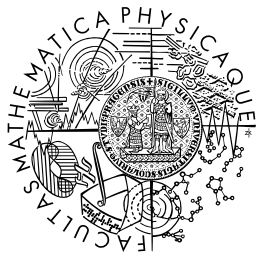


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Jan Provazník

Textual Ciphers as a Tool for Better Understanding the Transformers

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: Mgr. Jindřich Libovický, Ph.D.
Study programme: Computer Science

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature, and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague date 2024–05–09 Jan Provazník

I would like to thank my supervisor Mgr. Jindřich Libovický, Ph.D., for his guidance throughout my work on this thesis. I would also like to thank my family and friends for their support and encouragement.

Title: Textual Ciphers as a Tool for Better Understanding the Transformers

Author: Jan Provazník

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Jindřich Libovický, Ph.D., Institute of Formal and Applied Linguistics

Abstract: The Transformer architecture is very popular, so it is potentially impactful to interpret what influences its performance. We test the hypothesis that the model relies on the linguistic properties of a text when working with it. We remove interference with cultural aspects of meaning by using a character-level task with the ByT5 Transformer model. We fine-tune ByT5-small to decipher sentences encrypted with text ciphers (Vigenère, Enigma). We annotate a sentence dataset with linguistic properties using published NLP tools. On this dataset, we study the relationships between the linguistic properties and the fine-tuned ByT5 decipherment error rate. We analyze correlations, train ML models to predict error rates in sentences from the properties, and interpret feature importance with SHAP. We find small significant correlations but fail to predict error rates from the properties. We conclude the properties we identified do not give much insight into the performance of the Transformer.

Keywords: Transformer NLP interpretability deep learning ciphers

Název práce: Textové šifry jako nástroj pro lepší pochopení modelů Transformer

Autor: Jan Provazník

Ústav: Ústav formální a aplikované lingvistiky

Vedoucí: Mgr. Jindřich Libovický, Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt: Architektura Transformer je velmi populární, takže může být potenciálně významné interpretovat, co ovlivňuje její výkon. Testujeme hypotézu, že model se při práci s textem spoléhá na jeho lingvistické vlastnosti. Abychom eliminovali vliv kultury na význam, používáme úlohu pracující na úrovni znaků s Transformer modelem ByT5. Dotrénujeme ByT5-small na dešifrování vět zašifrovaných pomocí textových šifer (Vigenère, Enigma). Anotujeme evaluační dataset vět pomocí publikovaných nástrojů pro NLP. Na evaluačním datasetu zkoumáme vztahy mezi lingvistickými vlastnostmi a četností chyb dotrénovaného ByT5 při dešifrování vět. Analyzujeme korelace, trénujeme ML modely na predikci četnosti chyb věty z jijích lingvistických vlastností a interpretujeme důležitost vlastností pomocí SHAP. Nacházíme malé signifikantní korelace, ale predikce četnosti chyb z vlastností selhává. Dospíváme k závěru, že identifikované vlastnosti neposkytují vhled do výkonu Transformerů.

Klíčová slova: Transformer NLP interpretovatelnost deep learning šifry

Contents

Introduction

Recent advances in NLP (Natural Language Processing) have been driven by deep neural networks (DNNs) and in particular by the *Transformer* architecture [Vas+17]. Trained Transformer models are commonly referred to as LLMs (Large Language Models), 'large' because they have billions of parameters, and 'language' because they operate on sequences of text. Billions of numbers are hard, if not impossible to understand for humans, which is in tension with our need to understand them for regulation, debugging, and trust as they are deployed to all areas of life.

Years after LLM releases and their rise in popularity, there still is not a sufficient understanding or even a standard methodology for investigating why LLMs are so good and where they fail. One approach is mechanistic interpretability, which finds explanations by looking at the inner workings of the model and trying to make sense of the billions of numbers, decomposing them into algorithms or circuits with a clear purpose. Another approach is behavioral or data-driven interpretability, which finds explanations by looking at the model's behaviors, accepting that they are black boxes with inscrutable insides.

In this thesis, we choose the *behavioral* approach to interpret the Transformer model. Through this lens, we want to understand the relationship between the Transformer's performance and the linguistic properties of the input text.

We remove interference with cultural aspects of meaning by using a character-level task with the *ByT5-small* model [Xue+22]. The task is deciphering sentences encrypted with substitution ciphers: Vigenère and Enigma. The decipherment task allows us to focus on quantifiable linguistic properties of the sentences, sidestepping hard-to-quantify semantics. It is easy to implement training as a variant of translation; evaluation consists of comparing the Transformer's output to the correct decryption on a character level. With publicly available NLP pipelines and simple algorithms, we measure various linguistic properties with which we annotate sentences in evaluation datasets in 3 languages. We fine-tune ByT5 on the decipherment task in English, Czech, and German.

We study the relationship between Transformer performance and *automatically annotated linguistic properties (AALP)* at different points in training. We ana-

lyze correlations, predict error rates from property values with simple ML models (Linear Regression, MLP, Random Forest, XGBoost), and assess how important each property is for error rate predictions with SHAP. SHAP is a game theory-based method for interpreting the predictions of ML models by attributing how input features contribute to them.

We find that the performance of the Transformers fine-tuned on the decipherment task cannot be consistently well explained by the AALP values. This result suggests that the properties we identified are not fundamental to the Transformer's good performance on various tasks.

In the first chapter, we present the background for the studied models, why are they interesting, and how to formulate decipherment as a task for them. In the second chapter, we discuss the notion of interpretability, approaches to it and outline the goal for our experiments. In the third chapter, we ascertain the studied linguistic properties and describe the detailed specifications for our experiments. In the fourth chapter, we present the results of the experiments and discuss how they relate to our goal.

Acknowledgments

We used GitHub Copilot[1] and GPT-4[2][Ope23] when writing and debugging code for experiments and visualizations. We used Grammarly[3], GPT-4, and Github Copilot to typeset the text, format and solicit feedback.

[1] `https://github.com/features/copilot`

[2] `https://openai.com/gpt-4`

[3] `https://app.grammarly.com/`

Chapter 1

Sequence modeling with Transformers

This chapter discusses progress in language modeling and introduces the Transformer model architecture and its context. Next, it goes into the specifics of the ByT5 model; the ciphers we use in our experiments and related work on decipherment with deep neural networks (DNNs).

1.1 Language modeling

Language modeling is a task in Natural Language Processing (NLP). We have a sequence of words or characters and want to estimate the joint probability of it occurring in a language. A common formulation is predicting what next word should come in a sequence. Performing language modeling well requires a model to capture the context and dependencies within the sequence. Before the era of end-to-end DNNs, simple language modeling was a component of systems for generative tasks: automatic speech recognition, and machine translation.

Many other NLP tasks can be stated in a sequence modeling frame. For example: translation is a sequence of text in the source language followed by text in the destination language; summarization is a sequence of text followed by a sequence of its summary; classification can be expressed as the sequence we want to classify followed by a classifying symbol. Due to the simplicity of training and the availability of vast amounts of data, language models are often used as a starting point for creating specialized models for NLP tasks. Language model training as a base for other tasks gained popularity with BERT [Dev+18] reaching state-of-the-art results in classification and question answering. The success of ChatGPT and similar products further shows the effectiveness of this approach.[1]

[1] GPT-4 [Ope23] rewrote this sentence to sound natural.

1.2 Transformer

The seminal paper "Attention is All You Need" by Vaswani et al. [Vas+17] introduced the *Transformer* architecture. It has been a dominant machine learning architecture for NLP tasks since 2017, first in machine translation, later in language modeling and other tasks. The Transformer is based on the idea of self-attention, relating multiple positions in a sequence to compute its representation. The architecture utilizes various techniques accumulated in the field of deep learning, leading to its good performance and success.

The original Transformer consists of an encoder and a decoder, each having multiple layers. Subsequent developments have shown that it is viable to use just the decoder for generative tasks; examples include the GPT family of models [Rad+19]. Encoder-only models are also used, mainly for classification, for example, BERT [Dev+18].

Next, we describe the purpose of each part of the architecture, and Figure 1.1 illustrates how they are connected.

Tokenization

Usually, the Transformer operates on sequences of tokens obtained by *subword tokenization* of the input text. The motivation for subwords is that a natural language corpus empirically has a long-tailed Zipfian distribution of word frequencies. Infrequent words being one token leads to too many tokens, and segmenting every character has not taken off due to inefficiency. As a compromise, with subword tokenization common words and word parts in tokenizer training data result in one token. Rare words are segmented into smaller subwords. Segments of rare words are still seen enough during Transformer training so the model learns to use them. Common algorithms for segmenting to subwords are SentencePiece used by T5 [Raf+19], Byte-Pair Encoding (BPE) used by the GPTs [Rad+19], and WordPiece used by BERT [Dev+18]. A substantial deviation in our experiments compared to state-of-the-art models is that we do not use subword tokenization.

Embedding layer and positional encoding

First, each token is converted to a vector of dimension d_{model} in an Embedding layer. It is implemented as a lookup table that for each possible token has a vector associated with it. The model learns Embedding vectors.

The Attention sublayer interacts with all tokens of the sequence but has otherwise no notion of their order. The *positional encoding* adds vectors of dimension d_{model} to each embedding vector to represent information about the position of the token in the sequence. The original paper [Vas+17] uses a

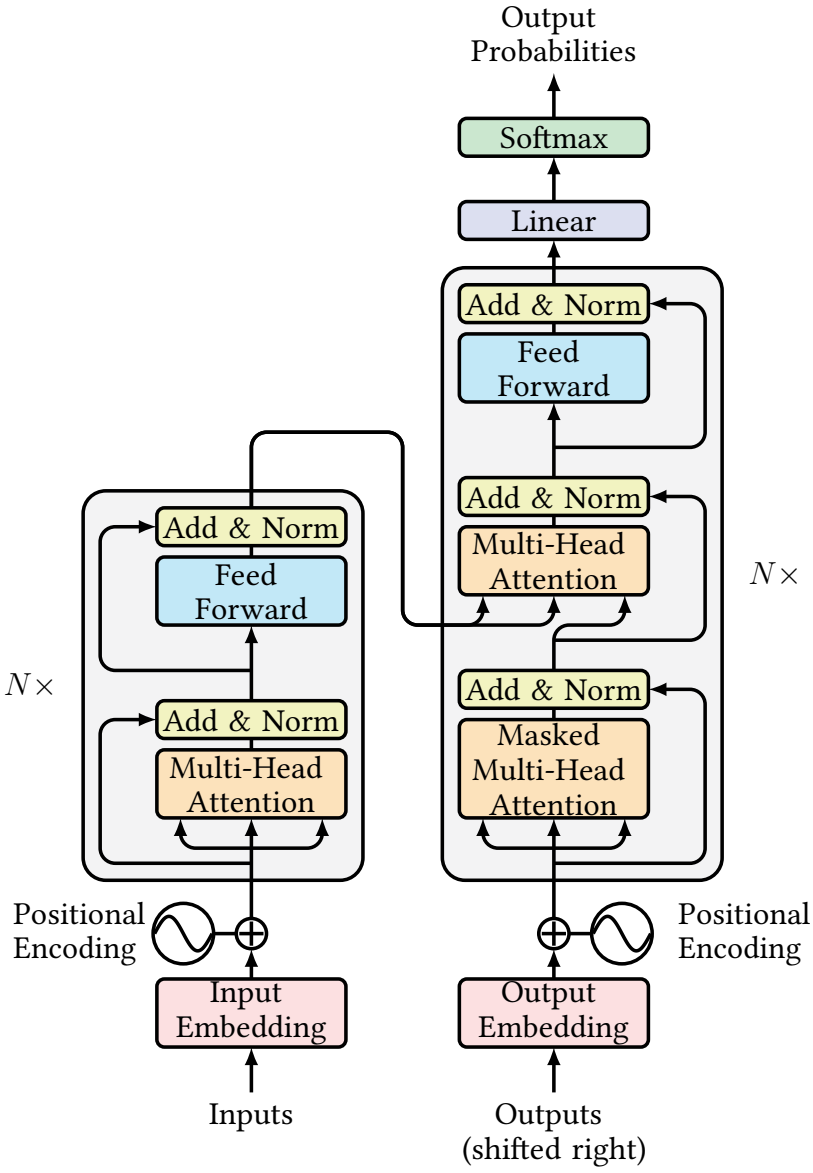


Figure 1.1 Transformer architecture by Vaswani et al. [Vas+17]

sinusoidal function of the position of the token in the sequence. Learned positional embeddings are also used [Dev+18] or token position is handled later in the Attention sublayer [Raf+19].

Attention sublayers

Prior approaches to sequence modeling such as Recurrent Neural Networks (RNNs) had issues with long-range dependencies in sequences. The Transformer addresses this issue with the attention mechanism.

Attention is used in 3 places in the Transformer architecture: self-attention sublayers in the encoder, self-attention sublayers with masking in the decoder, and encoder-decoder attention sublayers. There are H attention heads in each sublayer (usually tens of them) with independent weights.

Definition 1. *Scaled dot-product attention*

$$\begin{aligned} \textit{Attention}(Q, K, V) &= \textit{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \\ \textit{head}_i &= \textit{Attention}(QW_i^Q, KW_i^K, VW_i^V), \\ \textit{MultiHead}(Q, K, V) &= \textit{Concat}(\textit{head}_1, \ldots, \textit{head}_H)W^O. \end{aligned}$$

Transformer uses the multiheaded scaled dot-product attention. The model learns the values of matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $W^O \in \mathbb{R}^{H \cdot d_v \times d_{\text{model}}}$ during training.

Intuitively the attention mechanism facilitates communication between each pair of tokens and in later layers between richer composite representations.

Feed forward sublayers

It is a Multi-Layer Perceptron (MLP) with one hidden layer of dimension d_{ff} (usually $4 \cdot d_{\text{model}}$). Various activation functions can be used in the feed-forward sublayer, the original paper uses the simplest $\text{ReLU}(x) = \max(0, x)$. Recent models use more complicated functions: GELU, Swish, SwiGLU, GeGLU [Sha20].

$$\text{FFN}(X) = \text{activation}(XW_1 + b_1)W_2 + b_2$$

The model learns the weights $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ and biases $b_1 \in \mathbb{R}^{d_{\text{ff}}}$, $b_2 \in \mathbb{R}^{d_{\text{model}}}$ of each feed-forward sublayer during training.

Layer normalization

Layer normalization is a layer proposed by Ba, Kiros, and Hinton [BKH16] for deep learning models. Given a vector X, the output is normalized to have zero

mean and unit variance. It was empirically shown to improve training time and performance. Layer normalization in the Transformer is applied either before or after each sublayer, depending on the specific variant.

Residual connections and droput

A residual connection [He+15] skips a (sub) layer and adds the input of a (sub) layer to the output of another, creating a "shortcut". For a given vector X and a sublayer, the output Y is computed as:

$$Y = X + \text{sublayer}(X)$$

This way the gradient can flow through a layer even if its internal weights are zero. Residual connections are used in the Transformer to help with the vanishing gradient problem. Deep learning models without it can get stuck in training with weights not being updated as the gradient is too small in the early layers.

After each sublayer *dropout* is applied either after the layer normalization or before it. Dropout is a regularization technique that with probability p sets activations in a layer to zero. It is commonly used in deep learning to improve generalization [Sri+14].

Output

After the encoder and decoder, their output is passed through a dense layer and softmax to get the probability distribution of the token we are inferencing. When using the Transformer, we can greedily select the token and feed it back to the model. This is called *greedy decoding*. Alternatively, we can sample from the distribution of probabilities of the next token, optionally we can influence the distribution that is sampled from with a temperature parameter. Or with *beam search* we keep track of the k most probable sets (usually sequences) of multiple inferred tokens and select the best one at the end. Beam search has the potential for better results but a higher computational cost.

1.2.1 Training

The goal of training is to minimize the loss function. The loss function for the Transformer is the cross-entropy between the predicted token distribution for each position and the actual token.

Definition 2 (Cross-Entropy Loss)**.** *Given a set of predicted probabilities $\hat{Y}$ and a corresponding set of target classes Y over a sequence of length T with M possible classes.*

$$\mathit{CrossEntropyLoss}(Y, \hat{Y}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{M} y_{t,c}\log(\hat{y}_{t,c})$$

where:

- *$y_{t,c}$ is a binary indicator (0 or 1) denoting whether class label c is the correct classification for the token at position t.*
- *$\hat{y}_{t,c}$ represents the predicted probability that the token at position t belongs to class c.*

Loss minimization is performed using the backpropagation algorithm and an optimizer (e.g. Adam [KB15]) which computes how to adjust the weights of the model so we hopefully converge to a good solution after a lot of steps.

Language modeling Transformers which are suitable for use on multiple tasks such as T5 [Raf+19] or GPT-4 [Ope23], are trained on at least terabytes of text for ever-increasing amounts of GPU/TPU hours. Kaplan et al. [Kap+20] described that if we want a better performance increasing any of the following helps; model size, training data size, and used compute (training FLOPs). The paper also outlines optimal ratios for these variables. It is possible to train the Transformer on a single machine or a distributed system with many machines and GPUs/TPUs.

Pre-training

The first step of training a modern language model is pre-training, which means training the model on a large, not particularly clean dataset (such as a large portion of the internet), with a lot of computing resources (GPUs/TPUs). A task obtained from plaintext is used, for example, demasking, deshuffling, or predicting the next token in a sequence. The cost of pre-training is very prohibitive so then the pre-trained artifact is used for many downstream tasks.

Fine-tuning

After a model is pre-trained on a large dataset, we can *fine-tune* (further train) it on a smaller, clean dataset to achieve better results on a specific task. For example, to fine-tune a general model for translation, we can use the following supervised pattern:

```
Input: "translate from English to German: <English text>"
Output: "<German translation>"
```

Or simply put source text in the input and target text in the output.

1.3 ByT5

We base our work on the pre-trained *ByT5-small*, a 300M parameter Encoder-Decoder Transformer model presented by Xue et al. [Xue+22]. ByT5 was pre-trained with the span corruption task on the mC4 dataset introduced by Raffel et al. [Raf+19]. ByT5 architecture mirrors the original Transformer, with a few modifications: layer normalization is placed before each sublayer instead of after, its bias is removed, and an alternative relative positional representation is used, adding a scalar to the logits used for computing attention weights. Usually, Encoder-Decoder models use the same number of layers in the encoder as in the decoder, but ByT5 has more layers in the encoder, in a 3:1 ratio.

The distinctive feature of ByT5 is using bytes as tokens interpreted as UTF-8. Common token count in Transformer language models is in the order of 10^5–10^6 such as in GPT-2 [Rad+19], mT5 [Xue+21], and BERT [Dev+18]. Using bytes leads to 256 tokens + special tokens, and a big reduction of parameters needed to handle distinct tokens. It also gives less advantage to English representation, which due to its usual overrepresentation in training data has subword tokens covering common words, whereas other languages have to use more tokens to represent the same words. Still, languages with non-Latin scripts are at a disadvantage, as they have to use more bytes to represent a character in UTF-8.

When ByT5 was trained in 2021, its large variants reached (at that time) state-of-the-art results on multilingual NLP benchmarks, at the cost of slower processing compared to subword tokenized models, but with more robustness to noise in input. Since then it has been made obsolete for most tasks by larger models (e.g. GPT-4 [Ope23], Mistral-7B[Jia+23]). Large byte-level models with significant improvements in capability have not been released, but there is interest in exploring alternative tokenization and encoding. For example Limisiewicz et al. [Lim+24], while building on ByT5 architecture, outperformed ByT5 on multilingual tasks by using a custom encoding informed by morphemes, which also improved efficiency and diminished language modeling gap across languages.

mC4

The mC4 dataset presented by Xue et al. [Xue+21] is a cleaned multilingual text dataset covering 101 languages. mC4 was created by scraping the web and then filtering out low-quality content and content in languages that are not well represented. The total size is 6.6×10^9 pages totaling 6.3×10^{12} tokens.[2]

[2]Subword tokens from [Xue+21], which can be estimated as 1–4 characters per token.

ByT5 training

ByT5 uses *Span corruption* as the pre-training task. A sequence of tokens from the training set is taken, and spans of n tokens (the mean is 20) are masked to sentinel tokens; weights of the network are adjusted to predict what tokens were in the masked spans. We can download the pre-trained model from the HuggingFace model Hub[3] and fine-tune it in a supervised way on specific tasks with the `transformers` Python package.

1.4 Seq2Seq decipherment

Deciphering substitution ciphers can be formulated as a sequence-to-sequence task. We can look at it as a translation where the source language is the ciphertext and the target language is the plaintext. Ciphers lack cultural confounders, that are present in other tasks such as translation, which makes ciphers a good toy task.

1.4.1 Substitution ciphers

Substitution ciphers are a simple and insecure method of encryption where we take the units of plaintext (letters) and according to a key and a cipher algorithm replace them with other units. They can often be broken by frequency analysis and brute force.

We use the following two ciphers for our experiments.

Vigenère cipher

The Vigenère polyalphabetic cipher uses a keyword $K = k_1 k_2 \dots k_n$ to shift each letter in a sentence $S = s_1 s_2 \dots s_m$. The rolling variant repeats the keyword to obtain a string of the same length as the plaintext $K_{ext} = \overbrace{k_1 k_2 \dots k_n k_1 k_2 \dots}^{m}$. To get the ciphertext we shift each letter by the corresponding letter in K_{ext}. The ciphertext $R = r_1 r_2 \dots r_m$ is composed from $r_i = s_i + k_i \mod 26$.

Enigma

The Enigma was a cipher machine used by the German army during WW2, its strength was based on the key shifting throughout the text and many possible settings. It is easily breakable with modern cryptanalysis. The original Enigma has 3 rotors altering the substitution, that rotate after each keypress and a plugboard

[3] `https://huggingface.co/google/byt5-small`

with additional letter swaps. When a key on the keyboard is pressed, current flows through the plugboard and rotors to the lampboard where the output letter lights up. [4] Interpunction and spaces are represented as 'X'. We use a simplified version without the plugboard.

1.4.2 Related work on decipherment modeling

NLP was interested in decipherment modeling even before the deep learning and Transformer era [NSN13]. It is often used as a toy problem for testing techniques and models. Deciphering real historical ciphers is also studied.

Greydanus [Gre17] trained RNNs to decipher Vigenère, Autokey, and Enigma with a prepended key. They trained models to generalize to decipher keys unseen in training. It demonstrates that decipherment modeling is a viable task for neural networks. Kambhatla, Bigvand, and Sarkar [KBS18] used an RNN language model to score candidate plaintexts for substitution ciphers and decreased beam search size compared to prior n-gram approaches. Aldarrab and May [AM21] explored the decipherment of substitution ciphers using a 40M parameter Transformer model trained from scratch with a representation of frequency analysis of the text. With beam search, they achieved a good performance.

[4] Enigma visualization: `https://observablehq.com/@tmcw/enigma-machine`.

Chapter 2

Interpreting machine learning

In this chapter, we describe the concept of interpretability, approaches to it, and related work on Transformers. With this context, we formulate and analyze our research question for this thesis.

2.1 Interpretability

Machine learning methods and especially neural networks are often viewed as black boxes, which work well in many tasks but the explanations for why are lacking. This poses a challenge as these models are increasingly used in high-stakes applications with concerns of deception, fairness and bias. There are various competing definitions of *interpretability* in the overlapping fields of interpretable and explainable AI (XAI), AI ethics, AI alignment, and AI safety. They revolve around the notion of humans understanding the causes of the decisions of the models or the ability to predict the model's outputs and behavior. This is a desirable property as advanced ML systems are deployed to all areas of life. For the context of this work, we use the term mostly as 'finding explanations for behavior and capability of models'. This is similar to the definition by Biran and Cotton [BC17].

There is a varying degree of interpretability in different ML models. Decision trees are comparatively interpretable due to explicitly representing the decision boundaries. On the other hand, deep neural networks are opaque due to their vast number of parameters and complex interactions. DNNs reached high popularity, so our understanding has to catch up even though it is a hard problem.

There are multiple categorizations of interpretability approaches in the literature and many methods operate on their boundaries. Firstly, extracting explanations can be directed at specific instances of the model's predictions such as asking "Why did the model recommend giving this person a loan"?, this

is called a local explanation. Or we can ask about the model's behavior in general, such as "What features are important for the model's predictions"? this is called a global explanation.

Secondly, a possible distinction is between *post hoc* and *developmental* interpretability. In post hoc interpretability, we try to find explanations for the model's behavior after it has been trained. In developmental interpretability, we design or influence models to have architectural properties that make them more interpretable or extract explanations from models during prediction.

Thirdly, a distinction is between *data-driven* and *mechanistic* interpretability. Data-driven interpretability methods find explanations by looking at the model's behavior while accepting that it is a black box. Mechanistic interpretability finds explanations by looking at the inner workings of models and decomposing them into more understandable components.

2.1.1 Feature attribution

One way to interpret a model is to analyze how it uses its input features for predictions. This is called *feature attribution*. It is an instance of a post hoc method, it can be both local and global. In some ML models, we can attribute straightforwardly, for example in Linear Regression, looking at weights and expected intervals of inputs is enough. For other ML models such as neural networks, we need to use more sophisticated methods.

2.1.2 SHAP

Shapley Additive exPlanations presented by Lundberg and Lee [LL17] are a feature importance framework for a prediction of any model. They were adopted within ML including NLP as reviewed by Mosca et al. [Mos+22] due to their rigorous theoretical grounding. Molnar [Mol22] explains the concept of Shapley Values from cooperative game theory.[1]

> A prediction can be explained by assuming that each feature value of the instance is a "player" in a game where the prediction is the payout. Shapley values — a method from coalitional game theory — tell us how to fairly distribute the "payout" among the features.

Computing Shapley values directly is computationally expensive, it requires considering all possible combinations of features. With SHAP, we efficiently compute Shapley values for features in each example and then visualize and aggregate them to get an idea about the global feature importance.

[1] `https://christophm.github.io/interpretable-ml-book/shapley.html`

The `shap` Python package[2] implements efficient Shapley value estimation or computation for various models, including neural networks and tree-based models. In sequence modeling, SHAP can also attribute which tokens in the input are important for the next token prediction.

2.1.3 Interpretability of Transformers

After introducing the notion of interpretability we can look into what it means to interpret a Transformer model and review related work. Transformer interpretability is currently a hot area of research as products hinge on them, they are used in higher-stakes applications and it is generally useful to know what a model has learned for debugging it. Additionally, we can ask how Transformer models tend to work in general and look for explanations transferrable to other models.

Transformer mechanistic interpretability focuses on what higher-level algorithms happen during the model inference. Nanda et al. [Nan+23] explained how small Transformers learn modular addition: first by memorizing training data and later explicitly representing the algorithm for it in its weights and forgetting memorization. They also reverse-engineered the algorithm used by the Transformer, showing it solves the task with a different algorithm than humans. Friedrich et al. [Fri+22] investigated modifying the Transformer to provide interactive explanations. Bricken et al. [Bri+23] proposed a technique for an automatic breakdown of a network to components that are easier to understand.

One of the large-scale goals of the field is automating trustworthy interpretability. Bills et al. [Bil+23] used the GPT-4 model [Ope23] to find explanations for all 307200 MLP neurons in the GPT-2 XL model [Rad+19]. They showed activations of a GPT-2 neuron on a text to GPT-4, which summarized what the activation pattern plausibly does, and then simulated the summary description activation pattern and compared how well the patterns match.

A component that is often analyzed in Transformer interpretability is the attention mechanism. Voita et al. [Voi+19] analyzed the attention heads in the Transformer model and found that they can be categorized into different types of behavior and some are redundant. Mareček and Rosa [MR19] investigated whether attention represents syntax and found evidence for it by extracting constituency trees from the attention activations.

In this work, we also use linguistic insights. We want to know if Transformer models rely on concepts similar to those that human linguists identified and if they do, which ones are important.

[2] `https://shap.readthedocs.io/`

2.2 Research question

To what extent do linguistic properties of input text explain the performance of a Transformer model working with that text?

2.2.1 Analysis

Let us operationalize the question to make it tractable with experiments and analyze pitfalls. When evaluating and interpreting models on NLP tasks such as translation, the space of possible properties and influences that explain the behavior is very large and their interactions are complex.

Task selection on which we study the behavior might also confound the evaluation with cultural and semantic biases that are hard to identify and obscure multilingual generalizability. To limit semantic and cultural connotations, we use a character-level task with a clear metric of what is correct. For that, we need a model operating on characters or bytes. As a byproduct, we get equalized tokenization across languages with Latin script, making the task easily multilingual. On the other hand, we sacrifice the utility of evaluating Transformers on realistic tasks.

Explanation by prediction

We formulate interpretability as a prediction problem. We have a black box model M that takes structured input X and produces structured output Y. If we wanted to predict the output of the model it would require solving the original problem that M is trained on. As a simplification, we can train a model N to predict a metric m on the output of M. We extract *properties* from the input X and use them as features for N to predict m.

$$N(\text{properties}(X)) \approx m(M(X))$$

If N reaches good performance in predicting m on M, we say that the `properties` function interprets the model M. Using SHAP on N further clarifies which properties are important.

M in our case is fine-tuned Transformer, Ns are simple ML models and `properties` are linguistic properties.

Assumptions

We make several assumptions to make the original question tractable in our experiments, and still say something useful.

- Prediction of performance is a reasonable proxy for assessing the importance of linguistic properties for the Transformer.
- Simple ML models (Linear Regression, MLP, RandomForest, XGBoost) suffice for regressing Transformer performance if linguistic properties provide enough signal.
- Character-level decipherment tasks help reduce reliance on semantics and cultural considerations. The task provides a clear metric of performance.
- Linguistic representations in ByT5 and subword-tokenized models are similar.
- The reliance on linguistic properties does not change discontinuously when scaling different Transformer models. We use a much smaller model than state-of-the-art.
- Fine-tuning on the decipherment task does not break the pre-trained linguistic representations which are those that are interesting.

After considering the simplifying assumptions we formulate the refined and actionable research question:

Can we find a post hoc data-driven global explanation for the performance of ByT5-small on the decipherment task with linguistic properties by predicting the performance with simple ML models?

Chapter 3

Experiments

In this chapter, we describe in detail our experiment pipeline for studying the relationship between the performance of the Transformer model and the linguistic properties of the input text.

3.1 Overview

Our experiment execution has 3 phases.

- Create an evaluation dataset with *automatically annotated linguistic properties (AALP)*.
- Fine-tune ByT5 models on the decipherment task.
- Analyze the relationship between fine-tuned ByT5 performance and the AALP.

Figure 3.1 shows the whole pipeline of the experiments.

3.2 Data

We work with training and evaluation data in English, German, and Czech. We download the data in the form of sentences from StatMT-Newscrawl[1] dataset [Koc+22]. We take the training set from the data labeled 2012 and the evaluation set from 2013 for each language.

[1] `https://data.statmt.org/news-crawl/`

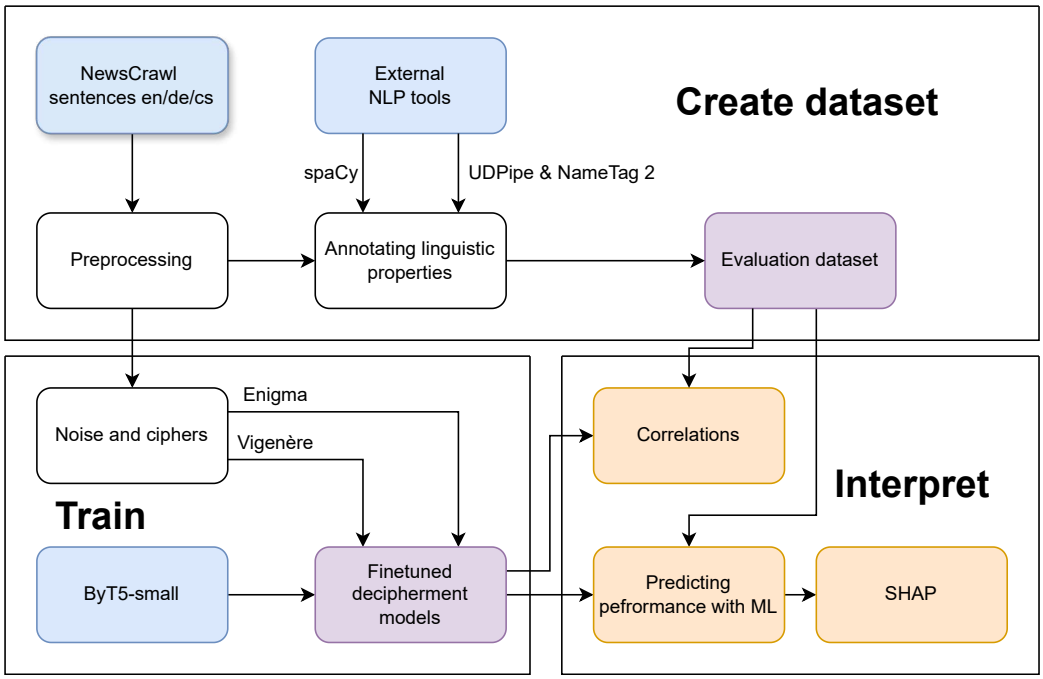


Figure 3.1 3 stages of experiment pipeline.

3.2.1 Preprocessing

To avoid too short or too truncated sentences, we filter the Newscrawl sentences to those that have 150–220 characters. We remove diacritics and punctuation marks from the sentences and lowercase them. We truncate each sentence (only letters now) to 200 characters, both in training and evaluation datasets.

The Newscrawl dataset in the languages we choose, is not sufficiently clean. In the evaluation dataset, outliers could break our analysis, therefore we filter out sentences in other languages with the FastText language identification model [Jou+16a; Jou+16b],[2] and remove sentences that contain too many numbers or special characters (which suggest they are not natural language but some tabular data e.g. sports match results) with static filters. We end up with 10^5 sentences (approx. 1.9M tokens) in the training set and 10^5 sentences in the evaluation set for each language.

3.2.2 Linguistic properties

To test the relationships between the performance of the Transformer and the linguistic properties of inputted sentences we have to first identify interesting

[2]Available in the `fastText` Python package.

properties. They should be easily computable with simple algorithms or available tools, so we can automatically annotate an evaluation dataset with nontrivial size.

First, we define a generic measure of how unusual a sentence is, with regards to a trait that is somehow distributed in a language. Possibilities for this measure include Cross-Entropy or KL-divergence D_{KL}, we use the *Jensen-Shannon Divergence (JSD)* because it has the nice property of being symmetric.

Definition 3 (Jensen-Shannon Divergence)**.**

$$JSD(P \parallel Q) = \frac{1}{2} D_{KL}\left(P \parallel \frac{1}{2}(P+Q)\right) + \frac{1}{2} D_{KL}\left(Q \parallel \frac{1}{2}(P+Q)\right)$$

where the Kullback-Leibler Divergence $D_{KL}(P \parallel Q)$ *is defined as:*

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

We identify 12 linguistic, neural language modeling-related and surface-level text properties to annotate and study.

JSD of character distributions

We want to measure how anomalous the sentence is in terms of character distribution. For the evaluation set X and sentences $S \in X$, we compute the *unigram character* distribution P_X of the whole set and P_S for each sentence. We annotate each sentence S with JSD$(P_X \parallel P_S)$. We also compute the *character bigram distribution* in the evaluation set Q_X and sentences Q_S and analogously annotate each sentence S with JSD$(Q_X \parallel Q_S)$. If the Transformer performance correlates with character distributions, it would suggest that for the decipherment task, it does something akin to frequency analysis.

Number of named entities

We count the number of named entities for each sentence in the evaluation set. A named entity is an object in the real world that is denoted with a proper name, i.e. a person, location, organization, etc. This property could show us if the Transformer anchors on words that substantially constrain the meaning of the rest of the sentence. To automatically extract entities we use for English: the spaCy small NLP pipeline model 3.7.1[3]; for German: the spaCy medium NLP pipeline model 3.7.0[4]; for Czech: the NameTag 2 model presented by Straková, Straka, and Hajič [SSH19] which is available as an API.[5]

[3] en_core_web_sm-3.7.1 model release

[4] de_core_news_md-3.7.0 model release

[5] `https://ufal.mff.cuni.cz/nametag/2`

Depth of the dependency tree

Publicly released NLP pipelines can parse sentences into syntactic dependency trees. We annotated sentences with their depth computed by traversing them from the root. We use the same spaCy pipeline for English and German as before, and for Czech the UDpipe model UT2.5 released by Straka and Straková [SS17] via the `spacy_udpipe` Python package. The depth is a proxy for the structural complexity of a sentence.

JSD of part of speech tags

JSD of *part of speech (PoS)* tags measures how typical or anomalous is the sentence structure. We use NLP models from spaCy and UDPipe to classify PoS tags for each word in evaluation set sentences, normalize their counts to a relative frequency distribution, and compute each sentence's PoS JSD from the relative frequency distribution of the whole evaluation set.

We also compute and annotate the JSD of PoS tag bigrams.

GPT-2 tokenizer tokens per character

We compute the average number of tokens per character in the sentence when tokenized by the GPT-2 BPE tokenizer [Rad+19]. BPE was conceived of as a compression algorithm so the density of tokens is a proxy for how common are the words used in a sentence.

GPT-2 perplexity

We measure the perplexity of the evaluation sentences consisting of tokens $w_1 \dots w_N$ with the GPT-2 [Rad+19] model p.

$$\text{Perplexity}(w_1 \dots w_N) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log p(w_i | w_{1,\dots,w_{i-1}})\right)$$

We can interpret perplexity as how surprised a language model is when it sees the sentence; surprised, in terms of what it learned during training, and how hard it is to understand the sentence.

Frequency of specific characters

JSD of character distribution might not be fine-grained enough so we add more very simple properties for interesting characters.

- *Space* is obviously interesting.

- The *letter 'e'* is the most common letter in English and German and 2nd in Czech.
- The *letters 'j', 'q', 'x', and 'z'* are the least common letters in English, and also rare in German and Czech, we sum them as one property for rare letters.

Number of characters in the sentence

The *Text Length* property tests if the Transformers understand they should generate the same number of tokens in the decipherment task.

3.3 Training

3.3.1 Decipherment tasks

We fine-tune *ByT5-small* in a translation setting. The input is an encrypted sentence and the output is the corresponding plaintext sentence.

Vigenère task

To get a training pair we generate a random 3-letter key and encrypt the sentence with the rolling key Vigenère cipher described in Section 1.4.1. We add noise to the task by randomly replacing 15% of the characters in the plaintext with another character. For the Transformer to successfully decipher the plaintext it has to somehow internally guess the key.

Constant Enigma task

We use the `py-enigma`[6] Python package implementation of the Enigma algorithm outlined in Section 1.4.1. We use a constant key (initial setting of the rotors) for encrypting each sentence. It would suffice for the Transformer to learn the mapping for each position, i.e. how much to shift each distinct letter. To force the Transformer to do something smarter, we add noise as before.

3.3.2 Training runs

We fine-tune for 40 epochs on 80000 sentences in pairs (encrypted, plaintext) with 10000 pairs as a development set. We train the final models on `PyTorch v2.0`[7] backend using the `HuggingFace Transformers v4.38` Trainer[8] with mostly

[6] `https://pypi.org/project/py-enigma/`
[7] `https://pytorch.org/get-started/pytorch-2.0/`
[8] `https://huggingface.co/docs/transformers/main_classes/trainer`

`Seq2SeqTrainer`'s default settings with AdamW optimizer. We set the initial learning rate to 2×10^{-3} and set a linear scheduler warmup for 20% of the training. We set the batch size and gradient accumulation to arrive at an effective batch size per device of 192, i.e. 192 sentences with cipher and plaintext. We save a checkpoint after every 5 epochs, so we obtain 8 checkpoints. We train on the AIC (Artificial Intelligence Cluster),[9] each Transformer with 4xNVIDIA 2080 GPUs. Each training run takes about 30 hours. We do this for each task and language to get 48 checkpoints from 6 runs: `Vig_EN`, `Vig_DE`, `Vig_CZ`, `Enigma_EN`, `Enigma_DE`, `Enigma_CZ`.

3.4 Analysis

We evaluate each checkpoint by inferencing the translation from ciphertext to plaintext for the evaluation dataset containing 100000 examples, for this, we employ 1xNVIDIA 2080 GPU (on AIC). We use greedy inference, picking the most probable token at each step of plaintext generation. For each checkpoint, we obtain candidate decipherment for each sentence in the evaluation set.

Definition 4 (Character Error Rate)**.** *For a reference sequence and candidate sequence.*

$$CER = \frac{S+D+I}{N}$$

where S, D, and I are the minimum number of substitutions, deletions, and insertions respectively to obtain the reference from the candidate sequence (also known as the Levensthein distance). N is the length in characters of the reference sentence.

The *Character Error Rate (CER)* comes from speech recognition evaluation, it is also useful for us because our Transformer models operate on variable length inputs and the task is character level.

From the generated decipherments and the original text, we compute the CER for each sentence in the evaluation set. We analyze the correlations between the fine-tuned Transformer's performance and the linguistic properties of the input and how they evolve during training.

3.4.1 Error prediction

We train MLP, Random Forest, XGBoost, and Ridge regression models, to predict the CER from linguistic properties as features. We use the `scikit-learn` and `XDGBoost` packages with model implementations, and train on 90000 examples

[9] `https://aic.ufal.mff.cuni.cz/`

from the evaluation set. We obtain 4 regression models for each fine-tuned checkpoint. We compute the MAE (Mean Absolute Error) and R^2 of the predictive models on a test set containing 10000 examples. We compare the predictive models to a baseline model that always predicts the mean CER of the train set.

3.4.2 SHAP feature importance

We assess the feature importance in the XDGBoost models by computing Shapley values for each example in the test set containing 10000 and plotting them in a summary plot, which visualizes feature contributions to the prediction. We use TreeSHAP [LEL18] implemented by the `shap` Python package to compute Shapley values efficiently.

Chapter 4

Results and discussion

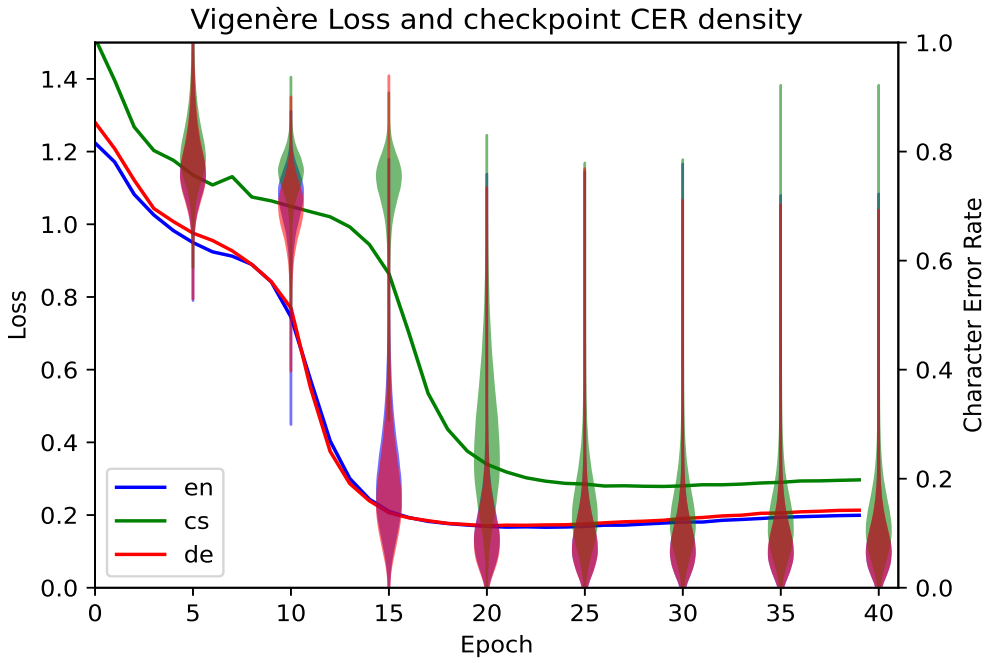
In this chapter, we present the results of the experiment pipeline described in Chapter 3. Then we discuss how the results answer our research question (Section 2.2) and relate to the interpretability of Transformers.

4.1 Training

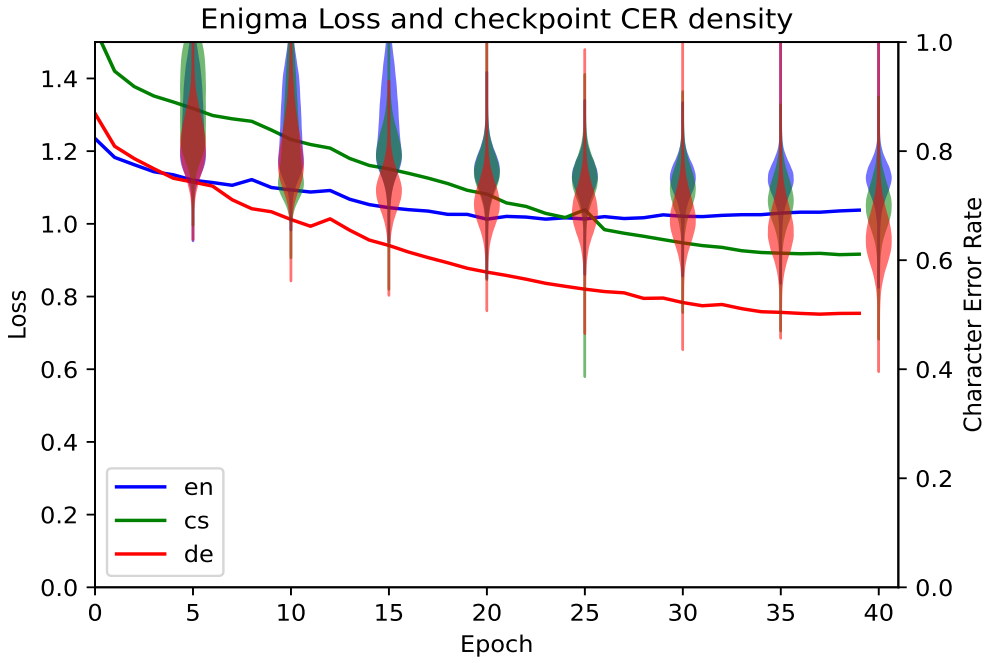
During the training of the Vigenère models the loss decreased as did the CER and ByT5 learned the decipherment task for all 3 languages. As shown in Figure 4.1a there was a marked dip in the loss corresponding to a rapid decrease in CER. We suppose that is the moment when the model cracked how to generally decipher the Vigenère cipher, it happened 5 epochs later in `Vig_CS` than in `Vig_EN` and `Vig_DE`. The models still committed errors at later epochs, due to the noise in input some parts of the sentences were not decipherable only from context.

The Enigma models for all languages did not converge on learning the task well, there was no corresponding loss dip in Figure 4.1b and the CER slowly decreased but remained high after 40 epochs and improvement halted, especially in English. The Enigma models learned to output words and sentences in the correct language, but they often did not correspond to the input. Enigma was surprisingly harder to learn for the Transformer even though the mapping with a constant setting is from a human standpoint more predictable.

The poor performance in Enigma models does not wholly prevent us from investigating whether the *Automatically annotated linguistic properties (AALP)* can explain the variance in the model's performance.



(a) Training loss and evaluation CER in `Vig_EN`, `Vig_DE`, `Vig_CS` checkpoints



(b) Training loss and evaluation CER in `Enigma_EN`, `Enigma_DE`, `Enigma_CS` checkpoints

Figure 4.1 Training loss and checkpoint evaluation.

4.2 Correlations

We computed the Pearson correlation coefficient between AALP and the CER for each checkpoint.

To ensure the correlations were real, we tested hypotheses that the correlations were zero with the Bonferroni correction for multiple testing, with a significance level of 0.01. There are in total 12 properties and we tested the significance in 6 models each with 8 checkpoints, totalling 576 tests. So the corrected significance level is $0.01/576 \approx 1.7 \cdot 10^{-5}$. At this level, we rejected most hypotheses that the correlations are zero for most of the properties.

Figure 4.2 shows that strong correlations disappeared as the training progressed. There, we visualize non-significant correlations with hollow points and significant correlations with filled points which are most of them. Vigenère models had generally smaller correlations than Engima models whose CER at early checkpoints strongly correlated with Text Length. `Vig_EN` retained correlation with GPT-2 metrics while CER generally decreased, which is not the case for `Vig_DE` and `Vig_CS`. Correlation evolution figures for Transformers fine-tuned on Czech and German are in the Appendix A.2.

We visualize the correlations also in a matrix for `Enigma_EN` in Figure 4.3. There are some medium-sized correlations between the features. The correlation matrices for other ciphers and languages are in the Appendix A.2.

Correlations were mostly small and the medium to large ones occurred before the ByT5 learned the decipherment tasks. There is not a clear pattern in the correlation evolutions that holds across languages and ciphers.

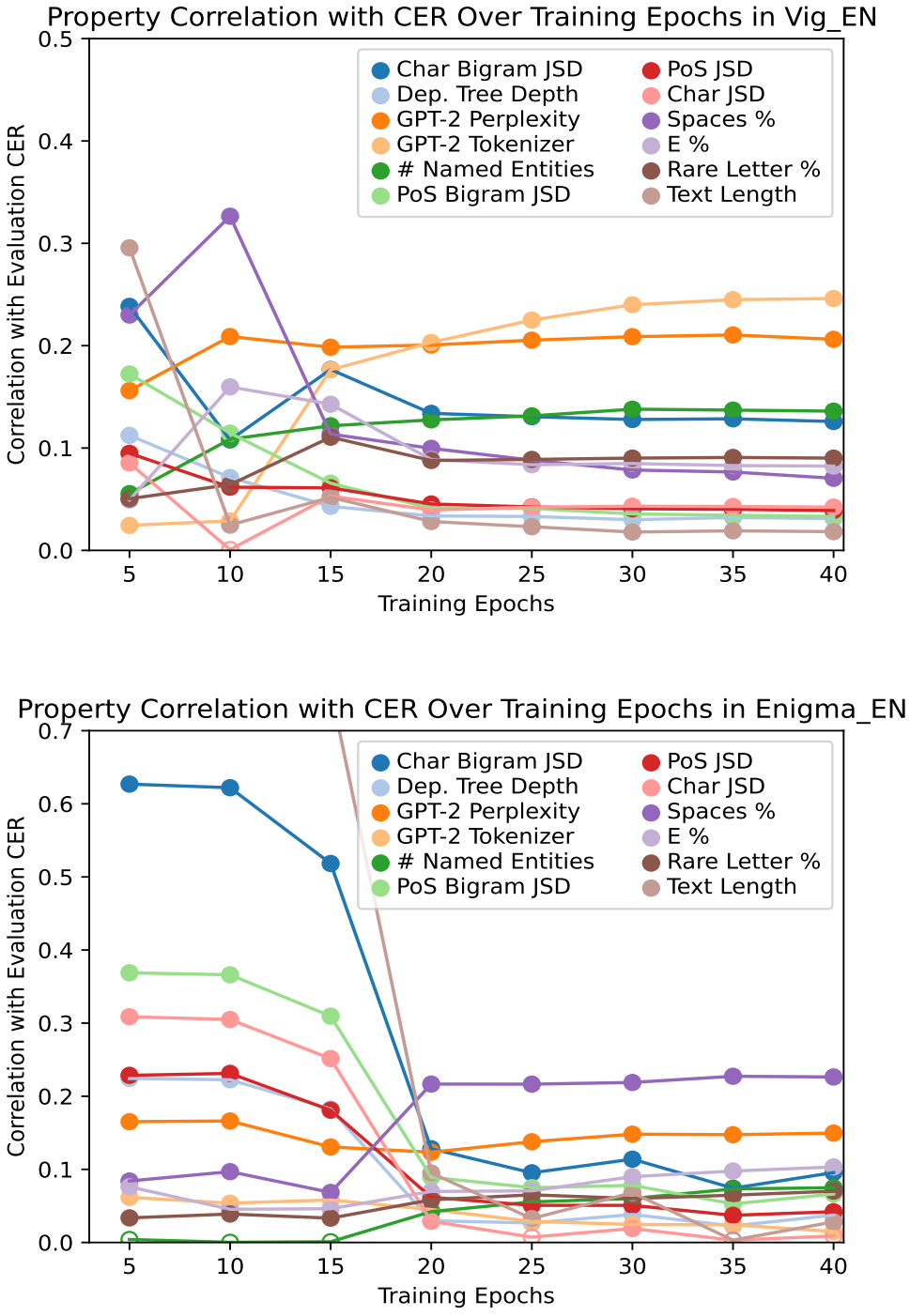


Figure 4.2 Evolution of correlation magnitude of CER with AALP; `Vig_EN` and `Enigma_EN`

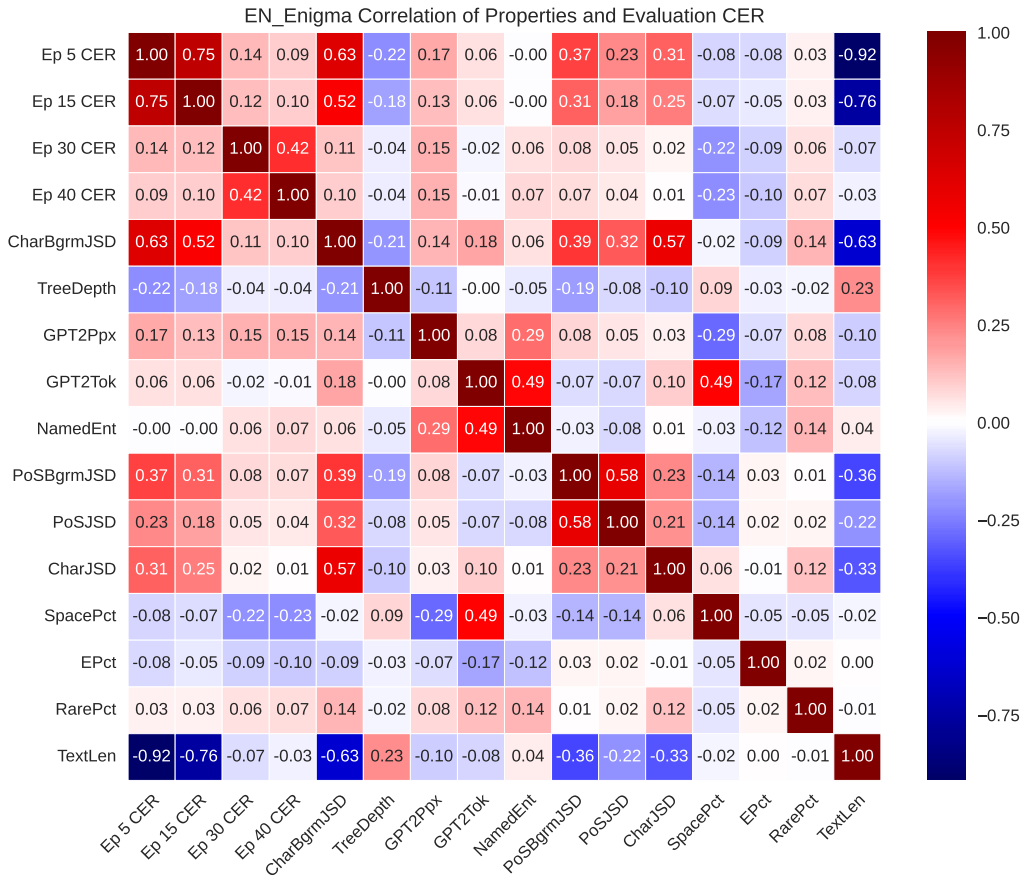


Figure 4.3 Correlation matrix of the AALP and the CER for `Enigma_EN`.

Model	MAE	R^2
Mean	0.044	-0.000
Ridge	0.041	0.155
XGBoost	0.041	0.138
RForest	0.041	0.140
MLP	0.041	0.131

(a) Epoch 5

Model	MAE	R^2
Mean	0.071	-0.000
Ridge	0.067	0.126
XGBoost	0.068	0.099
RForest	0.068	0.107
MLP	0.067	0.122

(b) Epoch 15

Model	MAE	R^2
Mean	0.038	-0.000
Ridge	0.035	0.137
XGBoost	0.036	0.115
RForest	0.036	0.119
MLP	0.036	0.126

(c) Epoch 40

Table 4.1 Predicting the CER from the AALP in `Vig_EN` checkpoints.

Model	MAE	R^2
Mean	0.062	-0.000
Ridge	0.022	0.860
XGBoost	0.019	0.887
RForest	0.019	0.887
MLP	0.021	0.871

(a) Epoch 5

Model	MAE	R^2
Mean	0.062	-0.000
Ridge	0.033	0.599
XGBoost	0.033	0.584
RForest	0.034	0.584
MLP	0.034	0.586

(b) Epoch 15

Model	MAE	R^2
Mean	0.025	-0.000
Ridge	0.024	0.082
XGBoost	0.024	0.060
RForest	0.024	0.065
MLP	0.025	-0.022

(c) Epoch 40

Table 4.2 Predicting the CER from the AALP in `Enigma_EN` checkpoints.

4.3 Performance prediction

Even though the correlations are small, they could jointly point to a structure in the data that can be used to predict fine-tuned ByT5's performance and thus explain it in the sense of Section 2.2.1.

We trained Linear Regression (Ridge), MLP, Random Forest, and XGBoost models to predict the CER at a checkpoint from the AALP. We show their performance predicting CER from AALP on the test set in Table 4.1 and Table 4.2 for `Vig_EN` and `Enigma_EN`. We compare them in terms of Mean Absolute Error (MAE) and R^2 and show a baseline 'Mean' model which always predicts the mean of training data. The interpretation is that we want low MAE and high R^2 (close to 1). Full results for all languages and ciphers are in the Appendix A.2.

For models of Vigenère Transformer performance, the R^2 was close to zero, which means that the models could not capture the causes for variance in the CER. For early epochs of the Enigma Transformers, the performance models were able to predict the CER well. Using these models provided improvement in the Mean Absolute Error (MAE) over the baseline prediction of the mean CER.

In later epochs, the models were not able to predict the CER well, which mirrors the decrease in correlations apparent in Figure 4.2. We see the decrease in the ability to predict from a decrease in the R^2 of the models as training progresses and MAE close to the baseline 'Mean' prediction.

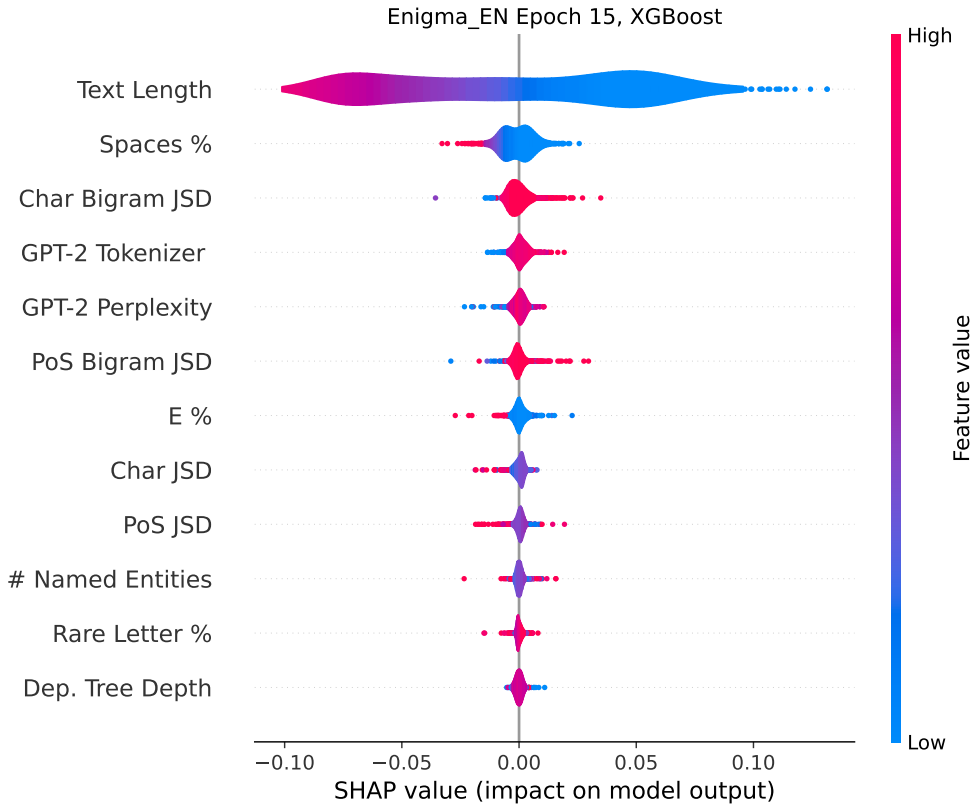


Figure 4.4 SHAP values for the XGBoost model predicting the CER for the `Enigma_EN` epoch 15.

Generally, RandomForest and XGBoost models were better than Linear Regression and MLP models, which sometimes even underperformed the 'Mean' model.

4.4 SHAP

We interpreted the XGBoost regressor with SHAP for each checkpoint. Due to the low R^2 of predictive models for Vigenère checkpoints, this did not give us any insights. The mean absolute contribution for each feature was close to 0.

For Enigma models, it confirmed the relationship from correlations: the Text Length is important in the early epochs. In later epochs, the variance in CER was again not well explained by predicting from the properties, therefore SHAP did not help us extract more insight and mean absolute feature contributions were close to 0. Figure 4.4 illustrates that for the `Enigma_EN` at epoch 15 Text Length had a clear impact on CER prediction, higher Text Length led to lower error rate,

but the mean absolute impact of other properties was close to zero. We do not include figures for other languages and ciphers. For early Enigma epochs, they are similar, and for others, they only show a lot of points hugging 0 impact.

4.5 Discussion

The model can learn the Vigenère cipher for all tested languages quite well. Czech and German are harder to fine-tune for, than English, which confirms existing research that pre-training on disproportionately English internet text provides an advantage and removing diacritics hurts performance in languages that use them. There is a moment in the Vigenère task training where the model groks how to decipher and the CER drops rapidly. This is not the case for the Enigma task, this is a bit surprising given that our Enigma has only 1 cipher setting and Vigenère has 26^3, but we did not pursue this lead further.

The correlation coefficients of properties to the CER would be considered small in behavioral sciences (Cohen [Coh88] denotes small as less than 0.3, and medium as less than 0.5). Lack of consistency across languages and ciphers first alerts us that the result is negative in terms of linguistic properties explaining the model's performance.

We see consistently, that the AALPs do not help explain the model's performance using prediction as training progresses (except the uninteresting Text Length in early Enigma checkpoints). In other words, the performance of the models seems to be robust to variance in the properties we measured. We suppose most of the variance appears from the noise we added to the task as described in Section 3.3.1.

The most useful insight that can be extracted from our results is that in tasks that depend on a specific length of outputs, it is a good idea to debug it; it took Enigma models many epochs to learn this dependency. The overall results provide evidence against our hypothesis that the performance of Transformers can be explained by easily measurable linguistic properties of the text they work with. We also have to question the assumption from Section 2.2.1 that the performance prediction methodology makes sense.

The experiments with decipherment provide evidence that data-driven approaches have a limited capacity to find good explanations. However, it is not conclusive, we might have just failed to identify impactful properties of text and or used an overly convoluted task. It would be easy to adapt this pipeline to other models, languages or ciphers but it does not seem like a productive area of research. The recent shift of interest to mechanistic methods may prove more useful.

Conclusion

We have aimed to understand the relationship between the performance of Transformers and the linguistic properties of their inputs. We have implemented a character-level decipherment task with Vigenère and Enigma ciphers. We fine-tuned the ByT5-small models on the task with English, German and Czech sentences. On data obtained from evaluating this task, we have created an analysis pipeline that annotates linguistic properties of input sentences with spaCy pipelines, and other tools (UDPipe, NameTag 2) and then correlate them with the models' performance at 8 checkpoints.

We have analyzed how these correlations evolve during training, they are small but significant. We have predicted the decipherment error rates from the properties via ML models and assessed feature importance in their predictions using SHAP. We have found that the performance of the Transformers on the decipherment task cannot be predicted well by the automatically annotated linguistic properties, therefore neither the SHAP analysis is informative.

A key limitation of the results is that since ByT5 [Xue+22] no better byte or character-level models have been released. It is unclear if our linguistic conclusions hold for subword-tokenized models and they have architectural issues with the character-level decipherment task. The analysis relies on a lot of steps that could introduce noise. The results shed limited light on the interpretability of Transformers in general, although they provide evidence against the importance of linguistic properties for Transformer performance.

Other tasks, properties and models can use the data-driven methodology we described: measuring properties of textual inputs, finding correlations on outputs or task performance, then making predictive models, and interpreting those. However, our results are discouraging. Mechanistic approaches seem more promising for the general quest to understand Transformers.

Future work can validate the result on other large byte-level models if they emerge or with more resources (more data, bigger ByT5 variants). A more systematic evaluation of the potential and limitations of the analysis pipeline would be desirable before applying it to other tasks and models.

Bibliography

[AM21] Nada Aldarrab and Jonathan May. "Can Sequence-to-Sequence Models Crack Substitution Ciphers?" In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*. Ed. by Chengqing Zong et al. Association for Computational Linguistics, 2021, pp. 7226–7235. DOI: 10.18653/V1/2021.ACL-LONG.561. URL: https://doi.org/10.18653/v1/2021.acl-long.561.

[BC17] Or Biran and Courtenay V. Cotton. "Explanation and Justification in Machine Learning : A Survey Or". In: 2017. URL: https://api.semanticscholar.org/CorpusID:3911355.

[Bil+23] Steven Bills et al. "Language Models Can Explain Neurons in Language Models". In: (2023). URL: https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html.

[BKH16] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *CoRR* abs/1607.06450 (2016). arXiv: 1607.06450. URL: http://arxiv.org/abs/1607.06450.

[Bri+23] Trenton Bricken et al. "Towards Monosemanticity: Decomposing Language Models With Dictionary Learning". In: *Transformer Circuits Thread* (2023). URL: https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[Coh88] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. 2nd ed. Routledge, 1988. ISBN: 9780805802832. DOI: 10.4324/9780203771587.

[Dev+18] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[Fri+22] Felix Friedrich et al. “Interactively Providing Explanations for Transformer Language Models”. In: *HHAI 2022: Augmenting Human Intellect - Proceedings of the First International Conference on Hybrid Human-Artificial Intelligence, Amsterdam, The Netherlands, 13-17 June 2022*. Ed. by Stefan Schlobach, María Pérez-Ortiz, and Myrthe Tielman. Vol. 354. Frontiers in Artificial Intelligence and Applications. IOS Press, 2022, pp. 285–287. DOI: 10.3233/FAIA220218. URL: https://doi.org/10.3233/FAIA220218.

[Gre17] Sam Greydanus. “Learning the Enigma with Recurrent Neural Networks”. In: *CoRR* abs/1708.07576 (2017). arXiv: 1708.07576. URL: http://arxiv.org/abs/1708.07576.

[He+15] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[Jia+23] Albert Q. Jiang et al. “Mistral 7B”. In: *CoRR* abs/2310.06825 (2023). DOI: 10.48550/ARXIV.2310.06825. arXiv: 2310.06825. URL: https://doi.org/10.48550/arXiv.2310.06825.

[Jou+16a] Armand Joulin et al. “Bag of Tricks for Efficient Text Classification”. In: *CoRR* abs/1607.01759 (2016). arXiv: 1607.01759. URL: http://arxiv.org/abs/1607.01759.

[Jou+16b] Armand Joulin et al. “FastText.zip: Compressing text classification models”. In: *CoRR* abs/1612.03651 (2016). arXiv: 1612.03651. URL: http://arxiv.org/abs/1612.03651.

[Kap+20] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. In: *CoRR* abs/2001.08361 (2020). arXiv: 2001.08361. URL: https://arxiv.org/abs/2001.08361.

[KB15] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[KBS18] Nishant Kambhatla, Anahita Mansouri Bigvand, and Anoop Sarkar. “Decipherment of Substitution Ciphers with Neural Language Models”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*. Ed. by Ellen Riloff et al. Association for Computational Linguistics, 2018, pp. 869–874. DOI: 10.18653/V1/D18-1102. URL: https://doi.org/10.18653/v1/d18-1102.

[Koc+22] Tom Kocmi et al. “Findings of the 2022 Conference on Machine Translation (WMT22)”. In: *Proceedings of the Seventh Conference on Machine Translation (WMT)*. Ed. by Philipp Koehn et al. Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics, Dec. 2022, pp. 1–45. URL: https://aclanthology.org/2022.wmt-1.1.

[LEL18] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. “Consistent Individualized Feature Attribution for Tree Ensembles”. In: *CoRR* abs/1802.03888 (2018). arXiv: 1802.03888. URL: http://arxiv.org/abs/1802.03888.

[Lim+24] Tomasz Limisiewicz et al. “MYTE: Morphology-Driven Byte Encoding for Better and Fairer Multilingual Language Modeling”. In: *CoRR* abs/2403.10691 (2024). DOI: 10.48550/ARXIV.2403.10691. arXiv: 2403.10691. URL: https://doi.org/10.48550/arXiv.2403.10691.

[LL17] Scott M. Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 4765–4774. URL: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[Mol22] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. 2022. ISBN: 9798411463330. URL: https://christophm.github.io/interpretable-ml-book.

[Mos+22] Edoardo Mosca et al. “SHAP-Based Explanation Methods: A Review for NLP Interpretability”. In: *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*. Ed. by Nicoletta Calzolari et al. International Committee on Computational Linguistics, 2022, pp. 4593–4603. URL: https://aclanthology.org/2022.coling-1.406.

[MR19] David Mareček and Rudolf Rosa. “From Balustrades to Pierre Vinken: Looking for Syntax in Transformer Self-Attentions”. In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP at ACL 2019, Florence, Italy, August 1, 2019*. Ed. by Tal Linzen et al. Association for Computational

Linguistics, 2019, pp. 263–275. DOI: 10.18653/V1/W19-4827. URL: https://doi.org/10.18653/v1/W19-4827.

[Nan+23] Neel Nanda et al. "Progress measures for grokking via mechanistic interpretability". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/pdf?id=9XFSbDPmdW.

[NSN13] Malte Nuhn, Julian Schamper, and Hermann Ney. "Beam Search for Solving Substitution Ciphers". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*. The Association for Computer Linguistics, 2013, pp. 1568–1576. URL: https://aclanthology.org/P13-1154/.

[Ope23] OpenAI. "GPT-4 Technical Report". In: *CoRR* abs/2303.08774 (2023). DOI: 10.48550/ARXIV.2303.08774. arXiv: 2303.08774. URL: https://doi.org/10.48550/arXiv.2303.08774.

[Rad+19] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: 2019. URL: https://api.semanticscholar.org/CorpusID:160025533.

[Raf+19] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *CoRR* abs/1910.10683 (2019). arXiv: 1910.10683. URL: http://arxiv.org/abs/1910.10683.

[Sha20] Noam Shazeer. "GLU Variants Improve Transformer". In: *CoRR* abs/2002.05202 (2020). arXiv: 2002.05202. URL: https://arxiv.org/abs/2002.05202.

[Sri+14] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313. URL: https://dl.acm.org/doi/10.5555/2627435.2670313.

[SS17] Milan Straka and Jana Straková. "Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe". In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, August 3-4, 2017*. Ed. by Jan Hajic and Dan Zeman. Association for Computational Linguistics, 2017, pp. 88–99. DOI: 10.18653/V1/K17-3009. URL: https://doi.org/10.18653/v1/K17-3009.

[SSH19] Jana Straková, Milan Straka, and Jan Hajič. “Neural Architectures for Nested NER through Linearization”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by Anna Korhonen, David R. Traum, and Lluís Màrquez. Association for Computational Linguistics, 2019, pp. 5326–5331. DOI: 10.18653/V1/P19-1527. URL: https://doi.org/10.18653/v1/p19-1527.

[Vas+17] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[Voi+19] Elena Voita et al. “Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned”. In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by Anna Korhonen, David R. Traum, and Lluís Màrquez. Association for Computational Linguistics, 2019, pp. 5797–5808. DOI: 10.18653/V1/P19-1580. URL: https://doi.org/10.18653/v1/p19-1580.

[Xue+21] Linting Xue et al. “mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Ed. by Kristina Toutanova et al. Association for Computational Linguistics, 2021, pp. 483–498. DOI: 10.18653/V1/2021.NAACL-MAIN.41. URL: https://doi.org/10.18653/v1/2021.naacl-main.41.

[Xue+22] Linting Xue et al. “ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models”. In: *Trans. Assoc. Comput. Linguistics* 10 (2022), pp. 291–306. DOI: 10.1162/TACL_A_00461. URL: https://doi.org/10.1162/tacl%5C_a%5C_00461.

Appendix A

Reproducing experiments and all results

A.1 Reproducing experiments

The attached scripts[1] contain code for training, annotating properties, analysis and visualizations.

To run the code install the dependencies from `requirements.txt`, notable packages are `transformers, torch, spacy, shap, pandas, matplotib`.

- The `src/` directory contains code used across multiple training runs and annotations.
- The `reproducible/` directory contains Jupyter notebooks for training the Transformers models on decipherment tasks. Models we analyzed were trained with scripts 21–26.
- The `data/` directory contains notebooks for creating evaluation datasets. Measure each property, inference evaluation decipherments with checkpoints and merge resulting `.csv` files.
- The `analysis/` directory contains notebooks for predictive and SHAP analysis and visualizations.

We ran the notebooks on the AIC[2] cluster with GPUs managed with Slurm[3] via `run_notebook.sh`, `run_notebook4gpu.sh` or `run_cpunotebook.sh` scripts. Any cluster with Slurm should work very similarly and using notebooks locally with access to GPUs is even easier.

[1] Available from `https://github.com/JanProvaznik/enigma-transformed`

[2] `https://aic.ufal.mff.cuni.cz/`

[3] `https://slurm.schedmd.com/`

A.2 All figures and tables

Correlation evolution

Evolution of character error rate (CER) correlations with automatically annotated linguistic properties (AALP) corresponding to Section 4.2 Figure A.1 for German models and Figure A.2 for Czech.

Correlation matrices

Correlation matrices of the AALP and the CER corresponding to Section 4.2. Figure A.3 for English models, Figure A.4 for German models and Figure A.5 for Czech models.

Predictive models evaluation

Evaluation of predictive models in terms of mean absolute error (MAE) and R^2 corresponding to Section 4.3. Table A.1 for English models, Table A.2 for German models and Table A.3 for Czech models.

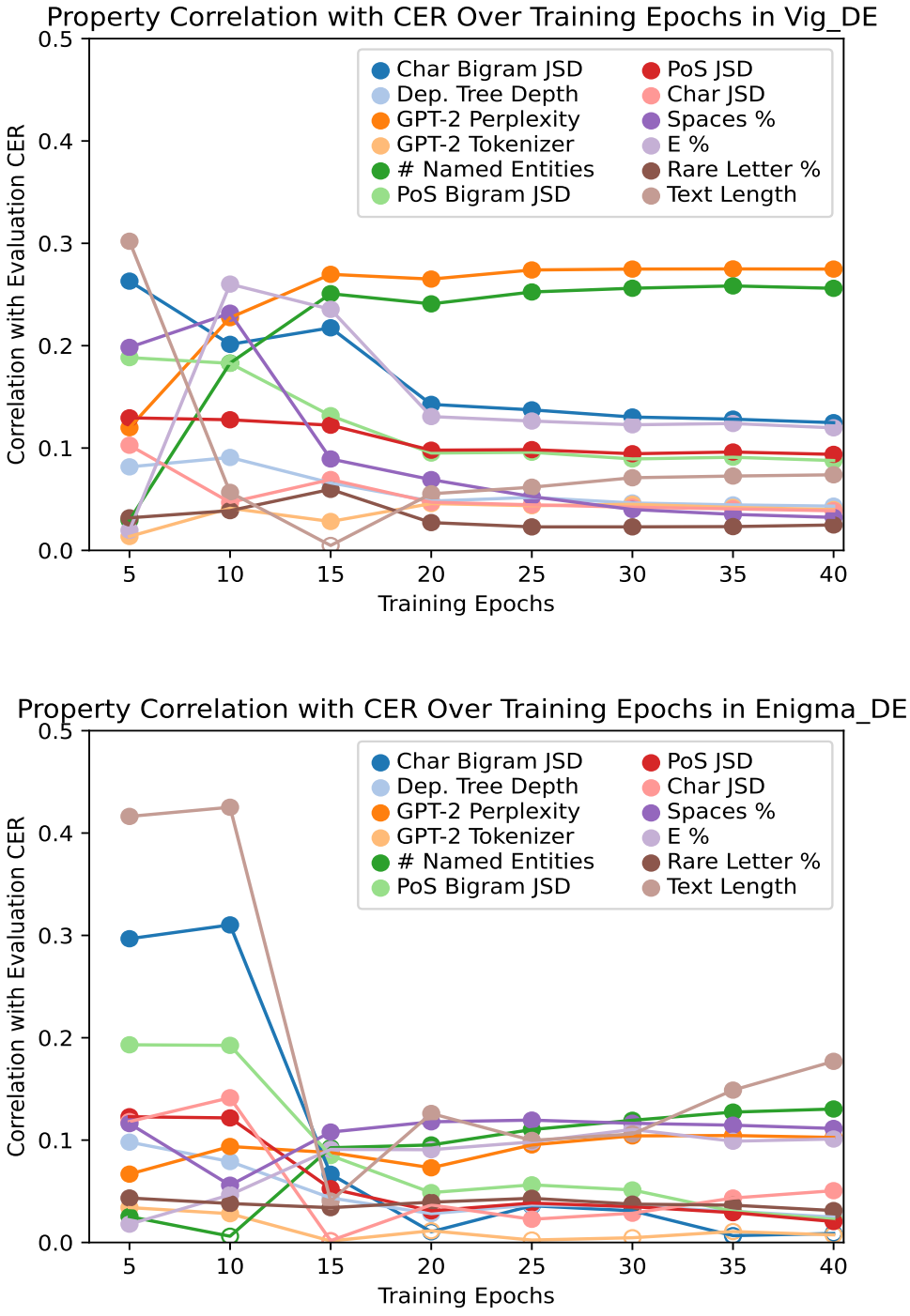


Figure A.1 Evolution of correlation magnitude of CER with AALP, `Vig_DE` and `Enigma_DE`.

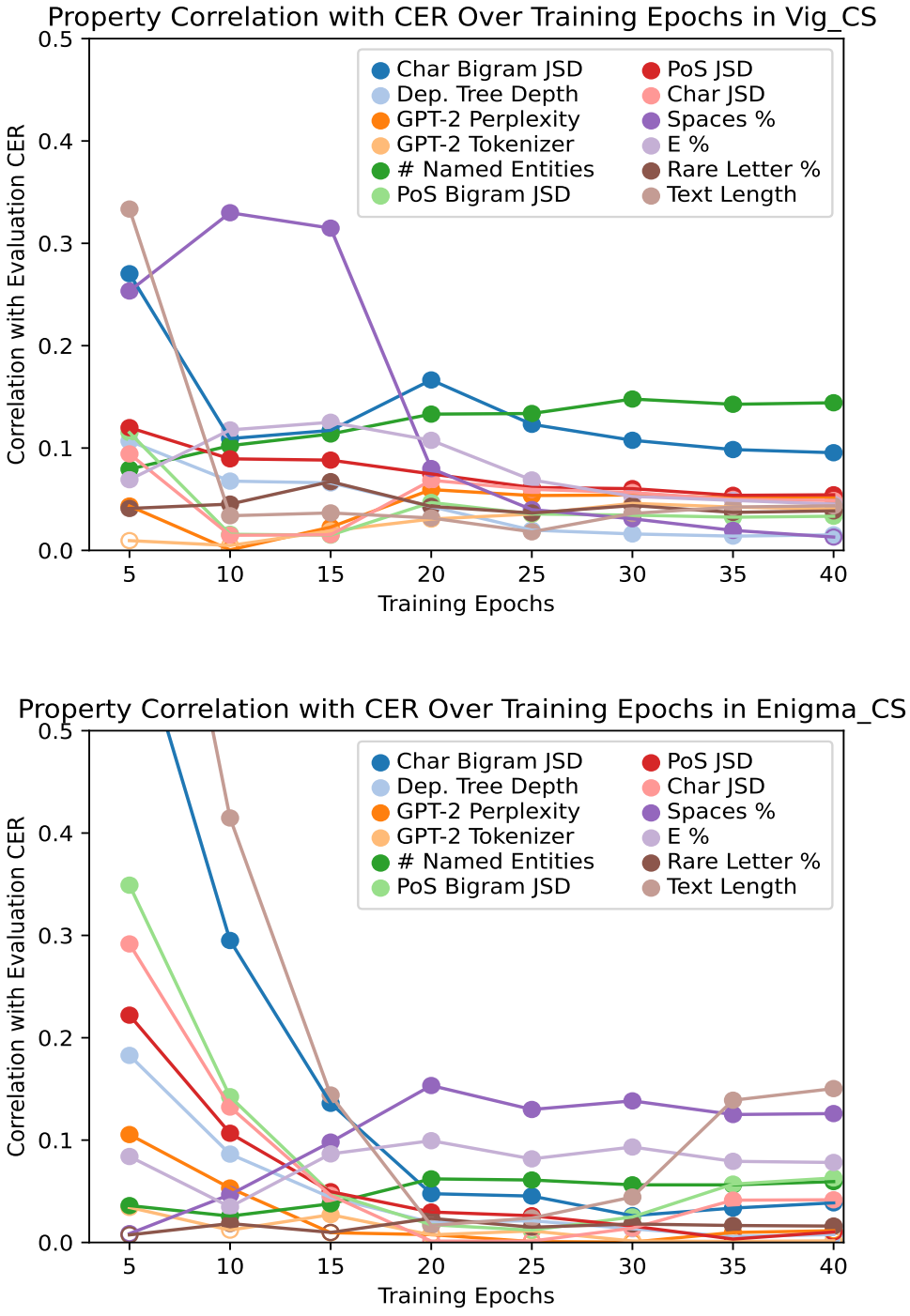


Figure A.2 Evolution of correlation magnitude of CER with AALP, `Vig_CS` and `Enigma_CS`.

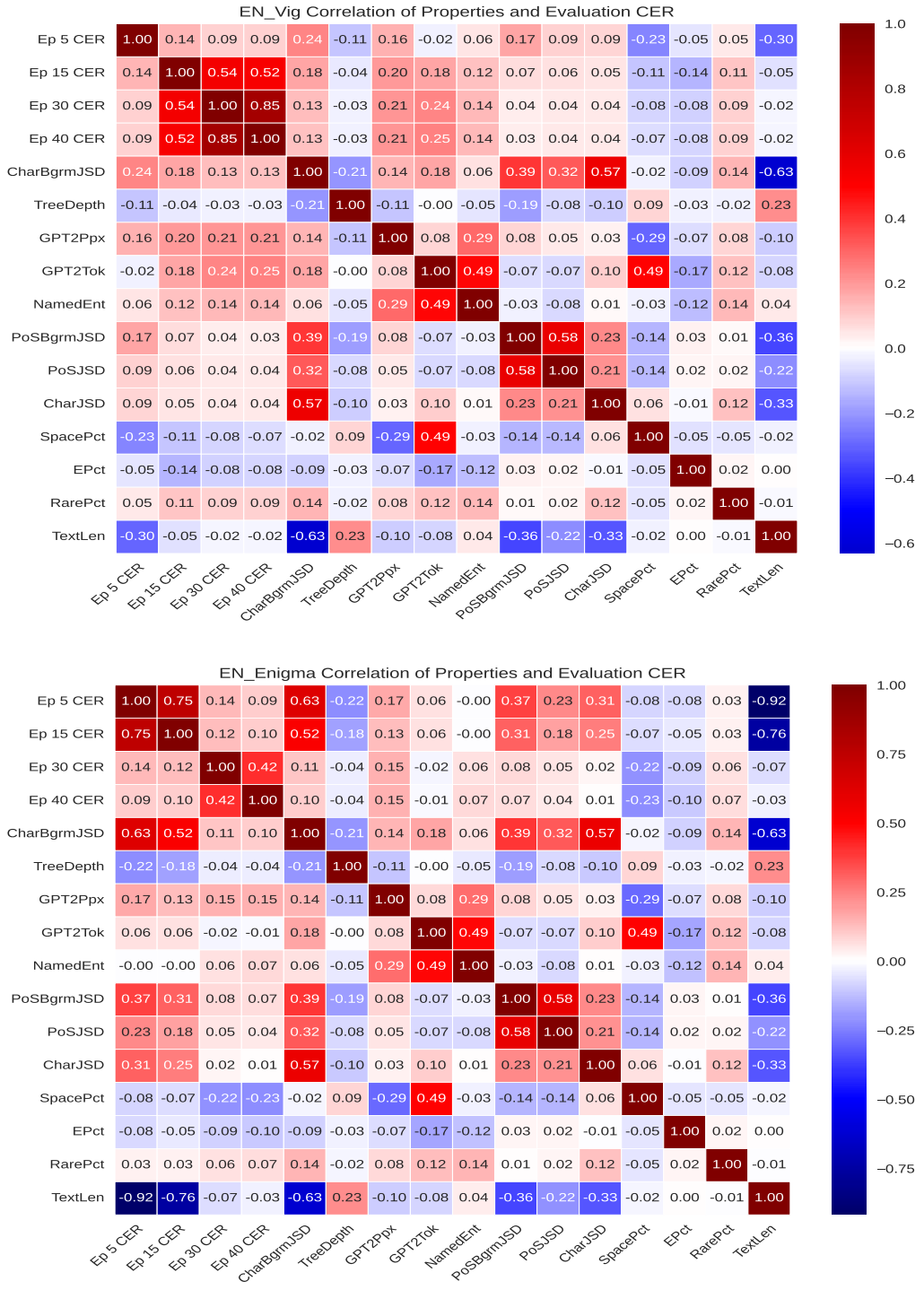


Figure A.3 Correlation matrix of the AALP and the CER in `Vig_EN` and `Enigma_EN`.

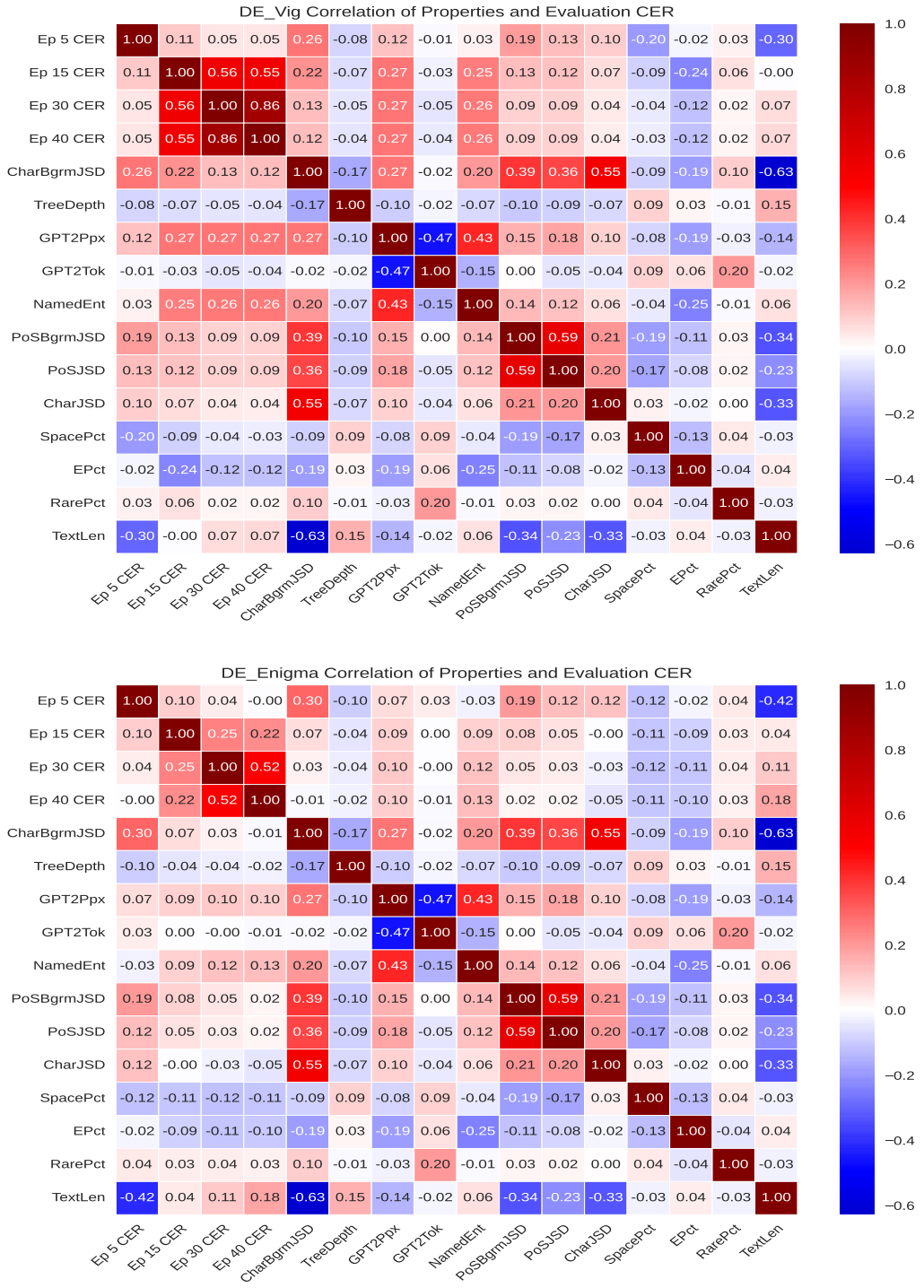


Figure A.4 Correlation matrix of the AALP and the CER in `Vig_DE` and `Enigma_DE`.

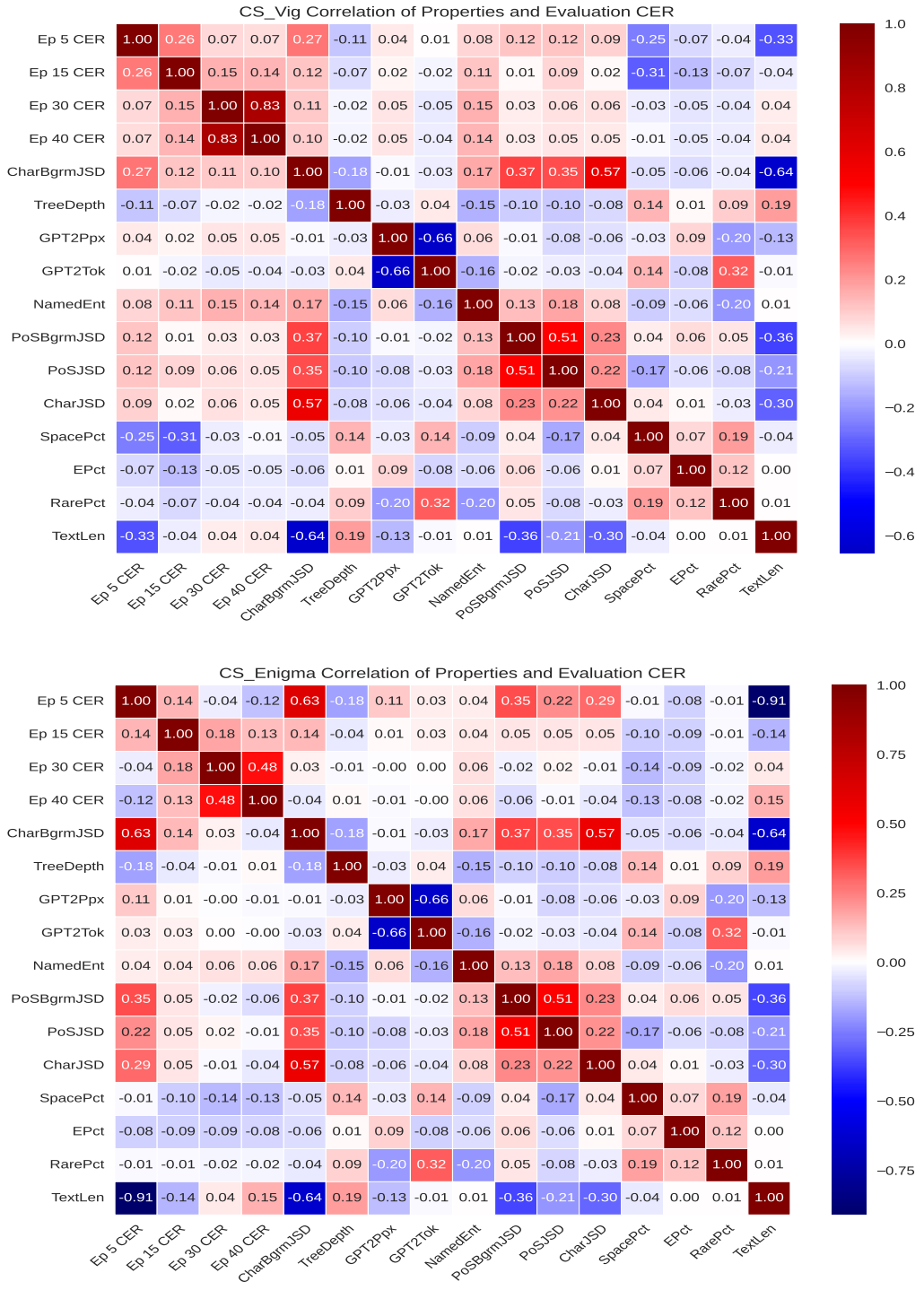


Figure A.5 Correlation matrix of the AALP and the CER in `Vig_CS` and `Enigma_CS`.

Vigenère

Model	MAE	R^2
Mean	0.044	-0.000
Ridge	0.041	0.155
XGBoost	0.041	0.138
RForest	0.041	0.140
MLP	0.041	0.131

(a) Epoch 5

Model	MAE	R^2
Mean	0.037	-0.000
Ridge	0.034	0.165
XGBoost	0.034	0.143
RForest	0.034	0.142
MLP	0.034	0.121

(b) Epoch 10

Model	MAE	R^2
Mean	0.071	-0.000
Ridge	0.067	0.126
XGBoost	0.068	0.099
RForest	0.068	0.107
MLP	0.067	0.122

(c) Epoch 15

Model	MAE	R^2
Mean	0.046	-0.000
Ridge	0.043	0.124
XGBoost	0.044	0.107
RForest	0.044	0.107
MLP	0.043	0.122

(d) Epoch 20

Model	MAE	R^2
Mean	0.039	-0.000
Ridge	0.036	0.140
XGBoost	0.037	0.122
RForest	0.037	0.121
MLP	0.036	0.127

(e) Epoch 30

Model	MAE	R^2
Mean	0.038	-0.000
Ridge	0.035	0.137
XGBoost	0.036	0.115
RForest	0.036	0.119
MLP	0.036	0.126

(f) Epoch 40

Enigma

Model	MAE	R^2
Mean	0.062	-0.000
Ridge	0.022	0.860
XGBoost	0.019	0.887
RForest	0.019	0.887
MLP	0.021	0.871

(g) Epoch 5

Model	MAE	R^2
Mean	0.063	-0.000
Ridge	0.022	0.859
XGBoost	0.020	0.885
RForest	0.020	0.883
MLP	0.020	0.880

(h) Epoch 10

Model	MAE	R^2
Mean	0.062	-0.000
Ridge	0.033	0.599
XGBoost	0.033	0.584
RForest	0.034	0.584
MLP	0.034	0.586

(i) Epoch 15

Model	MAE	R^2
Mean	0.028	-0.000
Ridge	0.027	0.075
XGBoost	0.027	0.089
RForest	0.027	0.092
MLP	0.027	0.066

(j) Epoch 20

Model	MAE	R^2
Mean	0.025	-0.000
Ridge	0.024	0.072
XGBoost	0.024	0.040
RForest	0.024	0.048
MLP	0.025	-0.001

(k) Epoch 30

Model	MAE	R^2
Mean	0.025	-0.000
Ridge	0.024	0.082
XGBoost	0.024	0.060
RForest	0.024	0.065
MLP	0.025	-0.022

(l) Epoch 40

Table A.1 Predicting CER from AALP with ML models, English

Vigenère

Model	MAE	R^2
Mean	0.049	-0.000
Ridge	0.045	0.155
XGBoost	0.046	0.127
RForest	0.046	0.132
MLP	0.046	0.090

(a) Epoch 5

Model	MAE	R^2
Mean	0.039	-0.000
Ridge	0.035	0.184
XGBoost	0.035	0.163
RForest	0.035	0.159
MLP	0.036	0.149

(b) Epoch 10

Model	MAE	R^2
Mean	0.067	-0.000
Ridge	0.062	0.165
XGBoost	0.062	0.163
RForest	0.062	0.166
MLP	0.062	0.166

(c) Epoch 15

Model	MAE	R^2
Mean	0.044	-0.000
Ridge	0.041	0.121
XGBoost	0.041	0.129
RForest	0.041	0.132
MLP	0.041	0.143

(d) Epoch 20

Model	MAE	R^2
Mean	0.039	-0.000
Ridge	0.036	0.122
XGBoost	0.036	0.136
RForest	0.036	0.137
MLP	0.036	0.142

(e) Epoch 30

Model	MAE	R^2
Mean	0.038	-0.000
Ridge	0.035	0.130
XGBoost	0.035	0.137
RForest	0.035	0.139
MLP	0.035	0.152

(f) Epoch 40

Enigma

Model	MAE	R^2
Mean	0.040	-0.000
Ridge	0.035	0.203
XGBoost	0.033	0.266
RForest	0.033	0.265
MLP	0.034	0.202

(g) Epoch 5

Model	MAE	R^2
Mean	0.047	-0.000
Ridge	0.041	0.196
XGBoost	0.041	0.186
RForest	0.041	0.195
MLP	0.042	0.173

(h) Epoch 10

Model	MAE	R^2
Mean	0.030	-0.000
Ridge	0.029	0.046
XGBoost	0.029	0.042
RForest	0.029	0.037
MLP	0.029	0.014

(i) Epoch 15

Model	MAE	R^2
Mean	0.031	-0.000
Ridge	0.030	0.058
XGBoost	0.030	0.075
RForest	0.030	0.077
MLP	0.031	0.011

(j) Epoch 20

Model	MAE	R^2
Mean	0.034	-0.000
Ridge	0.033	0.060
XGBoost	0.033	0.066
RForest	0.033	0.069
MLP	0.036	-0.127

(k) Epoch 30

Model	MAE	R^2
Mean	0.037	-0.000
Ridge	0.035	0.075
XGBoost	0.035	0.067
RForest	0.036	0.067
MLP	0.036	0.039

(l) Epoch 40

Table A.2 Predicting CER from AALP with ML models, German

Vigenère

Model	MAE	R^2
Mean	0.046	-0.001
Ridge	0.041	0.199
XGBoost	0.042	0.181
RForest	0.042	0.185
MLP	0.042	0.185

(a) Epoch 5

Model	MAE	R^2
Mean	0.029	-0.000
Ridge	0.026	0.142
XGBoost	0.027	0.123
RForest	0.027	0.123
MLP	0.031	-0.102

(b) Epoch 10

Model	MAE	R^2
Mean	0.036	-0.000
Ridge	0.034	0.127
XGBoost	0.034	0.104
RForest	0.034	0.103
MLP	0.036	0.033

(c) Epoch 15

Model	MAE	R^2
Mean	0.092	-0.000
Ridge	0.089	0.075
XGBoost	0.090	0.058
RForest	0.090	0.065
MLP	0.089	0.075

(d) Epoch 20

Model	MAE	R^2
Mean	0.052	-0.000
Ridge	0.050	0.057
XGBoost	0.050	0.051
RForest	0.051	0.048
MLP	0.050	0.062

(e) Epoch 30

Model	MAE	R^2
Mean	0.048	-0.000
Ridge	0.047	0.052
XGBoost	0.047	0.052
RForest	0.047	0.048
MLP	0.047	0.067

(f) Epoch 40

Enigma

Model	MAE	R^2
Mean	0.064	-0.000
Ridge	0.024	0.839
XGBoost	0.022	0.868
RForest	0.022	0.867
MLP	0.024	0.846

(g) Epoch 5

Model	MAE	R^2
Mean	0.071	-0.000
Ridge	0.065	0.173
XGBoost	0.066	0.141
RForest	0.066	0.157
MLP	0.066	0.150

(h) Epoch 10

Model	MAE	R^2
Mean	0.033	-0.000
Ridge	0.032	0.041
XGBoost	0.031	0.072
RForest	0.031	0.078
MLP	0.031	0.057

(i) Epoch 15

Model	MAE	R^2
Mean	0.029	-0.000
Ridge	0.028	0.035
XGBoost	0.028	0.038
RForest	0.028	0.051
MLP	0.029	-0.021

(j) Epoch 20

Model	MAE	R^2
Mean	0.031	-0.000
Ridge	0.030	0.036
XGBoost	0.030	0.058
RForest	0.030	0.061
MLP	0.030	0.012

(k) Epoch 30

Model	MAE	R^2
Mean	0.032	-0.000
Ridge	0.031	0.048
XGBoost	0.031	0.037
RForest	0.031	0.044
MLP	0.032	0.016

(l) Epoch 40

Table A.3 Predicting CER from AALP with ML models, Czech