large Greek lexicon (composed by the Iliad and Odyssey). Greek forms were first normalized (diacritics removed, lowercased) and then latinized via a direct character map aligned to Linear B conventions (e.g., $\xi \rightarrow ks$); special cases included the digamma (ϝ with later re-insertion of f/h during reconstruction) and the labio-velar q (permitted to align with $\{q, p, k\}$).

Matching logic. Each Linear B form is tokenized into syllabograms and compared against every latinized Greek word by scanning both sequences left-to-right and greedily aligning syllabograms to characters. The matcher handles three syllabogram classes, with tailored rules:

- **V (length 1):** align the same vowel; mismatches advance the Greek pointer (counted as a skip).
- **CV (length 2):** align the initial consonant and then the vowel; special handling covers clusters such as $k{+}s$ that straddle the next syllable (e.g., *ks*).
- **Specific triads (length 3):** a small set of syllabograms (e.g., *phu*) is matched as a fixed mini-pattern.

State tracked during scanning. The algorithm maintains (i) the total number of skipped Greek characters and the maximum consecutive skip streak (to avoid over-skipping), (ii) a flag for illegal syllable mappings, and (iii) a small relaxation allowing a single liquid glide (e.g., an isolated “r”).

Acceptance gates. After the pass, a candidate pair is accepted only if all of the following hold:

(i) near-complete coverage of the Linear B syllables, i.e. the pointer must reach the last or penultimate syllable;

(ii) the Greek pointer must lie within three characters of the word's end;

(iii) the total number of skips must be fewer than four and no more than two consecutive characters may be skipped;

(iv) an initial-character compatibility heuristic holds (to avoid wild misalignments);

(v) no illegal syllable mapping was flagged;

(vi) for short Linear B words, stricter thresholds are applied (fewer allowed skips and a smaller tail).

Outputs and design choice. For each accepted pair, the algorithm records a coverage score,

$$\text{coverage} = \frac{\text{matched syllables}}{\text{total syllables}},$$

and reconstructs the Greek surface form by re-inserting any f/h positions suppressed during normalization. The overarching design targets recall over precision: collect as many plausible pairs as possible, even at the cost of some spurious matches, to be filtered later.

```
def match(lin_b_words, greek_words):
    for lb_word in lin_b_words.keys():  # scan each Linear B entry
        lb_syllables = lb_word.split("-") # LB sequence to syllables
        for gr_word in greek_words.keys(): # scan each Greek candidate

            gr_chars = list(gr_word) # Greek word to char list
            i_syl = 0 # index over LB syllables
            j_chr = 0 # index over Greek chars
            skip_count = 0 # total skipped Greek chars
            skip_streak = 0 # current consecutive skips
            max_skip_streak = 0 # max consecutive skips seen
            invalid_syllable = False # flag for invalid syllable mapping
            skipped_syllables = [] # track skipped syllables if needed

            while i_syl < len(lb_syllables) and j_chr < len(gr_chars):
                lb_syllable = lb_syllables[i_syl]
                gr_char = gr_chars[j_chr]

                if len(lb_syllable) == 1:
                    if gr_char == lb_syllable:
                        skip_streak = 0
                        i_syl += 1
                        j_chr += 1
                    else:
                        skip_count += 1
                        skip_streak += 1
                        if skip_streak > max_skip_streak:
```

```
                            max_skip_streak = skip_streak
                        j_chr += 1

                elif len(lb_syllable) == 2:
                    cons = lb_syllable[0]

                    if cons == "k":
                        has_room = (j_chr + 2) < len(gr_chars)
                        tail = gr_chars[j_chr + 1 : j_chr + 3]
                        # checks double guttural
                        has_dg = has_room and ("".join(tail) ==
            ↪ lb_syllable)

                        if gr_char == "k" and not has_dg:
                            skip_streak = 0

                            has_next_char = (j_chr + 1) < len(gr_chars)
                            # checks ks
                            next_is_s = has_next_char and (gr_chars[j_chr
            ↪  + 1] == "s")
                            has_next_syl = i_syl < (len(lb_syllables) -
            ↪ 1)

                            if next_is_s and has_next_syl:  # matched ks
                                next_syl = lb_syllables[i_syl + 1]
                                if next_syl[0] == "s":
                                    i_syl += 2
                                    j_chr += 2
                                    vowel_ok = (
                                        j_chr < len(gr_chars)
                                        and next_syl[1] == gr_chars[j_chr
            ↪ ]
                                    )
                                    if vowel_ok:
                                        j_chr += 1
                                else:
                                    i_syl += 1
                                    j_chr += 1
                                    vowel_ok = (
                                        j_chr < len(gr_chars)
                                        and lb_syllable[1] == gr_chars[
            ↪ j_chr]
                                    )
                                    if vowel_ok:
                                        j_chr += 1
                            else:
                                i_syl += 1
                                j_chr += 1
                                vowel_ok = (
                                    j_chr < len(gr_chars)
                                    and lb_syllable[1] == gr_chars[j_chr]
                                )
                                if vowel_ok:
                                    j_chr += 1
                        else:
                            j_chr += 1
                            skip_count += 1
```

```
                            skip_streak += 1
                            if skip_streak > max_skip_streak:
                                max_skip_streak = skip_streak

                    if cons == "q":  # labio-velar mapped to {q,p,k}
                        is_labio_map = gr_char in ("q", "p", "k")
                        if is_labio_map:
                            skip_streak = 0
                            i_syl += 1
                            j_chr += 1
                            vowel_ok = (
                                j_chr < len(gr_chars)
                                and lb_syllable[1] == gr_chars[j_chr]
                            )
                            if vowel_ok:
                                j_chr += 1
                        else:
                            j_chr += 1
                            skip_count += 1
                            skip_streak += 1
                            if skip_streak > max_skip_streak:
                                max_skip_streak = skip_streak
                    ...
                if len(lb_syllable) == 3:
                    if lb_syllable == "phu":
                        has_u = ((j_chr + 1) < len(gr_chars)) and (
    ↪ gr_chars[j_chr + 1] == "u")
                        if gr_char == "p" and has_u:
                            skip_streak = 0
                            i_syl += 1
                            j_chr += 2
                        else:
                            j_chr += 1
                            skip_count += 1
                            skip_streak += 1
                            if skip_streak > max_skip_streak:
                                max_skip_streak = skip_streak
                    ...

            # allow single liquid glide
            if max_skip_streak == 2 and len(skipped_syllables) == 1:
                if skipped_syllables[0][0] == "r":
                    max_skip_streak = 0

            # --- Acceptance constraints ---
            start_ok = (
                lb_word[0] == gr_word[0]
                or (lb_word[0] == "p" and gr_chars[0] == "p")
                ...
            )
            lb_aligned = i_syl >= (len(lb_syllables) - 1)
            gr_near_end = j_chr >= (len(gr_word) - 3)
            skips_ok = skip_count < 4
            streak_ok = max_skip_streak <= 2
            mapping_ok = not invalid_syllable

```

```
            conds1 = (lb_aligned, gr_near_end, skips_ok, start_ok,
             ↪ streak_ok, mapping_ok)
            gate1 = all(conds1)

            short_lb = len(lb_syllables) <= 3
            near_end_tight = j_chr >= (len(gr_word) - 2)
            short_ok = (skip_count <= 2) and near_end_tight and (
             ↪ max_skip_streak < 2)
            # conditions for shorter LB words
            gate2 = (short_lb and short_ok) or (not short_lb)

            if gate1 and gate2:
                # Reconstruct normalized Greek, re-inserting 'f'/'h'
                greek_norm_chars = list(greek_words[gr_word])
                dropped = len(greek_words[gr_word]) < len(gr_chars)
                if dropped:
                    fh = ("f", "h")
                    positions = [p for p, c in enumerate(gr_chars) if c
             ↪ in fh]
                    for off, pos in enumerate(positions):
                        greek_norm_chars.insert(pos + off, gr_chars[pos])

                coverage = i_syl / len(lb_syllables)
                match_pair = ("".join(greek_norm_chars), coverage)
                lin_b_words[lb_word]["cognates"].append(match_pair)

    return lin_b_words
```

Listing 3.1. Brute-Force matching algorithm

The brute-force search has complexity $O(N_{\text{LB}} \cdot N_{\text{GR}} \cdot L)$, where N_{LB} is the number of Linear B forms, N_{GR} is the number of Greek forms, and L is the maximum length of words. For the translation dataset, $N_{\text{LB}} = 4809$ and $N_{\text{GR}} = 18646$, with $L \leq 19$, as the longest word is 𐀀𐀁𐀂𐀃𐀄𐀅𐀆𐀇𐀈𐀉𐀊𐀋 (a-ni-ja-e-e-ro-pa-jo-qe-ro-sa).

Outputs refinement. Clearly, this brute-force approach is not perfect and, while it attempts to filter out very different words as much as possible, it inevitably returns more pairs than true cognates. This limitation is acceptable because a second filtering stage prunes and re-ranks candidates by means of prompt engineering. The filter is driven by a structured XML prompt and produces a strict JSON response. The main instructions given to Gemini 2.5 Flash can be summarized as follows:

- **Invocation of Luo's principles**: the model must evaluate each proposal against the four cognate matching principles, which will be detailed in the next section, ensuring that they are applied consistently and rigorously. These principles are distributional similarity of matching characters, monotonic character mapping within cognates, structural sparsity of cognate mapping, and significant cognate overlap within related languages.
- **Character policy**: enforce the restricted output alphabet for Ancient Greek forms introduced in Section 3.2, disallowing accents, breathings, or subscript iota.

- **Phonological mapping tables**: require adherence to explicit Linear B → Ancient Greek correspondence tables for consonants and vowels, supported by contextual notes (e.g. treatment of labiovelars, liquids, or clusters).
- **Independence from hints**: all provided "proposed cognates" are treated as non-binding; the model must search beyond them and consider the broader Ancient Greek corpus.
- **Plausibility assessment**: evaluate candidates against a rubric of evaluation criteria, aligned with the four principles, and reject items that fail any check.
- **Reasoning scaffold**: follow an explicit five-step chain-of-thought structure to ensure consistency in the decision path: syllabogram analysis → monotonicity enforcement → sparsity audit → pattern comparison → composite ranking.
- **Calibration of likelihoods**: apply a calibrated numerical scale with illustrative examples ranging from well-attested to speculative cases, so that likelihood scores are interpretable and comparable across words.
- **Automatic downweighting and hard caps**: apply four penalties, for example, -0.3 for three or more non-trivial transformations or any reordering; -0.2 for rarity, conflict with scholarship, or semantic stretch, and enforce global caps: likelihood < 0.7 whenever the Linear B sequence contains unknown syllabograms such as *19, and < 0.85 for novel, unattested proposals even if other checks are passed.
- **Worked examples**: provide a wide set of input-output examples illustrating common patterns (such as $q{+}s$, digamma insertion, or suffixal transformations) to calibrate the model's application of rules and character policy.
- **Quality-control gate**: prefer fewer, higher-quality outputs over speculative additions and abstain when uncertain.
- **Instance metadata**: input word block: Linear B form, completeness level, optional definition from Chadwick and Ventris' vocabulary [18], any pre-proposed cognates with their brute-force matching scores, and entity type, to give context without constraining the final choice.

The complete XML prompt and the exact JSON schema are presented in the source code repository. After this automated filtering stage, the dataset was manually reviewed to ensure the quality and reliability of the final cognate pairs. Additionally, some cognates pairs were added from Ventris and Chadwick's vocabulary [18] when they were missing from the previous datasets.

3.3 Cognate Matching Model

In this section, the architecture and training procedure of the cognate matching model created by Jiaming Luo [12] are described. The model is based on a sequence-to-sequence (seq2seq) architecture with attention mechanisms, which has

been shown to be effective for various natural language processing tasks, including machine translation and text generation. The model is trained to take a Linear B word as input and generate its corresponding Ancient Greek cognate as output. As explained in Luo's paper, the main challenge of the decipherment task is the lack of a strong supervision signal that guides standard machine translation algorithms. To address this challenge, he proposed an architecture that learns patterns of language transformation. Moreover, the model is designed to be language-agnostic, meaning that it can be applied to other decipherment tasks beyond Linear B and Ancient Greek. In order to achieve this, the model relies on a set of principles that capture the general characteristics of cognate relationships between languages.

3.3.1 Cognate Matching Principles

The four principles that guide the model's architecture and training are the following:

1. **Distributional similarity of matching characters:** matching characters are expected to appear in similar places in corresponding cognates, therefore their local context should match as well.
2. **Monotonic character mapping within cognates:** cognates are expected to exhibit largely monotonic alignments, as character reorderings are rare.
3. **Structural sparsity of cognate mapping:** cognate matchings are expected to be sparse and near one-to-one between segments derived from the same proto-origin.
4. **Significant cognate overlap within related languages:** it is assumed that the derived vocabulary will provide sufficient coverage for recovering lost cognates.

3.3.2 The Generative Framework

The model architecture relies on a latent variable $\mathcal{F} = \{f_{ij}\}$, representing the word-level alignment between the words in the lost language $\mathcal{X} = \{x_i\}$ and the known language $\mathcal{Y} = \{y_j\}$. The following joint probability is derived:

$$\mathbf{P}(\mathcal{X}, \mathcal{Y}) = \sum_{\mathcal{F} \in \mathbb{F}} \mathbf{P}(\mathcal{F})\, \mathbf{P}(\mathcal{X} \mid \mathcal{F})\, \mathbf{P}(\mathcal{Y} \mid \mathcal{F}, \mathcal{X}) \propto \sum_{\mathcal{F} \in \mathbb{F}} \mathbf{P}(\mathcal{Y} \mid \mathcal{X}, \mathcal{F}) = \sum_{\mathcal{F} \in \mathbb{F}} \prod_{y_j \in \mathcal{Y}} \mathbf{P}(y_j \mid \mathcal{X}, \mathcal{F}),$$

where a uniform prior is assumed for $\mathbf{P}(\mathcal{F})$ and $\mathbf{P}(\mathcal{X} \mid \mathcal{F})$, and independence and identical distribution (i.i.d.) are assumed across $y_j \in \mathcal{Y}$. Here, $\mathbb{F}$ denotes the set of all valid values of the latent variable $\mathcal{F}$.

The conditional probability $\mathbf{P}(y_j \mid \mathcal{X}, \mathcal{F})$ is further decomposed as:

$$\mathbf{P}(y_j \mid \mathcal{X}, \mathcal{F}) = \sum_{x_i \in \mathcal{X}} f_{ij} \cdot \mathbf{P}_\theta(y_j \mid x_i),$$

where $\mathbf{P}_\theta(y_j \mid x_i)$ is a neural seq2seq model with parameters θ that generates y_j given x_i. In this framework, the latent variable $\mathcal{F}$ captures the alignment between

the two languages, incorporating the global constraints described by Properties 3 and 4, while the neural model learns to generate cognates based on character-level constraints defined in Properties 1 and 2. However, directly optimizing the joint probability is infeasible, as it requires summing over all possible values of $\mathcal{F}$. To overcome this issue, Luo adopted an Expectation-Maximization (EM) iterative training algorithm: the neural model is trained in the M-step to maximize the likelihood of $\prod_{y_j \in \mathcal{Y}} \mathbf{P}(y_j \mid \mathcal{X}, \mathcal{F})$ given fixed $\mathcal{F}$, while in the E-step $\mathcal{F}$ is estimated via a minimum-cost flow problem defined over the trained neural network.

3.3.3 NeuroDecipher Model

The neural model $\mathbf{P}_\theta(y_j \mid x_i)$, named NeuroDecipher, is based on a standard seq2seq architecture. The three main components of the model are the encoder, the decoder, and the attention mechanism.

Encoder. The encoder consists of a stack of two embedding layers and a bidirectional LSTM. This design enforces the requirement that character embeddings of the two languages reside in the same space, the Universal Character Embedding Space. This is achieved by using a universal embedding matrix $U \in \mathbb{R}^{\mathcal{U} \times E}$, where $\mathcal{U}$ is the dimensionality of the universal embedding space and E is the character embedding size used by the model. The second embedding layer is a lost language character weight matrix $W_x \in \mathbb{R}^{V_b \times \mathcal{U}}$, where V_b is the vocabulary size of the lost language. Therefore, the final embedding matrix for the lost language is $E_x = W_x U$. The bidirectional LSTM has hidden size H and produces a sequence of hidden and cell states (h_l, c_l) for $l = 1, \ldots, N$, where N is the number of layers of the LSTM.

Thus, the encoder input is a batch of Linear B sequences with shape $\mathbb{R}^{B \times L \times V_b}$, where B is the batch size, and L is the (maximum) sequence length. The encoder output has shape $\mathbb{R}^{B \times L \times 2H}$, where H is the hidden size of the LSTM and $2H$ accounts for the concatenation of forward and backward hidden states. Moreover, the hidden and cell states from all layers, each of shape $\mathbb{R}^{2N \times B \times H}$ (for N LSTM layers in each direction), are averaged along the first dimension and used to initialize the hidden and cell states of each layer of the decoder.

One of my experiments involved initializing the lost language LSTM's cell state with a projection of the FastText embeddings of the words in the batch concatenated. The motivation was that injecting additional contextual information might help the model capture relationships between Linear B and Ancient Greek more effectively. Empirically, this initialization stabilized training (more reliable convergence) but reduced final performance (see Section 3.3.7). A likely cause is representational mismatch: FastText captures word-level semantics, but the task needs character-level mappings; this biases the model toward irrelevant signals, which confuses learning and hurts generalization.

Decoder and Global Attention. Decoding proceeds by iterative next-character prediction from a fixed start token. The decoder is a multi-layer Custom LSTM. For the first LSTM layer at step t, the input is the concatenation of the context vector $\tilde{h}_{t-1}$ and the embedding of the previously generated character e_{t-1}; for higher layers, the input is the hidden state of the preceding layer. The last LSTM layer

produces the decoder state $c_t \in \mathbb{R}^{B\times H}$, which feeds a Global Attention module that integrates encoder and decoder's information.

Let the encoder outputs be $o_s(1), \ldots, o_s(L) \in \mathbb{R}^{B\times 2H}$. A learnable matrix $W_a \in \mathbb{R}^{2H\times H}$ linearly projects encoder states into the decoder space, and attention scores are computed by a scalar product with c_t:

$$s_t(k) \;=\; \big(W_a\, o_s(k)\big)\cdot c_t, \qquad \alpha_t \;=\; \mathrm{softmax}\big(s_t(1{:}L)\big) \in \mathbb{R}^{B\times L}.$$

The attention weights summarize encoder information in two complementary ways: a content summary in the hidden-state space,

$$c_s \;=\; \sum_{k=1}^{L} \alpha_t(k)\, o_s(k) \;\in\; \mathbb{R}^{B\times 2H},$$

and a content summary directly in the embedding space,

$$r \;=\; \sum_{k=1}^{L} \alpha_t(k)\, e_x(k) \;\in\; \mathbb{R}^{B\times E},$$

where $e_x(k)$ is the lost-language input embedding of the k-th encoder character.

The context vector $\tilde{h}_t$ fuses what the decoder is currently trying to produce (c_t) with what the encoder deems most relevant (c_s), and a residual anchor r in the input embedding space. Concretely, we concatenate c_s and c_t, map back to the model's embedding space with $W_o \in \mathbb{R}^{3H\times E}$, and add the residual:

$$\tilde{h}_t \;=\; W_o\,[\,c_s;\, c_t\,] \;+\; \rho\, r \;\in\; \mathbb{R}^{B\times E}.$$

The residual term r serves two purposes: (a) it stabilizes learning by providing a direct pathway from input embeddings to the decoder (improving gradient flow and helping early decoding steps), and (b) it regularizes the attention fusion by anchoring $\tilde{h}_t$ to the same representation space as the inputs, which reduces drift and improves alignment consistency. The scalar ρ controls the contribution of this residual connection.

The vector $\tilde{h}_t$ is then projected to the Ancient Greek vocabulary via

$$\mathrm{logits}_t \;=\; \tilde{h}_t\, E_y^{\top} \;\in\; \mathbb{R}^{B\times V_g}, \qquad E_y \;=\; W_y\, U \;\in\; \mathbb{R}^{V_g\times E},$$

followed by a softmax: $p_t = \mathrm{softmax}(\mathrm{logits}_t)$. The predicted character is $\hat{y}_t = \arg\max p_t$. For the next step, the "previous-token" embedding is taken as the expected embedding under p_t,

$$e_t \;=\; p_t\, E_y \;\in\; \mathbb{R}^{B\times E},$$

and the process repeats for a number of steps corresponding to the length of the longest Ancient Greek word in the corpus. This design makes the context vector $\tilde{h}_t$ the central carrier of both encoder-side evidence and decoder-side character-level information, while the residual pathway ensures robust, stable decoding grounded in the input embedding space. The overall model architecture is illustrated in Figure 3.1.

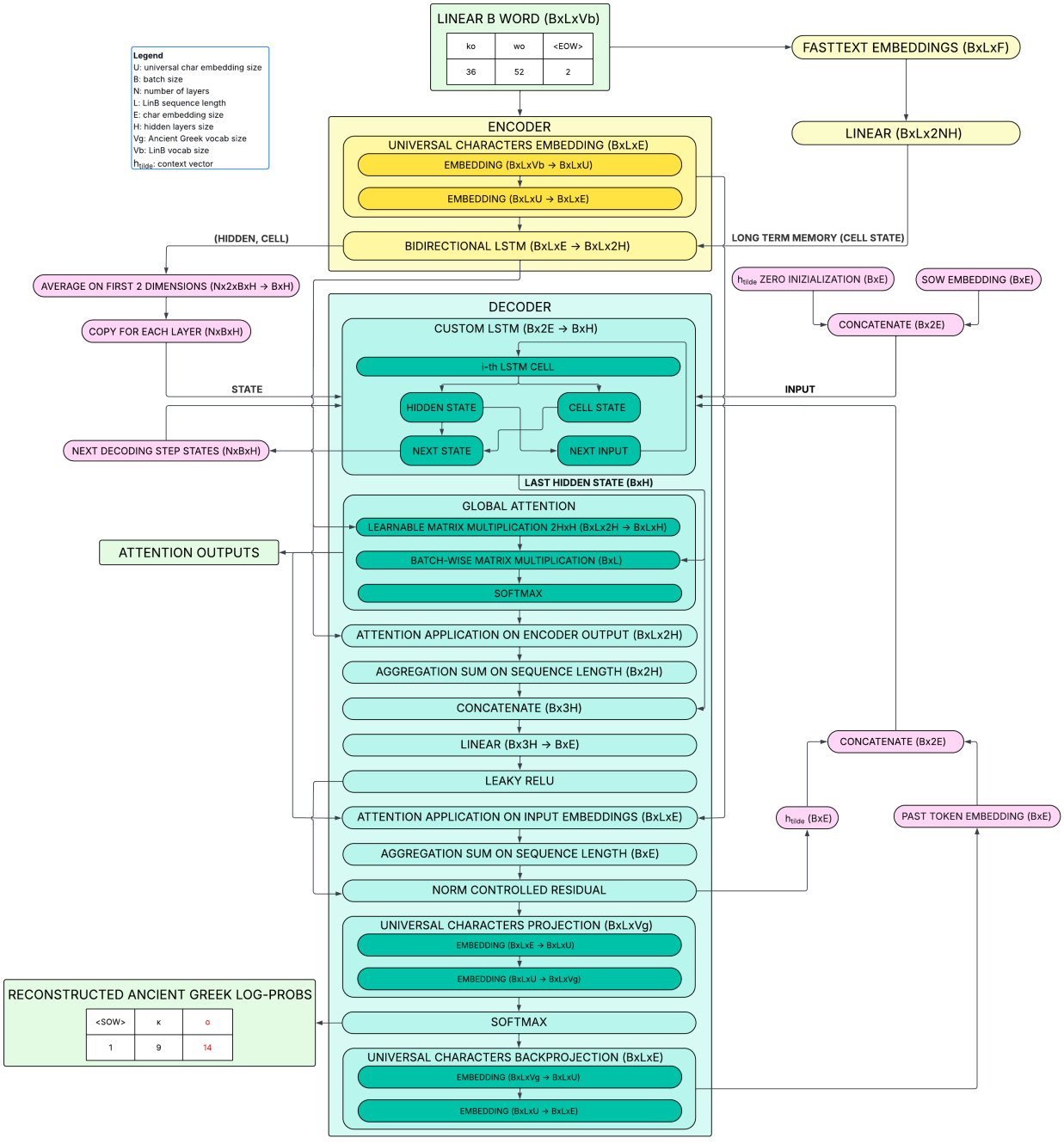


Figure 3.1. Graphical representation of the model architecture.

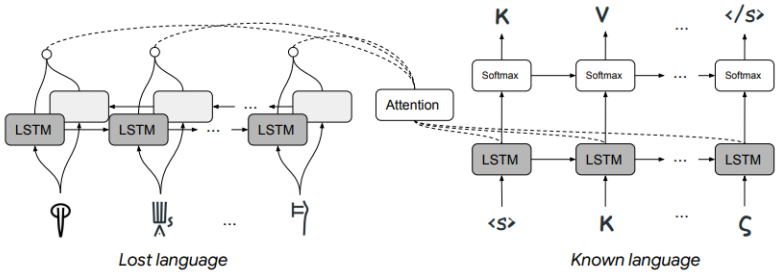


Figure 3.2. Overview of the model architecture.[1]

3.3.4 Output Refinement

The model supports three output modes. The first is a corpus-constrained maximum-likelihood selection (MLE) based on the decoder's character distributions. Given the decoder outputs $\{p_t\}$ for $t = 1, \ldots, T_{\max}$, where $p_t \in \mathbb{R}^{V_g}$ is a probability vector over the Greek vocabulary at step t, each candidate word $y_j = (y_{j,1}, \ldots, y_{j,T_w})$ in the lexicon is scored by the product of the probabilities of its characters:

$$\mathbf{P}_\theta(y_j \mid x_i) = \prod_{t=1}^{T_w} p_t[y_{j,t}]\,,$$

Equivalently, if $E(y_j) \in \{0,1\}^{T_w \times V_g}$ is the one-hot matrix of y_j (row t is one-hot for $y_{j,t}$), indicating with $E_t(y_j)$ the t-th row, then

$$\log \mathbf{P}_\theta(y_j \mid x_i) = \sum_{t=1}^{T_w} E_t(y_j)\, \log p_t\,,$$

which is the form used in practice for numerical stability.

As explained in Section 3.3.2, the other two output modes are defined via a minimum-cost flow over a bipartite graph. Edges from source to lost-language nodes have unit capacity; edges from known-language nodes to sink have capacity C (a hyperparameter); the edges between languages have capacity 1. The two modes differ only in how the edge costs d_{ij} (from x_i to y_j) are computed.

In the first flow mode, set $d_{ij} = -\log \mathbf{P}_\theta(y_j \mid x_i)$, i.e., reuse the corpus-constrained MLE score.

In the second flow mode, the character probabilities $\{p_t\}$ are used to sample multiple candidate Ancient Greek words, which are then compared to each lexicon word y_j via normalized edit distance. For each lost language item x_i we estimate, for every lexicon word y_j, an expected normalized edit distance under the decoder's output distributions:

1. **Sample S strings from the decoder.** Using the per-step character distributions $\{p_t\}_{t=1}^{T_{\max}}$, optionally sharpened by a temperature $\alpha > 1$, independently draw S strings and truncate at EOS: $s_{i,k} = (s_{i,k,1}, \ldots, s_{i,k,T_{s_{i,k}}})$, $k = 1, \ldots, S$.

[1] Figure 3.2 prepared by Jiaming Luo.

2. **Score each sample by its (log-)likelihood.** With an EOS mask $m_{i,k,t} \in \{0,1\}$, define

$$\log \mathbf{P}(s_{i,k} \mid x_i) \;=\; \sum_{t=1}^{T_{\max}} m_{i,k,t}\, \log p_t\big[s_{i,k,t}\big].$$

3. **Compute normalized edit distances to each candidate.** For every lexicon word y_j and each sample $s_{i,k}$,

$$\delta\big(y_j, s_{i,k}\big) \;=\; \frac{\mathrm{Lev}\big(y_j, s_{i,k}\big)}{\min\big(|y_j|, |s_{i,k}|\big) + 1}\,.$$

   Also the "original" candidate with zero distance is included: $\delta\big(y_j, s_{i,0}\big) \equiv 0$ and score $\log \mathbf{P}(s_{i,0} \mid x_i) \equiv \log \mathbf{P}_\theta(y_j \mid x_i)$.

4. **Weight and average.** Convert the $(1+S)$ log-scores $\{\log \mathbf{P}(s_{i,k} \mid x_i)\}_{k=0}^{S}$ to non-negative weights $\{w_{i,k}\}_{k=0}^{S}$ proportional to their likelihoods and normalized to sum to 1. Set the edge cost to the expected normalized edit distance:

$$d_{ij} \;=\; \sum_{k=0}^{S} w_{i,k}\, \delta\big(y_j, s_{i,k}\big).$$

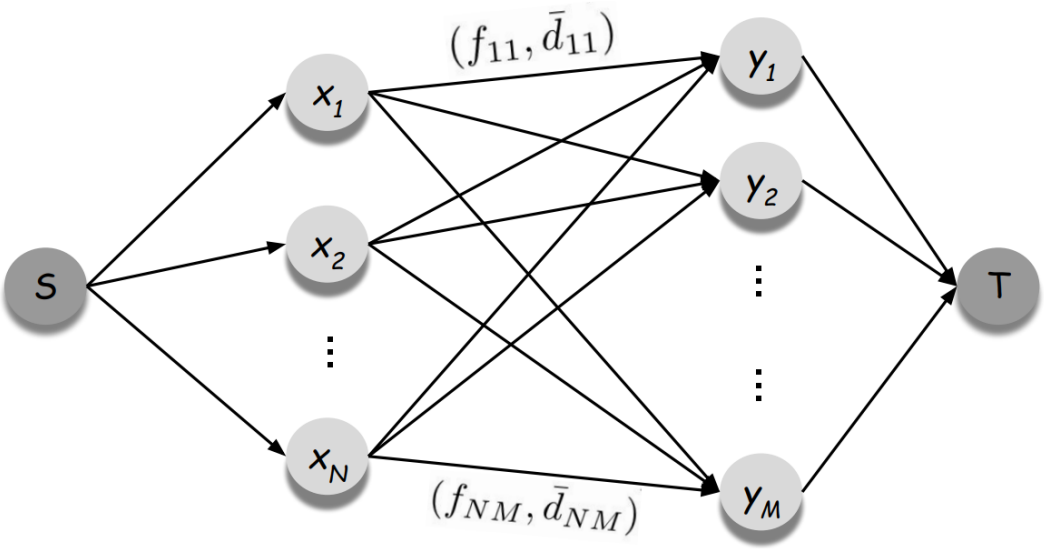


Figure 3.3. Overview of the min-cost flow architecture.[2]

3.3.5 The loss functions

The model is trained by minimizing the objective function introduced in Section 3.3.2. However, the flow solver returns sparse values, as the flow values for the

[2]Figure 3.3 prepared by Jiaming Luo.

edges are mostly zeros. This would likely discard most valid cognate pairs during training.

To address this issue, Luo proposed to use an exponential decay for the flow values in the loss function computation. These values are updated in each E-step and used in the M-step to compute the loss. Concretely, the flow values at iteration τ are updated as follows:

$$f_{ij}^{(\tau)} \;=\; \gamma\, f_{ij}^{(\tau-1)} + (1-\gamma)\, \tilde{f}_{ij}^{(\tau)} \quad \forall\, i, j,$$

where $\tilde{f}_{ij}^{(\tau)}$ are the flow values returned by the flow solver at iteration τ, and $\gamma \in [0, 1)$ is a hyperparameter that controls the decay rate.

The loss function is then defined as:

$$\mathcal{L}_{\text{NLL}} = -\sum_{y_j \in \mathcal{Y}} f_j^{(\tau)} \log \left(\sum_{x_i \in \mathcal{X}} \exp\Big(\log f_{ij}^{(\tau)} + \log \mathbf{P}_\theta(y_j \mid x_i) \Big) \right) \;=$$

$$-\sum_{y_j \in \mathcal{Y}} f_j^{(\tau)} \log \left(\sum_{x_i \in \mathcal{X}} f_{ij}^{(\tau)} \cdot \mathbf{P}_\theta(y_j \mid x_i) \right) = -\log \left(\prod_{y_j \in \mathcal{Y}} \left(\sum_{x_i \in \mathcal{X}} f_{ij}^{(\tau)} \cdot \mathbf{P}_\theta(y_j \mid x_i) \right)^{f_j^{(\tau)}} \right),$$

where $f_j^{(\tau)} = \sum_{x_i \in \mathcal{X}} f_{ij}^{(\tau)}$ is the aggregated flow into node y_j and acts as a weight for the loss of that word.

An additional loss component is introduced to encourage monotonic alignments. This component, called monotonic alignment regularization, penalizes unaligned positions of predicted characters. The loss acts directly on the attention weights α_t produced by the Global Attention module.

Concretely, the attention weights are interpreted, for each word of the lost language x_i, as the alignment probability over the input sequence at each decoding step t. Therefore, $\alpha_t^{(i)}(k) = \mathbf{P}(a_t^{(i)} = k \mid x_i)$, where $a_t^{(i)}$ is the alignment position at step t for word x_i. This probability is used to derive the expected alignment position for each character of the output sequence:

$$pos_t^{(i)} \;=\; \sum_{k=1}^{L} k \cdot \alpha_t^{(i)}(k) \;=\; \sum_{k=1}^{L} k \cdot \mathbf{P}(a_t^{(i)} = k \mid x_i)\,.$$

The monotonic alignment regularization loss is then defined as:

$$\mathcal{L}_{\text{MAR}} \;=\; \sum_{x_i \in \mathcal{X}} \sum_{t=1}^{T_{\max}} \big(pos_t^{(i)} - pos_{t-2}^{(i)} - 1 \big)^2,$$

to accommodate the fact that Linear B is a syllabic language and usually one Linear B sign corresponds to two Greek letters. An example of alignment between a Linear B word and its Ancient Greek cognate is presented in Figure 3.4.

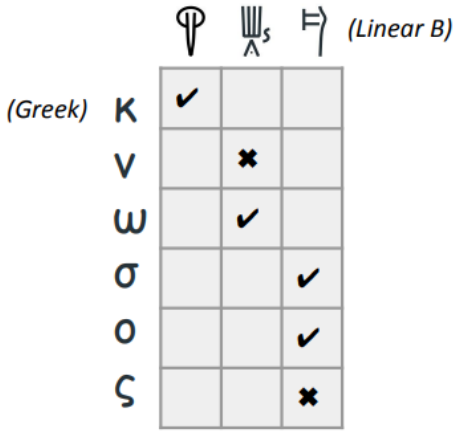


Figure 3.4. Example of alignment between a Linear B word and its Ancient Greek cognate. ✓ and ✘ indicate correct and incorrect alignment positions, respectively. The misalignment between (no) and ν corresponds to a deletion error, while the misalignment between (so) and ς corresponds to an insertion error.[3]

The final loss function is a weighted sum of the two components:

$$\mathcal{L} \;=\; \mathcal{L}_{\mathrm{NLL}} + \lambda\, \mathcal{L}_{\mathrm{MAR}}\,,$$

where λ is a hyperparameter that controls the trade-off between the two loss components.

3.3.6 Training Procedure

As anticipated in Section 3.3.2, training proceeds in an EM-style loop. We initialize the alignment flow matrix with a simple prior and then alternate between:

- **M-step:** fit the seq2seq parameters $\theta^{(\tau)}$ by maximizing the corpus likelihood under the fixed flows $f^{(\tau-1)}$.
- **E-step:** recompute edge costs from $\theta^{(\tau)}$, solve a minimum-cost flow to obtain raw flows $\tilde{f}^{(\tau)}$, and update the stable flows via exponential smoothing:

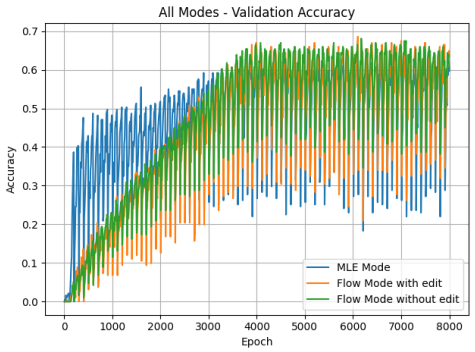
$$f_{ij}^{(\tau)} \;=\; \gamma\, f_{ij}^{(\tau-1)} \;+\; (1-\gamma)\, \tilde{f}_{ij}^{(\tau)}\,.$$

To reduce overfitting to the previous iterate and to avoid poor local minima, the neural parameters are re-initialized at the start of each new M-step. This procedure is summarized in the pseudocode of Algorithm 2.

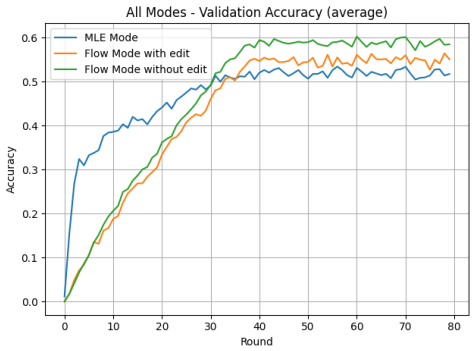
This behavior yields distinctive loss/accuracy trajectories for each split (Figure 3.5): curves change sharply at E-M boundaries because flows are recomputed and model parameters are re-initialized. A practical mitigation is to aggregate within each E-step (e.g., average across its M-steps) or, more conservatively, to report only the final M-step metrics.

[3]Figure 3.4 prepared by Jiaming Luo.

The full training procedure and loss computations are graphically illustrated in Figure 3.6.



(a) Accuracy at each M-step.



(b) Average accuracy at each E-step.

Figure 3.5. Validation accuracy during training. (a) Raw per M-step accuracy. (b) Accuracy averaged within each E-step.

Algorithm 2: Iterative training with EM and minimum-cost flow [4]

Input : $\mathcal{X}, \mathcal{Y}$ (vocabularies); T (number of iterations); N (number of cognate pairs to identify)

Output: $f_{ij}^{(T)}$ (final soft alignments/flows)

$f_{ij}^{(0)} \leftarrow \dfrac{N}{|\mathcal{X}|\,|\mathcal{Y}|}$ `// Initialize uniform flow`

for $\tau \leftarrow 1$ **to** T **do**

  $\theta^{(\tau)} \leftarrow \text{MLE-TRAIN}\big(f_{ij}^{(\tau-1)}\big)$

  $d_{ij}^{(\tau)} \leftarrow \text{EDIT-DIST}\big(x_i, y_j;\, \theta^{(\tau)}\big)$

  $\tilde{f}_{ij}^{(\tau)} \leftarrow \text{MIN-COST-FLOW}\big(d_{ij}^{(\tau)}\big)$

  $f_{ij}^{(\tau)} \leftarrow \gamma\, f_{ij}^{(\tau-1)} + (1-\gamma)\, \tilde{f}_{ij}^{(\tau)}$

  $\textsc{Reset}\big(\theta^{(\tau)}\big)$

return $f_{ij}^{(T)}$

Function *MLE-TRAIN*$\big(f_{ij}^{(\tau)}\big)$

  $\theta^{(\tau)} \leftarrow \arg\max_{\theta} \prod_{y_j \in \mathcal{Y}} \mathbf{P}_\theta\big(y_j \mid \mathcal{X}, \mathcal{F}\big)$

  return $\theta^{(\tau)}$

[4] Algorithm 2 taken by Jiaming Luo.

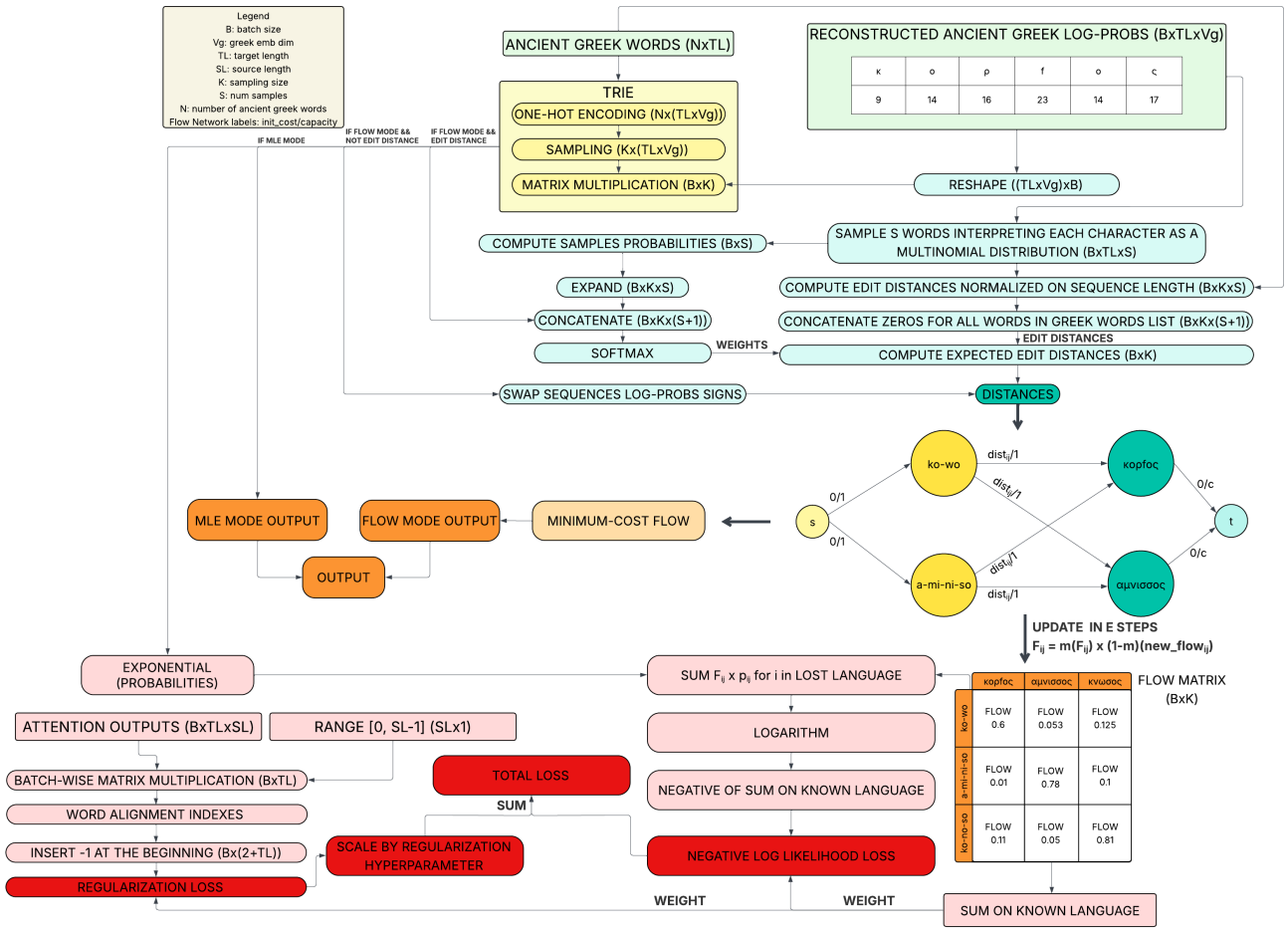


Figure 3.6. Graphical outline of Luo's training procedure.

3.3.7 Results

In this section, I present the results of my experiments with the cognate matching model on three datasets:

- **Luo's dataset** [12], which contains 919 cognate pairs.
- **Luo's corrected dataset**, which contains 976 cognate pairs, obtained by applying some manual corrections on Luo's dataset.
- **Our dataset**, which contains 1911 cognate pairs from both Luo's dataset and Tselentis' [17] dataset, with additional manually identified pairs using the brute-force algorithm, and Ventris and Chadwick's notes [18].

Overall, I ran two sets of experiments. In the first set, I compared the performance of the model on the full datasets, without defining train/validation/test splits. In the second set, I defined train/validation/test splits for both Luo's dataset and our dataset, and compared the performance of the model on each split. The sizes of the splits are the following: 80% of samples for training, 10% for validation,

and 10% for test. For both sets of experiments, I compared the model across all output modes, with and without initializing the LSTM cell states with FastText embeddings. The results are presented in the tables below.

LSTM Initialization	MLE	Flow without edit	Flow with edit
zero	0.830	0.831	0.781
custom	0.822	0.822	0.820

Table 3.1. Comparison of model performance on Luo's original dataset.

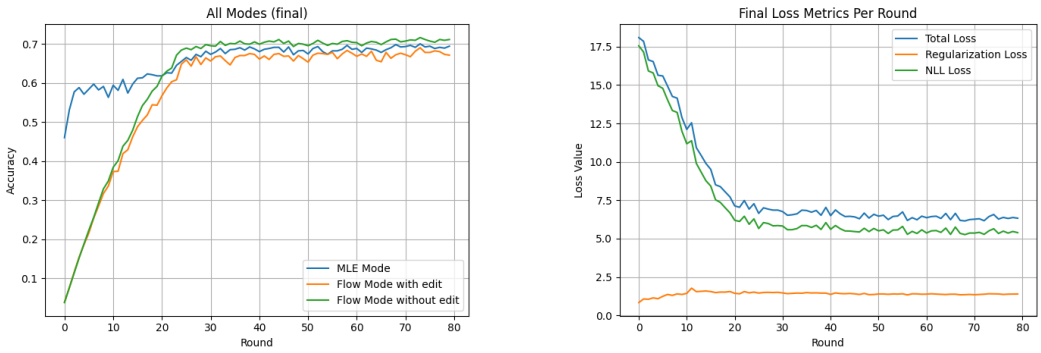
LSTM Initialization	MLE	Flow without edit	Flow with edit
zero	0.827	0.828	0.781
custom	0.841	0.841	0.840

Table 3.2. Comparison of model performance on Luo's corrected dataset.

LSTM Initialization	MLE	Flow without edit	Flow with edit
zero	0.694	0.711	0.671
custom	0.662	0.662	0.672

Table 3.3. Comparison of model performance on our dataset.

It can be observed that, despite the stabilization effect of the custom LSTM initialization, when the model converges with a zero initialization it achieves better performance. Particularly, the model attains higher accuracy on our more challenging and extensive dataset, reaching 71.1% with the flow without edit mode. The final accuracy and loss at each E-step during training on our dataset with zero initialization of the cell states are presented in Figure 3.7.



(a) Final accuracy at each E-step. **(b)** Final loss at each E-step.

Figure 3.7. Metrics for the zero-initialized model during training on our dataset. (a) Accuracy. (b) Loss.

The same trends are observed in the experiments with train/validation/test splits. However, due to the reduced size of the training set and the removal of some cognate pairs from it, the overall performance is lower. While the zero-initialized

model remains strong, the custom-initialized variant degrades substantially. The results are presented in the tables below.

LSTM Init	Split	MLE	Flow w/o edit	Flow w/ edit
zero	Train	0.789	0.790	0.735
	Validation	0.663	0.674	0.696
	Test	0.761	0.761	0.728
custom	Train	0.411	0.423	0.435
	Validation	0.391	0.424	0.435
	Test	0.359	0.359	0.435

Table 3.4. Performance (Train/Validation/Test) on Luo's original dataset.

LSTM Init	Split	MLE	Flow w/o edit	Flow w/ edit
zero	Train	0.782	0.785	0.732
	Validation	0.717	0.717	0.663
	Test	0.739	0.739	0.696
custom	Train	0.457	0.469	0.486
	Validation	0.446	0.467	0.511
	Test	0.370	0.380	0.511

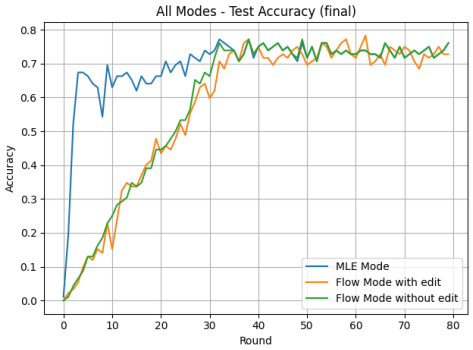
Table 3.5. Performance (Train/Validation/Test) on Luo's corrected dataset.

LSTM Init	Split	MLE	Flow w/o edit	Flow w/ edit
zero	Train	0.645	0.658	0.639
	Validation	0.597	0.602	0.634
	Test	0.625	0.646	0.646
custom	Train	0.238	0.275	0.367
	Validation	0.262	0.262	0.377
	Test	0.224	0.234	0.370

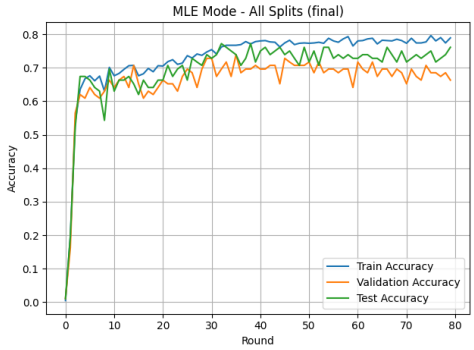
Table 3.6. Performance (Train/Validation/Test) on our dataset.

This significant drop in the custom-initialized runs is consistent with a loss-accuracy divergence caused by a representation mismatch. FastText injects word-level semantic priors into a character-mapping task. As a result, the model becomes confidently wrong on many instances: the negative log-likelihood can keep improving (higher confidence), while accuracy on held-out data declines (worse argmax decisions). In short, the initialization biases the decoder toward signals that are orthogonal to the target grapheme-level correspondences, leading to miscalibration and poorer generalization, due to representational mismatch.

The final accuracy of all splits, using MLE mode, and the final accuracy of the test split, using all output modes, of the zero-initialized model during training on our dataset are presented in Figure 3.8.



(a) Final accuracy at each E-step for the test set in all modes.



(b) Final accuracy at each E-step in MLE mode for all splits.

Figure 3.8. Metrics for the zero-initialized model during training on our dataset. (a) Accuracy for the test set in all modes. (b) Accuracy in MLE mode for all splits.

Chapter 4

Auxiliary Classifiers

To strengthen cognate predictions within the translation pipeline, I introduced a set of auxiliary tasks that supply syntactic and morphological cues. These tasks are:

1. Part of speech detection
2. Noun type classification (restricted to nouns and adjectives)
3. Inflection detection

They were chosen because they encode information that helps constrain the space of plausible Greek correspondences (and downstream meanings), especially under low-resource conditions. A separate classifier was trained for each task.

4.1 Dataset Creation

The dataset for these auxiliary tasks was assembled via prompt engineering with Gemini 2.0 Flash. It includes 3162 sampled words; a small subset carries partial labels (available for only one of the tasks).

The principal sources were:

- Our cognate dataset, which provides proposed Ancient Greek correspondences from which labels can be inferred;
- Ventris and Chadwick's notes [18], which also contain usable labels.

The prompts were XML-structured on input and constrained to strict JSON on output. The key prompt components were:

- **Task definitions and label sets:** clear label spaces for word type, part of speech, and inflection, with explicit instructions for edge cases.
- **Declension guidance:** a Linear B attested declension table (Table 4.1) mapping to Greek first/second/third declensions, to anchor inflection decisions.
- **Verbal morphology cues:** common endings and diagnostics (e.g., 3rd pl. *-si*; participles in *-me-no/-me-na*; infinitival patterns with final *-e*).

- **Adjectival behavior:** adjectives follow the same inflectional classes as nouns (thematic in *-o*, thematic in *-a*, athematic).
- **Affixes to ignore in analysis:** *-qe* (conj. "and"), *-te* (ablatival), *-de* (allative/demonstrative or negative prefix, context-dependent), *-pi* (instrumental/locative); the classifier should label the base form without these.
- **Consistency checks:** automatic sanity checks (e.g., verbs/adverbs must have inflection $= -1$, inflection must match endings and be compatible with the proposed cognates).

Table 4.1. Linear B declensional endings attested in the corpus (mapped to Greek classes). The first two columns correspond to Greek second declension, the next two to first declension, and the last two to third declension. Adjectives follow the same patterns (first/second vs. third).

Number	Case	Thematic *-o* (M.)	Thematic *-o* (N.)	Thematic *-a* (M.)	Thematic *-a* (F.)	Athematic (M./F.)	Athematic (N.)
Sing.	Nom.	*-o*	*-o*	*-a*	*-a*	variable	variable
Sing.	Gen.	*-ojo*	*-ojo*	*-ao*	*-a*	*-o*	*-o*
Sing.	Dat.	*-o*	*-o*	*-a*	*-a*	*-e/-i*	*-e/-i*
Sing.	Acc.	*-o*	*-o*	*-a*	*-a*	*-a*	variable (often = Nom.)
Plur.	Nom.	*-o/-oi*	*-a*	*-a*	*-a*	*-e*	*-a*
Plur.	Gen.	*-o*	*-o*	*-ao*	*-ao*	*-o*	*-o*
Plur.	Dat.	*-oi*	*-oi*	*-ai*	*-ai*	*-si/-ti*	*-si/-ti*
Plur.	Acc.	*-o*	*-a*	*-a*	*-a*	*-a/-e*	*-a*

Note. Linear B orthography is syllabic: these endings appear within syllabic constraints and can be preceded by arbitrary consonants.

In order to enforce determinism, I set a low temperature and greedy decoding (temperature $= 0$; top-k $= 1$), so that labeling is repeatable.

The full prompt code and the resulting dataset are available in the source code repository. The CSV file containing the dataset has fields:

```
linear_b,word_type,part_of_speech,inflection,confidence,reasoning
```

4.2 Task Definitions

Each task is a multi-class classification problem, with the following definitions.

4.2.1 Part of Speech detection

The part of speech detection task classifies words into the following categories:

0. Noun
1. Verb
2. Adjective

3. Adverb

It aims to provide syntactic context that can help disambiguate cognate predictions. For example, knowing that a word is a verb narrows down the plausible Greek correspondences and meanings. In practice, the classifier relies on surface cues in Linear B (e.g., 3rd plural *-si*; participial endings *-me-no/-me-na*; infinitival patterns with final *-e*), the proposed Greek cognates when available (morphology and lemma semantics), and lexicon notes. By policy, participles are always labeled as adjectives (not verbs). Also, functional suffixes (*-qe*, *-te*, *-de*, *-pi*) are ignored when deciding Part of Speech.

Examples.

- 𐀁𐀍𐀖(e-ko-si) → εχουσι 'they have': the final *-si* is a reliable 3rd plural marker; the item is a verb (part_of_speech= 1; inflection= −1).
- 𐀁𐀍𐀚𐀲(e-ko-me-na) → εχομενα '(things) being held': the *-me-na* participial ending indicates a participle, which we treat as an adjective (part_of_speech= 2), not a verb.

4.2.2 Noun type classification

Noun type classification focuses on distinguishing between different types of nouns and adjectives, based on the same information often provided by the Chadwick and Ventris' notes [18]. The classes are:

0. Proper names (including anthroponyms, theonyms and animal names)
1. Toponyms
2. Ethnonyms (names of populations)
3. Common nouns/adjectives

This is the most challenging task. Ambiguity between onomastic and common uses, heavy class imbalance (few ethnonyms vs. many common items), and context dependence in the tablets all conspire to make boundaries fuzzy. Evidence comes from onomastic patterns (e.g., divine titles, personal-name morphology), tablet context, and the semantics of the proposed Greek cognates. When a form functions adjectivally but names a deity or place (e.g., divine epithets, demonyms), it is categorized by its onomastic role (theonym, toponym, ethnonym) rather than its syntactic function.

Examples.

- 𐀔𐀜𐀬𐀪(po-ti-ni-ja) → ποτνια 'mistress, lady (title)': used as a divine title; proper name, theonym (word_type= 0).
- 𐀡𐀯𐀰(pa-i-to) → φαιστος (Phaistos): city name; toponym (word_type= 1).

4.2.3 Inflection detection

Inflection detection classifies nouns and adjectives according to their inflectional class, based on the declensional patterns attested in the Linear B corpus and mapped to Ancient Greek declensions (Table 4.1). The classes are:

0. Thematic in *-o* (masculine/neuter)
1. Thematic in *-a* (feminine/masculine)
2. Athematic (feminine/masculine/neuter)

This task is also non-trivial, as Linear B endings can overlap across classes: athematic genitive *-o* can mimic thematic patterns, and neuter nominative/accusative *-a* may coincide with thematic *-a*. To resolve ties, the classifier cross-checks the Greek cognate's morphological class and expected case/number patterns. Adjectives follow the same inflectional mapping as nouns; participles (treated as adjectives) are assigned to the appropriate class based on their endings. This mapping maps directly to Ancient Greek decletions table, where however most Linear B ambiguities are resolved. For this reason, also the Ancient Greek cognate is an important information to resolve the ambiguities and correctly classify some dubious words.

Examples.

- 𐀁𐀊𐀍𐀋𐀖(e-re-u-te-ro) → ελευθερος 'free': adjective with *-o* pattern; thematic in *-o* (inflection= 0).
- 𐀳𐀛𐀊𐀍(a-ke-re-u) → ἀγρευς 'hunter': athematic (inflection= 2), consistent with the *-eus* class.

4.3 Models Comparison

My first approach was to tokenize Linear B words into character sequences, embed them, and pass the embeddings to a BiRNN. The idea was inspired by text-infilling with bidirectional RNNs [13] and by Luo's bidirectional LSTM (Section 3.3.3). In practice, this setup trained slowly and struggled to converge to a satisfactory solution.

I therefore switched to a lighter feature-engineering route that reflects the structure of Linear B and plays well with linear classifiers. The key is to represent each word simultaneously at the syllabogram level and at the character n-gram level, and then concatenate the two views: The feature extraction pipeline is described below:

- **Inputs & normalization.** Latinized Linear B forms (e.g., `a-ke-re-u`, `po-ti-ni-ja`); dashes mark syllabogram boundaries. Keep dashes for syllable features; remove them for character n-grams.
- **Syllabogram bag (counts).** `CountVectorizer` with tokenizer $\lambda.x \mapsto x.\texttt{split}(\texttt{"-"})$ to build a bag-of-syllables (`a`, `ke`, `re`, `u`, `nwa`, `phu`); captures sign-level patterns (e.g., `-me-na`, final `-o`, titles in `-ta`).

- **Character n-grams (TF-IDF).** `TfidfVectorizer` with `analyzer=char`, `ngram_range` $= (2, 4)$ on dashless strings; emphasizes short motifs (`qo`, `ojo`, `ph`, `oi`). These features are concatenated with the syllabogram bag via `FeatureUnion`.
- **Output matrix.** Sparse X_{features}: rows = words; columns = syllabograms + n-grams. Suits fast linear models (e.g., `LinearSVC`); unseen tokens at test time are ignored.

```
from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
from sklearn.pipeline import FeatureUnion

features = FeatureUnion([
    ("syllables", CountVectorizer(
        tokenizer=lambda x: x.split("-"), token_pattern=None)),
    ("char_ngrams", TfidfVectorizer(
        analyzer="char", ngram_range=(2,4),
        preprocessor=lambda x: x.replace("-", "")))
])

X_features = features.fit_transform(forms)  # sparse matrix
```

Listing 4.1. Syllabogram counts + char n-gram TF-IDF (concatenated) and transform

This representation is efficient and effective, as it captures both the syllabic structure of Linear B and the morphological cues encoded in character sequences. As a result, it enables fast training and good generalization with most classifiers.

The classifiers compared in my experiments are:

- **Logistic Regression** (`LogisticRegression`): a linear model that estimates class probabilities via the logistic function; suitable for multi-class tasks with regularization.
- **Random Forest** (`RandomForestClassifier`): an ensemble of decision trees that aggregates predictions via majority voting; captures non-linear patterns and interactions.
- **Linear SVM** (`LinearSVC`): a support vector machine with a linear kernel; effective for high-dimensional sparse data.
- **Multinomial Naive Bayes** (`MultinomialNB`): a probabilistic classifier based on Bayes' theorem; assumes feature independence and is efficient for text data.
- **Histogram-based Gradient Boosting** (`HistGradientBoostingClassifier`): an efficient implementation of gradient boosting that uses histograms to bin continuous features; handles large datasets well.
- **Neural Network (MLP)** (`MLPClassifier`): a feedforward neural network with one hidden layer; capable of learning complex non-linear decision boundaries.

The implementation of these classifiers is taken from the `scikit-learn` library. Let us now compare the performance of different classifiers on the three tasks.

4.3.1 Results

The classifiers were evaluated using 5-fold cross-validation on the dataset of 3162 words.

Part of Speech detection. The results are summarized in Table 4.2.

Table 4.2. 5-fold cross-validation results for the Part of Speech task (mean values).

Model	Accuracy	Macro-F1
Logistic Regression	**0.8848**	0.3826
Random Forest	0.8770	0.2745
Linear SVM	0.8845	**0.4129**
Multinomial Naive Bayes	0.8712	0.2439
HistGradientBoosting	0.8777	0.3390
Neural Network (MLP)	0.8790	0.3022

Logistic Regression attains the highest accuracy, while Linear SVM matches it closely and yields the best Macro-F1, indicating better performance on minority classes. Overall, this task shows the best accuracies across models; however, class imbalance (nouns/adjectives dominate) inflates accuracy, as reflected by the more modest Macro-F1 values.

Noun type classification. The results are summarized in Table 4.3.

Table 4.3. 5-fold cross-validation results for the Noun type classification task (mean).

Model	Accuracy	Macro-F1
Logistic Regression	0.6341	0.4045
Random Forest	0.6015	0.2953
Linear SVM	**0.6401**	**0.4632**
Multinomial Naive Bayes	0.6044	0.2978
HistGradientBoosting	0.6012	0.3841
Neural Network (MLP)	0.6319	0.3653

This task is clearly more challenging, with lower overall accuracy. Linear SVM again performs best in both accuracy and Macro-F1, suggesting it handles class imbalance better than the alternatives.

Inflection detection. The results are summarized in Table 4.4.

Table 4.4. 5-fold cross-validation results for the Inflection detection task (mean values).

Model	Accuracy	Macro-F1
Logistic Regression	0.8097	0.7755
Random Forest	0.7405	0.6836
Linear SVM	**0.8372**	**0.8072**
Multinomial Naive Bayes	0.7298	0.6690
HistGradientBoosting	0.7855	0.7443
Neural Network (MLP)	0.7993	0.7611

Linear SVM achieves the best accuracy and Macro-F1, with Logistic Regression and the MLP close behind, while tree-based models and Naive Bayes lag in Macro-F1. Overall, inflection detection is easier than noun-type classification and Linear SVM performs very strongly. Note that one of the classes is under-represented, the athematic class, which can affect Macro-F1 despite high overall accuracy.

Bibliography

[1] S. Alexiou. “Minoan Civilization.” In: trans. by C. Ridley. Heraklion, Crete: Spyros Alexiou Sons, 1969. Chap. 2, p. 23.

[2] S. Alexiou. “Minoan Civilization.” In: trans. by C. Ridley. Heraklion, Crete: Spyros Alexiou Sons, 1969. Chap. 3, p. 34.

[3] S. Alexiou. “Minoan Civilization.” In: trans. by C. Ridley. Heraklion, Crete: Spyros Alexiou Sons, 1969. Chap. 4, pp. 50–51.

[4] S. Alexiou. “Minoan Civilization.” In: trans. by C. Ridley. Heraklion, Crete: Spyros Alexiou Sons, 1969. Chap. 5, p. 59.

[5] Vivi Andersson et al. “UPPERCASE IS ALL YOU NEED.” In: *Proceedings of SIGBOVIK 2025*. Presented at SIGBOVIK 2025, Carnegie Mellon University, April 4, 2025. Association for Computational Heresy. Pittsburgh, PA, USA, Apr. 2025. URL: https://sigbovik.org/2025/proceedings.pdf.

[6] Maximilian Beurer-Kellner, Jürgen Cito, and Zhendong Su. “LMQL: Prompting Is Programming.” In: *arXiv preprint arXiv:2212.06094* (2023). arXiv: 2212.06094 [cs.CL]. URL: https://arxiv.org/abs/2212.06094 (visited on 08/31/2025).

[7] J. Chadwick. “The Decipherment of Linear B.” In: Cambridge, United Kingdom: Cambridge University Press, 1958. Chap. 2, pp. 22–25.

[8] J. Chadwick. “The Decipherment of Linear B.” In: Cambridge, United Kingdom: Cambridge University Press, 1958. Chap. 3, pp. 26–28, 33–35.

[9] J. Chadwick. “The Decipherment of Linear B.” In: Cambridge, United Kingdom: Cambridge University Press, 1958. Chap. 4, pp. 40–66.

[10] J. Chadwick. “The Decipherment of Linear B.” In: Cambridge, United Kingdom: Cambridge University Press, 1958. Chap. 5, pp. 75–76.

[11] J. Chadwick. “The Decipherment of Linear B.” In: Cambridge, United Kingdom: Cambridge University Press, 1958. Chap. 6, pp. 81–84.

[12] Jiaming Luo, Yuan Cao, and Regina Barzilay. “Neural Decipherment via Minimum-Cost Flow: From Ugaritic to Linear B.” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3146–3155. DOI: 10.18653/v1/P19-1303. URL: https://aclanthology.org/P19-1303/.

[13] Katerina Papavassileiou, Dimitrios I. Kosmopoulos, and Gareth Owens. "A Generative Model for the Mycenaean Linear B Script and Its Application in Infilling Text from Ancient Tablets." In: *J. Comput. Cult. Herit.* 16.3 (Aug. 2023). ISSN: 1556-4673. DOI: 10.1145/3593431. URL: https://doi.org/10.1145/3593431.

[14] E. Salgarella. "Aegean Linear Script(s): Rethinking the Relationship between Linear A and Linear B." In: Cambridge, United Kingdom: Cambridge University Press, 2020. Chap. 1, pp. 1–7, 31–36, 39–40.

[15] E. Salgarella. "Aegean Linear Script(s): Rethinking the Relationship between Linear A and Linear B." In: Cambridge, United Kingdom: Cambridge University Press, 2020. Chap. 5, p. 378.

[16] E. Salgarella. "Aegean Linear Script(s): Rethinking the Relationship between Linear A and Linear B." In: Cambridge, United Kingdom: Cambridge University Press, 2020. Chap. 3, pp. 209–215.

[17] Chris Tselentis. *Linear B Lexicon*. Athens, Greece, Apr. 9, 2011. URL: https://archive.org/details/LinearBLexicon.

[18] Michael Ventris and John Chadwick. *Documents in Mycenaean Greek: Three Hundred Selected Tablets from Knossos, Pylos and Mycenae, with Commentary and Vocabulary*. 2nd ed. Cambridge: Cambridge University Press, 1973.

[19] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." In: *arXiv preprint arXiv:2201.11903* (2022). arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903 (visited on 08/31/2025).