Enhancing Classical Cipher Type Detection: Prompt Engineering with Common LLMs versus Usage of Custom AI Models

Maik Bastian[1], **Bernhard Esslinger**[2,*], **Eckehard Hermann**[3], **Nils Kopal**[4], **Harald Lampesberger**[3]
[1] CrypTool Project
[2] University of Siegen, Germany
[3] University of Applied Sciences Upper Austria, Austria
[4] Hochschule Niederrhein, Germany
[*] bernhard.esslinger@cryptool.org

Abstract

In the field of cryptography, identifying the type of cipher used in an encrypted message is crucial to effective cryptanalysis. Thus far, from a machine learning perspective, this classification problem has been tackled using specifically designed models, such as the Neural Cipher Identifier (NCID), which require data generation and model training capabilities. The recent advent of Large Language Models (LLMs) raises the following question: Can this classification problem be approached more effectively through prompt engineering? This paper explores various generic strategies for prompt engineering, such as chain-of-thought and in-context learning, by evaluating thousands of generated prompts for classical ciphers using open-source LLMs (on an Nvidia DGX system) and ChatGPT (via a browser interface and API). The classification accuracies achieved through these prompting techniques are compared with those obtained by NCID. Although our findings indicate that NCID still significantly outperforms the use of LLMs for cipher-type detection, the latter offers a more accessible approach to cryptography tasks. Both methods can benefit from domain-specific knowledge in cryptanalysis, highlighting the importance of expert input in improving initial classifications and handling complex cipher types.

1 Introduction

By exploiting weaknesses in a cipher algorithm, a cryptanalyst can attempt to decrypt the ciphertext without knowledge of the key (ciphertext-only attack) or to find the key, when both plaintext and ciphertext are known (known-plaintext attack). For these attacks, the cryptanalyst must first discover the type of cipher used to encrypt the original message. With this knowledge, they can decide on specific techniques appropriate for discovering the message or key of the ciphertext. Since there are a large number of possible cipher types, an automatic tool that discovers the actual cipher type can be very useful. We want to explore prompt engineering with pre-trained large language models (LLMs) as an alternative to established machine learning (ML) models for the task of cipher type detection of classical ciphers.

Training ML algorithms on ciphertexts has proven to be quite successful in identifying the cipher type of a given ciphertext. Neural Cipher Identifier (NCID) (Leierzopf et al., 2022) utilizes ML architectures that employ feature engineering and feature learning approaches, trained with features designed by experts in cryptanalysis and features automatically discovered from ciphertexts, respectively. These models can detect 55 classical cipher types standardized by the American Cryptogram Association (ACA)[1] with an accuracy of 82.78% (Leierzopf et al., 2021). Later additions included the recognition of five World War II-era rotor ciphers, achieving accuracies of 75.82%.[2] Although ML architectures are able to classify ciphertexts accurately, training them requires significant resources and considerable expert knowledge. Although the feature learning approach requires less domain knowledge, significant software engineering expertise is still required to implement the training process. The preparation of data, assembly of models, and implementation of training and evaluation loops all require considerable programming knowledge. Once the models are trained, they need to be stored and made

[1] See: https://www.cryptogram.org/

[2] For details see: https://github.com/cryptool-org/ncid#extended-models-trained-for-recognition-of-aca-and-rotor-ciphers

Proceedings of the 8th International Conference on Historical Cryptology, pages 1-11
Poznań, Poland, 16-18 June, 2025

available to users, which requires dedicated hardware. All these requirements prevent curious, less-technical users from implementing their own cipher identification schemes. Additionally, the trained ML models can only classify the fixed set of cipher types that they were given as input in the training process. Therefore, ciphertexts of an unseen cipher type will be classified as the wrong cipher type.

The advent of LLMs presents new tools for text recognition and processing. This research project evaluates whether they provide new approaches for cryptanalysis and, therefore, for the recognition of cipher types. Our assumption is that the underlying Transformer ML architecture, with its self-attention mechanism (Vaswani et al., 2017), should be able to identify the relations between the characters of the ciphertexts. The presented work explores the performance of several LLMs in classifying ciphertexts using only prompt engineering and compares the resulting classifications to those of NCID. Using the ability of LLMs to learn new tasks without explicit training, known as in-context learning, the process of prompt engineering attempts to improve the output of an LLM solely by altering the textual input to the model (Brown et al., 2020). Enabled by their pattern matching capabilities (Mirchandani et al., 2023), LLMs show potential for the task of ciphertext classification. Using in-context learning of an existing LLM as well as prompt engineering, there is no need to train a specialized ML model, thus lowering the barrier for less technical users. Ideally, they can perform an analysis simply by writing a detailed prompt and submitting it via the interface of an existing LLM. Given the unstructured format of the prompts, we expect the LLMs to be more flexible about the selection of cipher types that they can classify.

Given these considerations, this paper aims to answer the following research questions:

RQ1 How capable are LLMs in classifying classical ciphers using prompt engineering techniques?

RQ2 How do the classifications of LLMs compare to those of specific self-trained ML models?

RQ3 Does the prompt engineering approach enable less-technical users to classify classical ciphers?

To answer these questions, we explore the capabilities of different well-known and performant LLMs in classifying the same 60 classical ciphers (55 ciphers standardized by the ACA as well as five World War II-era ciphers) as examined by NCID, along with a plaintext type for comparison. For modern ciphers like AES or Triple DES, this classification should not work, as their ciphertexts show significantly fewer identifiable features. However, as Gohr showed in (Gohr, 2019), ML-based cryptanalysis of modern ciphers can be effective if they are round-reduced. The explored LLMs include OpenAI's ChatGPT, Meta's Llama, and Mistral's Mistral and Mixtral models.

In Section 2, NCID and its trained ML models for ciphertext classification are described. Afterwards, in Section 3, the generation of the ciphertexts and our developed prompt engineering strategies are presented. Section 4 discusses the accuracies achieved using the different strategies and the tested LLMs. Lastly, in Section 5, the classifications using LLMs are compared to those of NCID, and the usefulness of LLMs for the classification of ciphertexts is assessed.

2 Self-trained ML-based Classification

Self-trained ML models have already been shown to provide useful classifications of classical cipher types. Nuhn and Knight (2014) used neural networks to classify 50 classical ciphers of the ACA. They were able to associate 58.5% of the evaluated ciphertexts with the correct cipher type. Sivagurunathan et al. (2010) classified the three ciphers Playfair, Hill, and Vigenère using neural networks as well. In the following paragraphs, NCID will be presented in more detail.

NCID uses multiple self-trained ML models to implement ciphertext classification. Initially, it was trained to recognize 55 classical ciphers standardized by the ACA (Leierzopf et al., 2022). A further addition enabled the classification of the five World War II-era ciphers: Enigma, M209, Purple, Sigaba, and Typex.[3] NCID can be used through the Web interface at: `https://www.cryptool.org/cto/ncid/`. The models were trained with millions of ciphertexts generated from plaintexts from the Gutenberg library[4].

[3]Details about these additions can be found at: `https://github.com/cryptool-org/ncid#extended-models-trained-for-recognition-of-aca-and-rotor-ciphers`

[4]See: `https://www.gutenberg.org/`

For the classifications of the ACA ciphers, the length of the ciphertexts was set to 100 characters. This number was not determined by systematic research, but because it demonstrated usefulness in previous explorations. Ciphertexts of this length can be identified by manual or computer analysis, while significantly shorter ciphertexts cannot be identified meaningfully by either method, as they lack sufficient information for an unambiguous statistical evaluation. For the further addition of the rotor ciphers into NCID, the length of the ciphertexts was changed to a variable length between 100 and 1000 characters. These longer ciphertexts are needed to differentiate between the rotor ciphers, reducing the uncertainty seen in classifications with shorter ciphertexts. Given the advanced cryptographic complexity of the rotor ciphers, shorter ciphertexts show too few recognizable features for precise identification.

The four ML architectures feedforward neural network (FFNN), random forest (RF), Naive Bayes (NB), and support vector machine (SVM) were trained using a feature engineering approach that takes advantage of statistical features developed beforehand with expert knowledge in the domain of cryptanalysis. These features calculate general statistics of the ciphertexts, like the frequencies of single or multiple letters, the index of coincidence, and specific features targeting individual cipher types. In addition to the feature engineering approach, the two ML architectures Transformer and long short-term memory (LSTM) are trained using a feature learning approach. This approach requires the models to derive the properties of the cipher types without input from expert knowledge, as they are trained solely on raw ciphertexts.

The actual training process was performed iteratively, with a given upper limit of iterations to perform: The models are trained until either this limit is reached, or the process is automatically stopped because the models have not improved for a while. For each iteration, the generated ciphertexts need to be prepared. First, they need to be mapped into a numerical representation to be processable by the ML models. For models using a feature engineering approach, the features of the numerical ciphertext representation are calculated. Models using the feature learning approach will use the numerical representation directly. Once the ciphertexts are processed, the models are trained with the processed ciphertexts and their corresponding cipher types as input. After the training process is completed, the models are evaluated on another previously unseen set of ciphertexts. This time, the models are only provided with the ciphertexts and have to predict the cipher type. The predicted cipher type of each ciphertext is compared to the actual cipher type, and the percentage of correctly predicted cipher types of all evaluated ciphertexts is printed as the model accuracy. The training process must be repeated until all ML models are trained. Finally, the trained models are integrated into an ensemble model, combining their strengths and achieving an accuracy of 75.82% in correctly classifying the 55 ACA and five rotor ciphers.

To achieve these accuracies, many iterations were spent developing the training pipeline, looking for and improving features, and tweaking the hyperparameters of the ML architectures. Depending on the ML architecture, the models were trained between a few hours and up to a week on Nvidia DGX-1 hardware[5]. The resulting models have a combined size of about 2.5 GB. After the training process is completed, the models can classify new ciphertexts on less powerful hardware. Although training can take multiple days, the classification of a new ciphertext takes only a fraction of a second on a normal web server.

3 Prompt Engineering Method

For the evaluation of prompt engineering techniques, the same classical cipher types, as supported by NCID, are selected. These cipher types include the 55 ACA ciphers, the five rotor ciphers, and a plaintext type. The selection allows comparisons with the classifications of NCID and possibly other tools. For simplicity, the plaintext type will not be differentiated from the cipher types in the rest of this text and will simply be referred to as another cipher type. Thus, a total of 61 "cipher types" are evaluated. Approximately one million ciphertexts are created from randomly generated keys and randomly selected English plaintexts from the Gutenberg library, as was done for the training of NCID. The ciphertexts have a fixed length of 100 characters according to the implementation of NCID without the rotor ciphers. Sec-

[5]For more details see: `https://github.com/cryptool-org/ncid#original-models-trained-for-recognition-of-aca-ciphers`

tion 4 provides a discussion of longer ciphertexts. When using variable-length ciphertexts, we observed a tendency of the LLMs to classify ciphertexts based primarily on their length. As a first step, all generated ciphertexts are randomly split into a training set and an evaluation set to prevent leakage of the ciphertext into the context of the classification prompts. Therefore, the evaluation set only contains the ciphertexts that will be classified.

3.1 Naive Approach

First, we tried a naive approach of supplying a simple prompt to the LLM for classification. Although this naive approach does not yield useful results, it demonstrates basic knowledge of the LLMs about some of the ciphers standardized by the ACA and the five rotor cipher machines of the World War II era. Box 1 shows such a naive prompt. The prompt lists the ciphertext and the basic instruction to classify this ciphertext as one of the investigated ciphers.

To improve on the naive approach, the structure and context provided in a prompt are imperative. In an iterative process, many parameters and prompt engineering techniques were tested. Zero-shot, few-shot, and chain-of-thought are examples of such prompt engineering techniques found in literature.

The following paragraphs detail three different strategies, developed by ourselfs for building prompts, using the most promising parameters and techniques. We will refer to these three strategies as the *standard classification strategy*, the *clustered classification strategy*, and the *binary classification strategy*. The results obtained from these strategies are compared in Section 4.

> You are a cryptanalyst. What is the type of cipher of the following ciphertext in quotes "vetntw ornottlyiyettrehotlayneteaofeenfrertrecsheehrrlsm shngoecosehgsaotareoentuitdfrhuidlaotfdinhdxo"? Your options are: "quagmire4", "grandpre", "nihilist_transposition", ...

Box 1: A naive prompt, instructing an LLM to classify the given ciphertext.

3.2 General Considerations

Some general challenges we found while designing prompts for cipher-type detection are the limited input size and the uncommon structure of the prompts. The input text to an LLM is converted into tokens. Depending on the tokenizer, a token can represent a word, a fragment of a word, or a single character (Ali et al., 2024). The size of the prompts is defined by the context window, which specifies the number of tokens an LLM can work with. In the case of ciphertext, single characters are often represented by a single token. Tokenizers based on Byte Pair Encoding (BPE) assign common word fragments or common words to their own tokens but use fallback tokens to represent the individual characters of uncommon character sequences (Sennrich et al., 2016). The random character sequences of the ciphertexts will, therefore, be mostly represented by such character-level tokens. This leads to a very rapidly increasing number of tokens, easily exhausting the context window. Therefore, the amount of additional context that can be given to the LLM is limited, restricting the achievable performance of the model (Huang et al., 2024). Delimiting the ciphertexts explicitly, by providing the samples in an XML-like format, has improved the classifications in our evaluations. An example of this format is shown in Box 2. The ciphertexts of each cipher type are embedded in tags with the name of the cipher type, and each ciphertext is inserted into tags for delimitation.

```
<ciphertext-samples>
  <vigenere>
    <ciphertext>
      MOFOUMZZWVENLQGMULVPR...
    </ciphertext>
    <ciphertext>
      LTIZKTEJCMXYOACRWVLQQ...
    </ciphertext>
  </vigenere>
</ciphertext-samples>
```

Box 2: The format of the ciphertext samples that are part of all prompting strategies.

3.3 Chain-of-Thought Prompting Technique

Each strategy of prompt generation utilizes the chain-of-thought (CoT) prompting technique. Using CoT is an established approach to improve the output of LLMs without the need for fine-tuning (Wei et al., 2022). This technique structures the overall problem into smaller, intermediate steps and provides solutions for each step along with the corresponding reasoning, thus explaining the

overall thought process (Wei et al., 2022). In the context of ciphertext classification, they help the LLMs learn how to analyze ciphertexts and structure their output. A CoT should instruct the LLM on the characteristics of a given cipher type that will be evident in ciphertexts of that type. In this paper, only LLM-generated CoTs are investigated as they have the advantage of not requiring domain knowledge or time-consuming design by experts. To construct such a CoT, an LLM must perform an analysis on its own. Box 3 illustrates an example of a prompt designed to create a LLM-generated CoT. The prompt includes a ciphertext and its corresponding cipher type. The LLM must generate a reasoning for why this ciphertext was encrypted by the cipher type. The ciphertexts for the CoTs are taken from the ciphertext training set. To influence the generated reasoning of the CoT, the prompt includes specific instructions to observe the characteristics of the ciphertexts. These instructions provide general advice, such as to look for patterns in the ciphertext. In addition, they include specific statistical properties useful for cryptanalysis, such as the frequency of letter distributions and the index of coincidence. An example of a CoT can be found at `https://www.cryptool.org/download/research-files/self-generated-cot.png`.

> You are an expert in cryptanalysis. What are the characteristics of the following ciphertext in quotes "OFGXZFSFZEVBUXJOOECOYSXTSJCUROFUHKNUVFRFFELKHFBSQUNQNONJGZHUVBIVMJGFTPDJYNQUSQMJKXZZDJRXZQUHYHDOHMVI" which indicate that it is of type "slidefair"? Compare and contrast it with these other cipher types: "quagmire4", "grandpre", "nihilist_transposition", ...
> To characterize the patterns of the ciphertext use your own language skills as well as cryptanalytical tools, like the IoC, frequency analysis of single letters and bigrams. Limit your output to 2 paragraphs!

Box 3: The structure of CoT prompts for all prompting strategies.

The CoTs are generated by the LLM in a preprocessing step performed before the actual classification. The process of creating the CoTs is illustrated in Listing 1. Before being stored on disk, the CoTs are augmented with a header indicating the key information. This header lists the ciphertext and the corresponding cipher type. Without the header, it cannot be guaranteed that the generated output of the LLM repeats the ciphertext that was given in the initial prompt. Without this ciphertext, the most crucial information of the CoT is missing.

3.4 Standard Classification Strategy

The structure of the prompts for the standard classification strategy is the result of many iterations aimed at improving the classifications. Figure 4 shows the general structure of these prompts, which also serves as the basis for the other classification strategies. Each prompt contains a single ciphertext from the evaluation set that is classified by the LLM as one of the 61 cipher types examined. The prompts start with an initial context for the LLM, which includes the task the LLM has to perform, its role, as well as the structure it is expected to find in the rest of the prompt. Following this initial context, the previously generated CoTs and ciphertexts samples are provided. The ciphertexts for the CoTs and samples are extracted from the training set once more. The prompt is finalized with the actual instruction for the task to be performed. This instruction contains the ciphertext to classify, as well as a list of all 61 possible cipher types that the LLM can choose as a classification result. The instruction references the provided CoTs and samples and it requests a specific output format to simplify processing, evaluation, and debugging of the generated classification.

With ciphertexts of length 100, one CoT per cipher type, and 20 samples per cipher type, the prompts have a length of approximately 104,000 tokens. This number of tokens would, for example, exceed the context window of Mistral's Mixtral 8x7B model.[6] For complete evaluation of the classification accuracy of all 61 cipher types, 61 prompts must be provided to the LLM.

3.5 Clustered Classification Strategy

To improve the standard classification strategy with its large prompt size and numerous cipher types, the clustered classification strategy was conceived. To apply this clustering, expertise in the cryptographic domain was needed, thus limiting the following approaches to users with this expert knowledge. For the clustered classification strategy, the 61 ciphers are divided into 12 distinct

[6]For documentation of Mistral's models see: `https://docs.mistral.ai/getting-started/models/models_overview/`

```
chain_of_thoughts = {}
for cipher_type in cipher_types:
    ciphertext = train_dataset.random_ciphertext(cipher_type)
    prompt = f"Explain why '{ciphertext}' is of type '{cipher_type}'."
    cot = llm.generate(prompt)
    chain_of_thoughts[cipher_type] = cot
```

Listing 1: Pseudocode for generating CoTs for all cipher types.

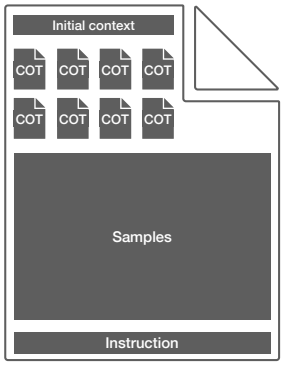

Figure 4: The general structure of the classification prompts used to classify a single ciphertext.

clusters. A table of the clusters and their corresponding cipher types can be found at `https://www.cryptool.org/download/research-files/cipher-clusters.png`. These clusters contain cipher types with comparable characteristics. Therefore, ciphers from different clusters are expected to be more easily distinguishable. For each complete evaluation, a single cipher type is randomly selected as a representation of the cluster. For example, the *Railfence* cipher is selected as a representative of the *Transposition* cluster. The ciphertexts of the selected cipher will be used in the CoTs, samples, and for the final classification. This means that effectively, 12 instead of 61 ciphers are compared by an LLM in each prompt. This strategy is expected to improve the classifications, as the differences in the ciphertexts will be greater than in the standard strategy and there will be less context in each prompt. Using a single CoT per cipher type and 20 samples per cipher type, the number of tokens per prompt is approximately 20,000 tokens. For a complete evaluation of classification accuracy, only 12 prompts must be executed, instead of the 61 prompts required by the standard strategy.

3.6 Binary Classification Strategy

The binary classification strategy is built on top of the clustered strategy, using only the 12 cluster types. The strategy is designed to reduce the amount of context needed in each prompt, helping the LLM to focus on the relevant characteristics of the ciphertext. Instead of augmenting the classification prompt with CoTs and samples of all examined cipher types, the binary classification strategy lists only CoTs and samples of a single cipher type in the classification prompt. The LLM has to determine whether the ciphertext matches this cipher type. For a complete classification of a single ciphertext, at least as many classifications as there are cipher types have to be performed. A complete evaluation of the 12 types of clusters would require at least $12 * 12 = 144$ prompts. Because the classification of a single ciphertext is split into several independent prompting steps, the LLM cannot directly compare the different cipher types and does not know its previous answers for the given ciphertext. Consequently, false positive results for a single binary classification are quite frequent. For better results, a single ciphertext is classified in multiple executions, ideally using different CoTs and samples for each execution. In the end, the LLM has to perform a reflection phase, using the results of the previous classifications to infer a single combined classification of the ciphertext. The prompt for the reflection phase contains the ciphertext, previous answers to classification executions, and samples of the cipher types answered in those previous executions. The LLM then has to select the overall answer. This reflection phase can also be performed after a single classification execution, but using multiple executions can improve accuracy. Listing 2 illustrates the process of the binary classification strategy.

The size of the classification prompts is approximately 3,500 tokens, and thus significantly smaller than in the previous strategies. The length of the prompts of the reflection phase depends on the number of positive classifications of the classi-

```
classifications = {}
for eval_cipher_type in cipher_types:
    eval_ciphertext = eval_dataset.random_ciphertext(eval_cipher_type)
    for tested_cipher_type in cipher_types:
        cot = chain_of_thoughts[tested_cipher_type]
        samples = train_dataset.load_random_samples(tested_cipher_type)
        prompt = f"""
            Analysis of {tested_cipher_type}:
            {cot}
            Samples of {tested_cipher_type}:
            {samples}
            Is '{eval_ciphertext}' of type '{tested_cipher_type}'?
            Answer with YES or NO.
            """
        classification = llm.generate(prompt)
        classifications[eval_cipher_type][tested_cipher] = classification
    if perform_reflection:
        positive_classifications = find_positive_classifications(
            classifications[eval_cipher_type]
            )
        samples = train_dataset.load_random_samples(positive_classifications)
        prompt = f"""
            The previous positive classifications for '{eval_ciphertext}'
            include the following cipher types: {positive_classifications}.
            Here are samples of these cipher types:
            {samples}
            Given these cipher types, what is the actual cipher type of
            the ciphertext '{eval_ciphertext}'?
            """
        final_classification = llm.perform(prompt)
        classifications[eval_cipher_type] = final_classification
```

Listing 2: Pseudocode for generating all classifications of the binary classification strategy.

fication phase and will be approximately 1,500 tokens long. Using a single execution per ciphertext, the total number of prompts that must be executed for a complete evaluation of the classification accuracy is the product of the number of cipher types multiplied by itself for the number of classification prompts, plus the number of cipher types for the number of reflection prompts. Therefore, with n as the number of cipher types, the total number of prompts is given by: $n*(n+1)$. With $n = 12$ clusters, the number of prompts equals 156.

4 Evaluation

The final results of the different prompting strategies described in Section 3 are assessed comparing the accuracies of the classifications of all strategies. The accuracies of the classifications are calculated after the complete execution of each of the three strategies, where the accuracy is the percentage of correctly identified cipher types out of all tested cipher types. The open-source models Meta Llama-3.1-70B-Instruct (quantized to 8-bit precision) and Meta Llama-3.1-8B-Instruct (unquantized), as well as OpenAI's ChatGPT 4o, were evaluated. While working on this research project, OpenAI had updated their ChatGPT 4o model. The specific model we used for the evaluation is ChatGPT 4o-2024-08-06. Additionally to the Llama models, two further open-source LLMs were tested: The quantized versions of Mistral-7B-Instruct-v0.3 and Mixtral-8x7B-Instruct-v0.1 with 16-bit precision. All these models were chosen because of their position on the huggingface leaderboard[7] and because they had already shown promising results for us in other tasks. However, of these four open-source models, only Llama-3.1-70B-Instruct and Llama-3.1-8B-Instruct showed acceptable ciphertext classifications in our tests.

All results were obtained using a Python script and the generated ciphertexts. This Python script simplified prompt generation and interaction with the LLMs. The open source models were executed on an Nvidia DGX H100[8], with eight Nvidia H100 GPUs with a total of 640 GB of VRAM, 2 Intel CPUs with a total of 112 cores, and 2 TB of RAM.

[7] For the current version of the leaderboard see: `https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard`

[8] Details about Nvidia DGX H100: `https://docs.nvidia.com/dgx/dgxh100-user-guide/introduction-to-dgxh100.html`

Each strategy was executed five times. The following results are the average of these five executions.

4.1 Evaluation with Short Ciphertexts

First, we discuss the results for ciphertexts with a length of 100 characters. Table 1 shows the classification accuracies of the Llama-3.1-70B-Instruct model, which was quantized to 8-bit precision. The best results are achieved using the binary classification strategy with an accuracy of 53.3%. Clustered classification shows less accurate results, while the standard classification strategy only achieves poor results. The evaluation of the Llama-3.1-8B-Instruct model is not detailed here for the sake of brevity. The classifications of this model are about 5 to 10 percent worse. However, it generates the results about three times faster than Llama-3.1-70B-Instruct quantized with 8-bit precision. For comparison, the superior classification results of NCID are shown in Table 3.

Table 1: Results of Llama-3.1-70B-Instruct quantized with 8-bit precision.

Strategy	Accuracy
Standard strategy	9.2%
Clustered strategy	43.3%
Binary strategy	53.3%

To compare the performance of open-source models, OpenAI's ChatGPT 4o was used as a commercial alternative. It achieved the best results compared to the open-source models. Table 2 presents the average accuracies for each of the three prompting strategies. Although the results are better than those of the Llama 3.1 models, the trend between the strategies behaves the same way as with Llama-3.1-70B. Only the clustered classification strategy and the binary classification strategy achieve better accuracy than random.

Table 2: Results of ChatGPT 4o-2024-08-06.

Strategy	Accuracy
Standard strategy	19.3%
Clustered strategy	46.7%
Binary strategy	58.3%

The standard classification strategy yields poor results, regardless of the model. In contrast, both the clustered classification strategy and the binary classification strategy significantly improve these results. We assume that the improvements achieved by these strategies in our evaluations can be attributed to the reduced set of cipher types that need to be recognized. Clustering of cipher types also provides the LLMs with ciphertexts with more distinct characteristics. The reduction of superfluous information in the prompts of the binary classification strategy, in comparison to the clustered classification strategy, further improved our results. Consequently, the binary classification strategy appears to be the best strategy, as it can provide all CoTs and ciphertext samples for each cipher type without overloading individual classification prompts. However, a disadvantage of this strategy is the considerable execution time required for both single classifications and complete evaluations of all cipher types.

4.2 Evaluation with Longer Ciphertexts

Experimentation with longer ciphertexts, such as with a length of 200 or 1000 characters, did not yield better classification results. Especially the 1000-character variant showed issues with regard to the increased number of tokens per prompt. Even in the binary classification strategy, the classification prompts have an average size of 22,000 tokens, compared to 3,500 tokens when using ciphertexts with 100 characters. The prompts of the standard classification strategy have an average size of 650,000 tokens when ciphertexts with 1000 characters are used, which exhausts the context window of many LLMs. There is a trade-off between the additional statistical information and patterns available in longer ciphertexts and the overall size of these more extensive prompts. At least with the models tested, the use of ciphertexts with more than 100 characters did not help the LLMs in identifying patterns.

4.3 Comparison with Specific Self-Trained Models

Compared to NCID, even the classifications achieved by ChatGPT 4o are unsatisfactory. In Table 3, the accuracies of the classifications using NCID are presented. The extended NCID is trained and evaluated with variable-length ciphertexts between 100 and 1000 characters, while the LLMs are only evaluated with ciphertexts that are 100 characters long. The standard evaluation uses the ensemble ML model and classifies all cipher types. The clustered evaluation uses the same clusters as in the prompt engineering strategies, and instead of retraining the existing NCID models, the accuracies of this evaluation are de-

termined after the models have made their predictions. In this case, a prediction is marked as correct if the predicted cipher type matches one of the cipher types in the correct cluster. Although the standard classification strategy could not successfully differentiate the 61 cipher types investigated, even when using ChatGPT 4o, NCID provides useful accuracies for classifications of all cipher types.

Table 3: Results of NCID.

Strategy	Accuracy
Standard evaluation	75.82%
Clustered evaluation	93.52%

5 Conclusion and Outlook

The presented work examined strategies for building prompts to detect cipher types and compared the classifications to ML-based NCID. We found that the format of the prompts is very important for these classifications. The CoT technique, together with samples of ciphertexts, provided LLMs with an additional context, improving their output of the classifications and the corresponding reasonings. The following subsections answer the research questions defined in Section 1.

5.1 RQ1: How capable are LLMs in classifying classical ciphers using prompt engineering techniques?

When evaluating the prompting strategies with different LLMs, it is evident that our most basic standard classification strategy using all 61 ACA and World War II-era rotor ciphers does not produce useful classifications. The accuracies achieved even by the most advanced models, classifying all 61 ciphers, only reach 20%. Applying a more distinct and simplified set of ciphers, as well as more complex prompting techniques, improves those results, producing average classifications. The clustered classification strategy, which uses only 12 types, achieves accuracies of 46.7% (see Table 2). The best classifications were achieved by the binary classification strategy. With this strategy, we reached an accuracy of 58.3%. As with the clustered classification strategy, the binary classification strategy considers a smaller set of cipher types compared to the standard classification strategy. Furthermore, the strategy reduces the size of the prompt by embedding only the CoTs and samples of a single cipher type, limiting the amount of superfluous information seen in each prompt.

5.2 RQ2: How do the classifications of LLMs compare to those of specific self-trained ML models?

Compared to classical ML techniques for ciphertext classification, as used by NCID, the classification accuracies achieved with our prompting strategies are significantly worse. Training specific ML models, especially with features developed using domain knowledge, can lead to great results. NCID achieves an accuracy of 75.82% when classifying all 61 cipher types. It only needs around 6 GB of VRAM to execute the trained models and can classify a ciphertext in fractions of a second. A disadvantage of ML models is the time-consuming training process. Although there is no need to train the existing LLMs when using in-context learning, classifying a single ciphertext takes significantly longer using LLMs. Moreover, considerable amounts of computing resources are needed to execute such a LLM. As an example, loading the full Llama-3.1-70B-Instruct model requires around 240 GB of VRAM. Relying on external providers to interface with an LLM removes the need for such hardware but introduces some disadvantages. In addition to the associated cost and potential usage limits, there is no guarantee that a hosted model will not be altered or replaced by the provider. While working on this research project, OpenAI had released a newer version of their ChatGPT 4o model, which generated worse output for the same prompts. Table 4 presents a comparative analysis of the advantages and disadvantages of employing prompt engineering with existing LLMs, as opposed to training specific ML models for the classification of ciphertext.

5.3 RQ3: Does the prompt engineering approach enable less-technical users to classify classical ciphers?

A big advantage of using LLMs for the task of cipher-type classification is the more approachable nature of the tools, decreasing the reliance on domain experts. Using the strategies detailed in this paper with sufficient samples of ciphertexts with known types, anyone can implement a classification scheme simply by providing written instructions to an existing LLM. The textual representation of the output can be helpful in inspecting the thought process of the LLM and for iterating

Table 4: Comparison of self-trained ML models and LLM prompt engineering for ciphertext classification, distinguished by development and production context.

Context	+ / -	ML models	LLM prompt engineering
Development	**Advantages**		• Classifications are interpretable • No need to train a model • Less reliance on domain experts
	Disadvantages	• Long training time • High hardware requirements for training • Domain experts and software engineers required	• Improvements to prompts can be non-obvious
Production	**Advantages**	• Best classification results • Fast classification • No dependency on external providers	• Classifications are interpretable
	Disadvantages	• Results hard to interpret	• Slow classifications • Dependency on external LLM providers, or • very high hardware requirements when self-hosted

the chosen approach. However, as shown by the unsatisfactory results of the standard classification strategy, cryptographic experts are still required to improve upon the initial classifications, especially when inspecting a multitude of possible cipher types.

5.4 Outlook

Future enhancements to LLMs could improve our prompt engineering approaches. Larger context windows would allow for longer prompts with more CoTs and samples with longer ciphertexts. There are further opportunities to optimize detailed prompting strategies. For example, the reflection phase of the binary classification strategy could be enhanced with statistics from previous evaluations, providing the LLMs with insight into their previous false classifications. Until these enhancements are developed, specifically trained ML models remain the better solution for ciphertext classification.

Acknowledgments

This work has been supported by Riksbankens Jubileumsfond, grant M24-0028: Echoes of History: Analysis and Decipherment of Historical Writings (DESCRYPT).

The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

[Ali et al.2024] Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, Charvi Jain, Alexander Weber, Lena Jurkschat, Hammam Abdelwahab, Chelsea John, Pedro Ortiz Suarez, Malte Ostendorff, Samuel Weinbach, Rafet Sifa, Stefan Kesselheim, and Nicolas Flores-Herr. 2024. Tokenizer choice for LLM training: Negligible or crucial? In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3907–3924, Mexico City, Mexico, June. Association for Computational Linguistics.

[Brown et al.2020] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

[Gohr2019] Aron Gohr. 2019. Improving attacks on round-reduced speck32/64 using deep learning. Cryptology ePrint Archive, Paper 2019/037.

[Huang et al.2024] Xijie Huang, Li Lyna Zhang, Kwang-Ting Cheng, Fan Yang, and Mao Yang. 2024. Fewer is more: Boosting llm reasoning with reinforced context pruning. In *EMNLP*, February.

[Leierzopf et al.2021] Ernst Leierzopf, Vasily Mikhalev, Nils Kopal, Bernhard Esslinger, Harald Lampesberger, and Eckehard Hermann. 2021. Detection of Classical Cipher Types with Feature-Learning Approaches. In *Data Mining. AusDM 2021. Communications in Computer and Information Science*. Springer Singapore.

[Leierzopf et al.2022] Ernst Leierzopf, Nils Kopal, Bernhard Esslinger, Harald Lampesberger, and Eckehard Hermann. 2022. A Massive Machine-Learning Approach For Classical Cipher Type Detection Using Feature Engineering. In *Proceedings of the 4th International Conference on Historical Cryptology HistoCrypt 2021*, jun.

[Mirchandani et al.2023] Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. 2023. Large language models as general pattern machines. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2498–2518. PMLR, 06–09 Nov.

[Nuhn and Knight2014] Malte Nuhn and Kevin Knight. 2014. Cipher type detection. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1769–1773, Doha, Qatar, oct. Association for Computational Linguistics.

[Sennrich et al.2016] Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.

[Sivagurunathan et al.2010] Ganapathi Sivagurunathan, Velayutham Rajendran, and Dr. T. Purusothaman. 2010. Classification of substitution ciphers using neural networks. In *IJCSNS International Journal of Computer Science and Network Security*.

[Vaswani et al.2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

[Wei et al.2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.