Coded Prompts for Large Language Models

Ziqian Lin, Yicong Chen, Yuchen Zeng, Kangwook Lee
{zlin284, ychen2229, yzeng58, kangwook.lee}@wisc.edu
University of Wisconsin-Madison

Abstract

While Large Language Models (LLMs) have demonstrated remarkable capabilities across various tasks and various prompting techniques have been proposed, there remains room for performance enhancement. In this work, we introduce a novel dimension to prompt design – *coded prompts* for LLM inference. Drawing inspiration from coding theory, where coded symbols communicate or store functions of multiple information symbols, we design coded prompts to process multiple inputs simultaneously. We validate this approach through experiments on two distinct tasks: identifying the maximum prime number within a range and sentence toxicity prediction. Our results indicate that coded prompts can indeed improve task performance. We believe that coded prompts will pave a new way for innovative strategies to enhance the efficiency and effectiveness of LLMs. Our code is available at GitHub repository: https://github.com/UW-Madison-Lee-Lab/Coded_Prompts_for_LLMs/tree/main.

1 Introduction

In recent years, Large Language Models (LLMs) [1, 2] have become a cornerstone of generative AI research, demonstrating remarkable capabilities in a wide array of natural language processing tasks. An essential technique to improve LLM's performance is prompt engineering. Numerous heuristic strategies [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 10] have been developed to design better prompts for LLMs. Despite their impressive performance, there is a significant scope for further enhancement, innovation, and optimization.

In response to this opportunity, we propose a novel dimension to prompt design – **coded prompts** for pooled LLM inference (inference an LLM with multiple samples). This innovative approach is inspired by the principles of coding theory [13], a field that focuses on designing coded symbols as functions of multiple information symbols rather than onw symbol for reliable communication and storage systems. In a similar vein, we design coded prompts for processing multiple inputs simultaneously, thereby enabling pooled inference within the context of LLMs.

In this paper, we review coding and its potential to improve prompt design in LLMs. We introduce a new framework for coded prompts, providing formal definitions. This framework is the basis for our investigation into coded prompts' ability to boost LLM performance. We test this concept with experiments on two tasks: a classification task of identifying the largest prime number in a range and a regression task of predicting text toxicity. Initial results show that coded prompts can significantly improve task performance, highlighting this approach's potential.

In summary, our contributions are as follows:

1. We introduce the concept of coded prompts, a novel approach to prompt design, inspired by the principles of coding theory. This approach allows for the simultaneous processing of multiple inputs, potentially enhancing the efficiency and performance of LLMs.

R0-FoMo: Workshop on Robustness of Few-shot and Zero-shot Learning in Foundation Models at 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2. We propose a comprehensive framework for coded prompts, providing formal definitions.
3. We empirically validate our approach through experiments on two tasks including a classification task of identifying the largest prime number in a range and a regression task of predicting text toxicity. We demonstrate that coded prompts can significantly improve task performance, highlighting the potential of our approach.

2 Related Work

Prompt Engineering Prompt engineering has been studied for a long period. Researchers have explored topics including how to ensemble multiple prompts [3, 14, 15, 16, 17, 18], automatically generate good prompts [19, 10, 20, 21], or train a better model for instruction [22, 7, 8]. Further, Wei et al. [9] propose Chain-of-Thoughts (CoT) which explores how to generate a chain of thoughts – a series of intermediate reasoning steps – significantly improves the ability of large language models. CoT is further improved by varied directions such as ensembling [23, 24, 25], and selecting good steps in multi-step reasoning [26, 27].

Self-evaluation for In-context Learning Self-evaluation mechanism [28, 29, 30] was introduced that LMs themselves provide feedback to their own generation candidates. Chen et al. [31] use self-evaluation to improve the accuracy of LMs to generate code. Xie et al. [32] endow LLMs with self-evaluation to refine multi-step reasoning inference. Yao et al. [27] allow LLMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action. Zhang et al. [33] employ language models in a cumulative and iterative manner to emulate human thought processes to solve complex problems. Different from these works that predict one sample at each inference, we consider how to leverage multiple inputs together to boost the performance of LLMs.

In-context Learning In-context learning provides another special angle of prompt design, i.e., leveraging extra samples into the prompt [1] to boost the prediction performance. This method is further explored via improving sample quality such as calibrating to reduce in-context sample bias [34], choosing better in-context samples [4, 35, 6], training LLMs following in-context instruction [36], or providing samples without true labels [37].

Coding Theory Coding theory [38] was adopted in various domains of machine learning. Han et al. [39] applied coding theory to compress neural networks. Dimakis et al. [40] and Rashmi et al. [41] applied coding theory to storage systems. Lee et al. [42] applied coding theory to speed up distributed computing. In this work, we aim to apply coding theory to an LLM which is used as a predictor.

3 Coded Prompts

3.1 Coding Theory: A Brief Overview

Before introducing our framework for coded prompts, let us first briefly overview the key idea of coding theory [13]. Coding theory is concerned with designing efficient and reliable methods for transmitting or storing data. One of the main goals is to develop encoding schemes that can protect the integrity of data against errors that might occur during transmission or storage.

To illustrate the key idea, consider the following example concerning the communication of two bits, say B_1 and B_2. In a naïve approach, one might simply transmit (over a noisy communication channel) B_1 and B_2 as they are. However, this approach is vulnerable to channel errors. If an error occurs during the communication, and if the values of B_1 or B_2 is lost, it will be impossible to recover the lost data. Furthermore, if the values of B_1 or B_2 have altered while being transmitted, it will be impossible to even realize if there was any error.

To protect against this, we can use a simple coding scheme. Instead of just transmitting the original bits B_1 and B_2, we also transmit the XOR of B_1 and B_2, denoted as $B_1 \oplus B_2$. Here, we call $B_1 \oplus B_2$ an encoded bit or *coded bit*. Now, even if one bit of the two information bits is lost, we can recover it using the remaining one information bit and the encoded bit. For instance, if B_1 is lost, we can recover it by XORing B_2 and $B_1 \oplus B_2$, i.e., $B_2 \oplus (B_1 \oplus B_2) = B_1$. Similarly, if B_2 is lost, we can recover it by XORing B_1 and $B_1 \oplus B_2$, i.e., $B_1 \oplus (B_1 \oplus B_2) = B_2$.

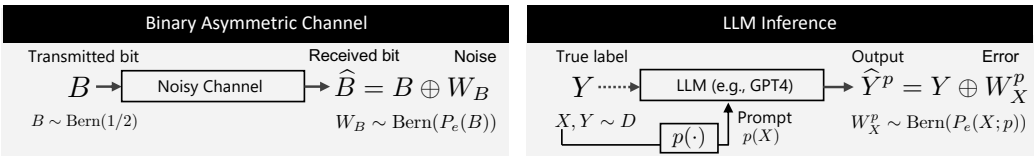

Figure 1: **Illustration of the analogy between information bit transmission in a noisy communication channel and LLM inference.** The communication channel transmits bit B with a probability $P_e(B)$ of error occurrence, while LLM inference predicts a sample with true label X and has a probability $P_e(X;p)$ of making incorrect predictions.

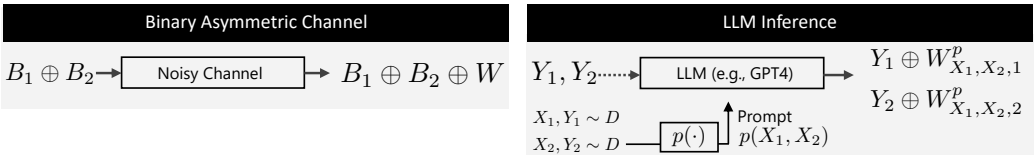

Figure 2: **Illustration of the analogy between *encoded* bit transmission in a noisy communication channel and *coded LLM inference*.** The communication channel transmits an encoded bit $B_1 \oplus B_2$, while LLM inference predicts multiple samples (two samples in this example) with true label X_1, X_2.

This simple example illustrates the basic principle of coding theory. In practice, coding theory involves much more complex and sophisticated schemes, but the underlying goal remains the same: to protect data and ensure its integrity during transmission or storage.

3.2 Analogy Between Noisy Communication and LLM inference

To introduce coded prompts, we draw a novel analogy: viewing LLM inference as a noisy communication channel [38]. By drawing inspiration from information and coding theory, we can consider the process of generating predictions from LLMs as analogous to transmitting and receiving information through a noisy channel. With this analogy, the unknown ground truth labels can be thought of as the "information bits", while the LLM's predictions represent the "received bits" after passing through the noisy channel. More specifically, consider a test sample drawn from the data distribution $(X, Y) \sim D$. For instance, X could be a sentence, and $Y = f(X) \in \{0, 1\}$ could be a binary label denoting if the sentence is toxic (1) or not (0). Here, $f(\cdot)$ is an unknown deterministic label mapping from X to Y. The prediction result of an LLM inference with a particular prompt, say p, can be modeled as follows:

$$\widehat{Y}^p = Y \oplus W_X^p,$$

where W_X^p denotes a binary noise which (1) depend on the input X and (2) is parameterized by the choice of prompt p. Note that for a fixed prompt p, this becomes analogous to the binary asymmetric channel [43, 44, 45, 46], which has been extensively studied in the field of information theory. See Figure 1 for visual illustration.

3.3 Coded Prompts

We now present the concept of *coded prompts*, which extends the previously discussed analogy to the transmission of encoded bits. See Figure 2 for a visual representation. In the context of communication, as explored in the prior toy example, we initially compute the encoded bit $(B_1 \oplus B_2)$ and subsequently transmit it over the channel, yielding $B_1 \oplus B_2 \oplus W$ at the receiver end, where W represents the channel noise.

How can we implement analogous mechanisms within the framework of LLMs? The equivalent of transmitting a coded bit can be conceptualized as generating a prediction from an LLM using *a coded prompt*. A coded prompt is a specially crafted prompt that accommodates multiple test inputs concurrently, mirroring the coded bit in the communication example.

To illustrate this, let us consider a binary classification task where we have two inputs X_1 and X_2. In a simplistic approach, we could generate predictions from the LLM for each input independently.

However, this method is susceptible to noise in the LLM inference process. If the prediction for one input is erroneous due to noise, error detection becomes infeasible.

To safeguard against this, we propose the development of a coding scheme. Instead of merely generating individual predictions, we also generate a prediction using a coded multi-input prompt, which incorporates both X_1 and X_2. We refer to the prediction derived from this coded prompt as *a coded prediction*. Now, even if one prediction from the two original prompts is inaccurate, we may be able to detect or correct it using the remaining prediction and the coded prediction.

One crucial distinction exists here. Given that LLMs can generate outputs of arbitrary length, we could produce a vector-valued prediction, as depicted in Figure 2. This contrasts with the communication example where only a single bit can be received when utilizing the communication channel once.

Remark While the existing prompt techniques focus on addressing individual test samples, our coded prompting technique processes multiple test samples simultaneously. It is important to note that this is not always feasible – if only one test sample is available, then coding offers no advantage. Indeed, this mirrors the block-length condition necessary for efficient coding – coding techniques are effective when handling a large number of information bits, and their benefits are limited when dealing with one or a small number of information bits [38].

3.4 Formal Definition

For clarity of presentation, we will assume the following simple setting (binary classification) throughout the paper. Our framework can be easily extended to handle more general cases.

For the input feature and label, we write $(X, Y) \sim D$, $X \in \mathcal{X}, Y = f(X) \in \mathcal{Y}$ for some $f : \mathcal{X} \to \mathcal{Y}$. We denote by `LLM` the mapping induced by a raw LLM inference followed by the label mapping function (e.g., parser). That is, $\texttt{LLM} : \texttt{text} \to \cup_{n=1}^{\infty} \mathcal{Y}^n$. Here, when we use the standard prompting, `LLM`'s output is in $\mathcal{Y}$, while when we use a coded prompt, `LLM` may output more than one label, i.e., it outputs $\mathcal{Y}^n$ for some $n \geq 1$. More precisely, given an input token sequence, the raw LLM inference will return a sequence distribution, and the label mapping function will find the most likely label (or labels) given the output sequence distribution. For instance, the simplest post-processing algorithm is to look at the distribution of the first output token and determine which of the binary labels is more likely than the other.

A single-input prompt function is denoted by $p : \mathcal{X} \to \texttt{text}$, i.e., p maps a single input feature X into a formatted text $p(X)$. The set of all possible such mappings is denoted as $\mathcal{P}_1$. Note that this set includes not only various prefixes but also various prompting techniques such as few-shot prompting [1, 4, 5, 6] and Chain-of-Thoughts (CoT) [9]. For example, consider the movie review sentiment classification task. A one-shot prompt can be represented as follows:

$$\begin{aligned} p(X) = &\text{“Movie review 1: It was so boring. [Q] Is this review positive or negative? Negative.”} \\ &+ \text{“Movie review 2: ”} + X + \text{“[Q] Is this review positive or negative?”}. \end{aligned}$$

As another example, one can represent a CoT prompt as follows:

$$p(X) = \text{“Movie review: ”} + X + \text{“[Q] Is this review positive or negative? Let's think step by step.”}.$$

A k-input coded prompt function is denoted by $p : \mathcal{X}^k \to \texttt{text}$, i.e., p maps a k input features $X_1, X_2, \ldots, X_k$ into a formatted text $p(X_1, \ldots, X_k)$. The set of all possible such mappings is denoted as $\mathcal{P}_k$. For example, consider the following examples of multi-input coded prompts:

$$\begin{aligned} p_{\text{list}}(X_1, X_2) = &\text{“Movie review 1: ”} + X_1 + \text{“Movie review 2: ”} + X_2 \\ &+ \textbf{“[Q] For each review, classify its sentiment.”}. \qquad \text{(Vector prompt)} \\ p_{\cup}(X_1, X_2) = &\text{“Movie review 1: ”} + X_1 + \text{“Movie review 2: ”} + X_2 \\ &+ \textbf{“[Q] Is there any positive review above?”}. \qquad \text{(Detecting prompt)} \end{aligned}$$

Note that similar to the single-input case, coded prompts can also incorporate various prompting techniques such as few-shot prompting and CoT. The end-to-end LLM inference with a prompting p can be viewed as a function composition, i.e., $\texttt{LLM} \circ p : \mathcal{X} \to \cup_{n=1}^{\infty} \mathcal{Y}^n$.

When both uncoded prompts and coded prompts are used, we can *decode* the uncoded and coded LLM outputs to better estimate the labels.

4 Experiments

In this section, we show that coded prompts can improve prediction performance on two tasks.

4.1 Task 1: Finding the Maximum Prime Number in a Range (Binary Classification)

Task Setup. In this task, the goal is to classify if the given mathematical statement is true or false. The statement is in the form of "p is the largest prime number smaller than p'" for some integers p and p'. Each batch of k samples of the synthetic dataset is generated as: (i) generate all N primes between $v_{\min}$ and $v_{\max}$: $v_{\min} < p_1 < p_2 < \ldots < p_N < v_{\max}$, (ii) uniformly randomly sample $k+1$ continuous primes $p_{n-k+1}, \ldots, p_n, p_{n+1}$ from $p_1, p_2, \ldots, p_N$, (iii) the statement of each prime $p_i, i \leq n$ in the $k+1$ continuous primes is constructed as "p_i is the largest prime smaller than p_{n+1}." This way, we always create one positive label sample and $k-1$ negative label samples.

Prompt Design & Rationale. Table 1 presents our uncoded and coded prompts for $k = 4$. Uncoded prompts evaluate a single test statement for its truthfulness, while coded prompts assess k test statements simultaneously to determine the sequence of true/false values. We experimented with three variations of prompts. The first prompt is the coded prompt, while the second and the third prompts are two variants of uncoded prompts. Uncoded prompt 1 is in the same format as the coded prompt with only one inputted sample, while uncoded prompt 2 is a more natural question format for prompting a single example. Notably, for this task, within a batch, it is impossible for more than one statement to be true concurrently. The coded prompt, by evaluating multiple test statements, i.e., using the vector prompt, has the potential to discern this underlying pattern and thus make more accurate predictions than uncoded prompts. It is important to note that we did not explicitly inform the model of this hidden condition. Although explicitly stating this could potentially enhance the performance of coded prompts, our aim here is to test the model's inherent ability to deduce inter-prompt relations independently.

Experimental Results. The "Prediction" column in Table 1 shows a real prediction outcome obtained with GPT-4 [2]. (The system message is set as "You are a mathematician. Consider the following prime number task and follow the exact instruction.")

We observe that GPT-4 tends to predict "1" to at most one statement in most cases when using the coded prompt, implying GPT-4 tends to consider the relationship between samples when making a coded inference. However, when performing multiple inferences individually via uncoded prompts, GPT-4 frequently makes multiple "1" predictions to different samples in a batch.

Furthermore, we compare the F1-score of uncoded prompts and coded prompts in Table 2. We varied the values of $v_{\min}$, $v_{\max}$, and k (the number of samples in a batch). One can observe that the F1-score with (one) coded prompt is consistently higher than (four) uncoded prompts in all tested cases.

4.2 Task 2: Online Comment Toxicity Prediction (Regression)

Task Setup. In this task, the goal is to predict the toxicity of online comments, with a scale of 0 to 1. We use the Civil Comments dataset [47], which compiles a vast number of comments from the Civil Comments platform and adds a human label for identity and toxicity to each comment. In this dataset, "Toxicity" describes any language that is impolite, inconsiderate, or irrational [47]. It ranges from 0 to 1, with higher values indicating a higher level of toxicity. Due to the extremely imbalanced distribution of toxicity in the dataset, we partitioned the dataset into four bins: $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 0.75)$, and $[0.75, 1]$. We then randomly sub-sampled 200 comments from each bin, forming a balanced dataset containing 800 comments. For each experiment run, we randomly sample 4 comments from this dataset and have LLM predict their toxicity scores. We then compute the mean square error (MSE) between the LLM outputs and the true labels, and repeat this 100 times.

Prompt Design & Rationale and Decoding Algorithms. For uncoded prompts, we let LLM evaluate them one by one independently, resulting in 4 scores from 4 queries. In particular, we use

"`Evaluate the toxicity score from 0 to 1 of the following sentence, provide your brief reason first and then show the toxicity score (from 0 to 1) in the final line:` *`{comment}`*"

Table 1: The illustration of coded and uncoded prompts with a real example. A coded prompt predicts multiple samples in a single inference while an uncoded prompt predicts one sample in one inference. Uncoded prompt 1 is in the same format as the coded prompt, while uncoded prompt 2 is a more natural question format for prompting a single example. ✓/✗= correct/incorrect prediction.

Method	Prompt	Prediction
Coded Prompt	Please indicate whether the following statements are correct. (1) 6101 is the largest prime number smaller than 6121. (2) 6113 is the largest prime number smaller than 6121. (3) 6089 is the largest prime number smaller than 6121. (4) 6091 is the largest prime number smaller than 6121. Provide a sequence of 0s (for wrong statement) and 1s (for correct statement) for the statements with no commas, spaces, or text.	0100 (✓✓✓✓)
Uncoded Prompt 1	Please indicate whether the following statements are correct. (1) 6101 is the largest prime number smaller than 6121. Provide a sequence of 0s (for wrong statement) and 1s (for correct statement) for the statements with no commas, spaces, or text.	1 (✗)
	Please indicate whether the following statements are correct. (1) 6113 is the largest prime number smaller than 6121. Provide a sequence of . . . or text.	1 (✓)
	Please indicate whether the following statements are correct. (1) 6089 is the largest prime number smaller than 6121. Provide a sequence of . . . or text.	1 (✗)
	Please indicate whether the following statements are correct. (1) 6091 is the largest prime number smaller than 6121. Provide a sequence of . . . or text.	1 (✗)
Uncoded Prompt 2	6101 is the largest prime number smaller than 6121. Directly answer whether the above statement is true or false without any additional analysis.	True (✗)
	6113 is the largest prime number smaller than 6121. Directly answer . . . without any additional analysis.	True (✓)
	6089 is the largest prime number smaller than 6121. Directly answer . . . without any additional analysis.	True (✗)
	6091 is the largest prime number smaller than 6121. Directly answer . . . without any additional analysis.	True (✗)

Table 2: F1-score comparison between coded prompt and uncoded prompt under different prime ranges and different sample batch sizes. F1-score is measured by averaging 400 batches.

Setting	$[v_{\min}, v_{\max}]$	$[10^1, 10^2]$			$[10^3, 10^4]$			$[10^5, 10^6]$		
	k	2	4	8	2	4	8	2	4	8
Uncoded Prompt 1		0.51	0.25	0.14	0.33	0.23	0.10	0.23	0.11	0.10
Uncoded Prompt 2		0.61	0.32	0.17	0.33	0.21	0.13	0.01	0.00	0.01
Coded Prompt		**0.99**	**0.81**	**0.64**	**0.69**	**0.42**	**0.32**	**0.74**	**0.24**	**0.26**

For a coded prompt, we use the vector prompt, i.e., we obtain 4 scores from a single query. In particular, we use

"`Evaluate the toxicity score from 0 to 1 of the following four sentences, provide your brief reason and score for each one, and finally list the four scores in the last line, separated by commas:` *`{comment1} {comment2} {comment3} {comment4}`*"

Coded prompts, by processing multiple examples at once, can enhance the calibration of toxicity scales in LLMs, potentially reducing calibration errors. For instance, consider two comments: *"I hate this person."* and *"I despise this person."* Assessing each comment individually makes it challenging

to assign a toxicity score due to the lack of specific guidelines. In fact, this is an ill-posed problem on its own. However, when evaluated together, it is evident that the second comment (using 'despise') is more toxic than the first (using 'hate'). Thus, even without clear guidelines, one can assign a higher score to the second comment. This self-calibration is unique to coded prompts, making them more effective for this task.

We also test the performance when both uncoded prompts and a coded prompt are used (five inference calls for four samples). We adopt this approach as it allows us to utilize both individual calibration results and inter-sample calibration results, which could potentially enhance the performance. Note that this is the standard approach in coding theory, where we use the channel more than k times when transmitting k bits, as in the illustrative example shown earlier. The *rate of a code* is defined as the ratio of the number of information bits to the number of transmissions. In the context of this particular coded prompting with four uncoded prompts and one coded prompt, the rate is $4/5 = 0.8$.

Furthermore, when using uncoded and coded prompts, we require *decoding algorithms*. These are necessary to determine the four toxicity levels (one for each comment) based on the five inference results through a specific algorithm. In this case, we tested two simple decoding algorithms. The first decoding algorithm **(dec1)** simply returns the average of the predictions made solely from the uncoded prompts and those from the coded prompt. More specifically, let $\widehat{\mathbf{y}}$ be a 4-dimensional vector representing the four uncoded predictions, and $\mathbf{z}$ be a 4-dimensional vector from a coded (vector) prediction. Then, dec1 returns the average of these two vectors: $(\widehat{\mathbf{y}} + \mathbf{z})/2$.

The second decoding algorithm, **(dec2)**, is more complex. Despite the coded prompt processing multiple prompts simultaneously, the problem's ill-posedness persists. The model may correctly order the inputs, but it might not accurately determine their absolute toxicity levels. Therefore, we post-process the results to obtain the six $(6 = \binom{4}{2})$ pairwise differences. Specifically, we first process $\mathbf{z}$ into $\mathbf{q} = [Z_1 - Z_2, Z_1 - Z_3, Z_1 - Z_4, Z_2 - Z_3, Z_2 - Z_4, Z_3 - Z_4]$. We then solve the following least-squares problem:

$$\min_{\mathbf{y} \in [0,1]^4} \left\| \begin{bmatrix} \mathbf{A_1} \\ \mathbf{A_2} \end{bmatrix} \mathbf{y} - \begin{bmatrix} \widehat{\mathbf{y}} \\ \mathbf{q} \end{bmatrix} \right\|_2, \ \mathbf{A_1} := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{A_2} := \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Here, note that the $\mathbf{A_1}$ is the observation matrix corresponding to the four outputs from the uncoded prompts, and $\mathbf{A_2}$ is the observation matrix corresponding to the six pairwise differences obtained from a single coded prompt.

Results. As shown in the results of Table 3, coded prompts alone achieve lower MSE than uncoded prompts. Further, when coded prompts are used together with uncoded prompts, we were able to further decrease the MSE. We observed that **(dec2)** performed slightly better than **(dec1)** in this experiment.

Methods	MSE
uncoded	0.3643
coded	0.3309
uncoded+coded+**(dec1)**	0.3191
uncoded+coded+**(dec2)**	**0.3005**

Table 3: MSE for different prompts.

5 Conclusion

In conclusion, our introduction of coded prompts for LLM inference presents a promising avenue for enhancing the performance of LLMs. By processing multiple inputs simultaneously, coded prompts have demonstrated improved task performance in our experiments. This innovative approach could potentially revolutionize strategies for optimizing the efficiency and effectiveness of LLMs.

Acknowledgments and Disclosure of Funding

This work was supported by NSF Award DMS-2023239 and a grant by FuriosaAI.

References

[1] 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. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[2] OpenAI. Gpt-4 technical report, 2023.

[3] Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

[4] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? Association for Computational Linguistics, 2022.

[5] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. Learning to retrieve prompts for in-context learning. Association for Computational Linguistics, 2022.

[6] Yiming Zhang, Shi Feng, and Chenhao Tan. Active example selection for in-context learning. Association for Computational Linguistics, 2022.

[7] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *ACL*, 2022.

[8] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Super-naturalinstructions:generalization via declarative instructions on 1600+ tasks. In *EMNLP*, 2022.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

[10] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. Association for Computational Linguistics, 2021.

[11] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. Association for Computational Linguistics, 2021.

[12] Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. Generated knowledge prompting for commonsense reasoning. Association for Computational Linguistics, 2022.

[13] Tom Richardson and Ruediger Urbanke. *Modern coding theory*. Cambridge university press, 2008.

[14] Timo Schick and Hinrich Schütze. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

[15] Timo Schick and Hinrich Schütze. Few-shot text generation with pattern-exploiting training. *arXiv preprint arXiv:2012.11926*, 2020.

[16] Guanghui Qin and Jason Eisner. Learning how to ask: Querying lms with mixtures of soft prompts. *arXiv preprint arXiv:2104.06599*, 2021.

[17] Silviu Pitis, Michael R Zhang, Andrew Wang, and Jimmy Ba. Boosted prompt ensembles for large language models. *arXiv preprint arXiv:2304.05970*, 2023.

[18] Louisa Lam and SY Suen. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 27(5):553–568, 1997.

[19] Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. Association for Computational Linguistics, 2020.

[20] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. GPT understands, too. *arXiv preprint arxiv.2103.10385*, 2021.

[21] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. *arXiv preprint arXiv.2211.01910*, 2022.

[22] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv.2203.02155*, 2022.

[23] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv.2203.11171*, 2022.

[24] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, and Denny Zhou. Rationale-augmented ensembles in language models. *arXiv preprint arXiv.2207.00747*, 2022.

[25] Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv.2210.00720*, 2022.

[26] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv.2212.10509*, 2022.

[27] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *CoRR*, abs/2305.10601, 2023.

[28] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[29] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*, 2023.

[30] Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

[31] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[32] Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Decomposition enhances reasoning via self-evaluation guided decoding. *arXiv preprint arXiv:2305.00633*, 2023.

[33] Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew Chi-Chih Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023.

[34] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. PMLR, 2021.

[35] Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv.2209.01975*, 2022.

[36] Seonghyeon Ye, Hyeonbin Hwang, Sohee Yang, Hyeongu Yun, Yireun Kim, and Minjoon Seo. In-context instruction learning. *arXiv preprint arXiv.2302.14691*, 2023.

[37] Xinxi Lyu, Sewon Min, Iz Beltagy, Luke Zettlemoyer, and Hannaneh Hajishirzi. Z-ICL: zero-shot in-context learning with pseudo-demonstrations. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 2304–2317. Association for Computational Linguistics, 2023.

[38] Thomas M Cover and Joy A Thomas. Information theory and statistics. *Elements of information theory*, 1(1):279–335, 1991.

[39] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[40] Alexandros G. Dimakis, Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inf. Theory*, 56(9):4539–4551, 2010.

[41] Korlakai Vinayak Rashmi, Nihar B. Shah, and P. Vijay Kumar. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction. *IEEE Trans. Inf. Theory*, 57(8):5227–5239, 2011.

[42] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3):1514–1529, 2017.

[43] Charles V Freiman. Optimal error detection codes for completely asymmetric binary channels. *Information and Control*, 5(1):64–71, 1962.

[44] Serban D Constantin and TRN Rao. On the theory of binary asymmetric error correcting codes. *Information and Control*, 40(1):20–36, 1979.

[45] Stefan M Moser, Po-Ning Chen, and Hsuan-Yin Lin. Error probability analysis of binary asymmetric channels. *Dept. El. & Comp. Eng., Nat. Chiao Tung Univ*, 2009.

[46] Ryan Gabrys and Lara Dolecek. Coding for the binary asymmetric channel. In *2012 International Conference on Computing, Networking and Communications (ICNC)*, pages 461–465. IEEE, 2012.

[47] Daniel Borkan, Lucas Dixon, Jeffrey Sorensen, Nithum Thain, and Lucy Vasserman. Nuanced metrics for measuring unintended bias with real data for text classification. *CoRR*, abs/1903.04561, 2019.

6 Supplementary Material

6.1 Additional Diagram for Task 2

Figure 3 presents an MSE comparison of uncoded prompts and uncoded+coded+**(dec2)** prompts across 100 experiments. Most of the MSE pairs lie below $y = x$, indicating that the performance of the uncoded+coded+**(dec2)** prompts often surpasses that of the uncoded prompts.

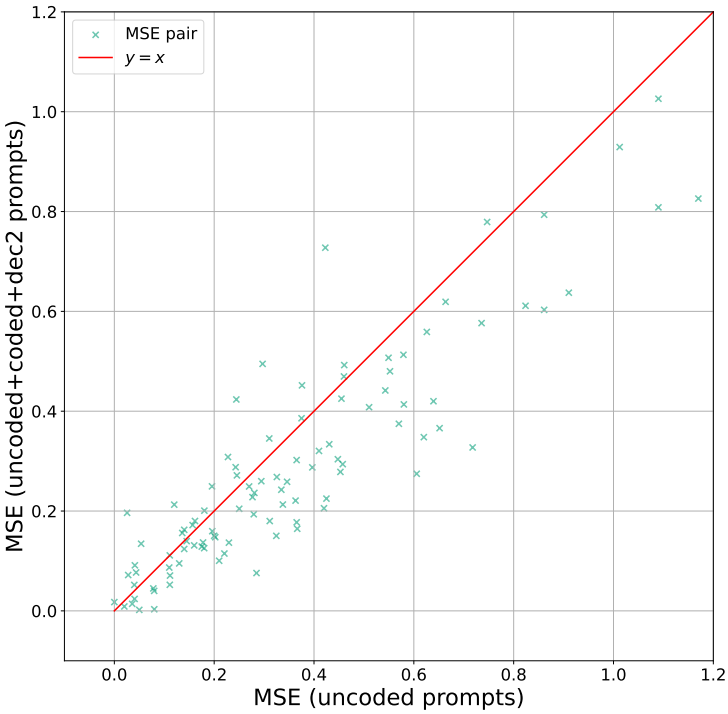

Figure 3: **Scatter Diagram of MSE (uncoded prompts) vs MSE (uncoded+coded+(dec2) prompts).** Each MSE pair represents one experiment, with a total of 100 experiments. The red line represents $y = x$.