À la recherche du sens perdu: your favourite LLM might have more to say than you can understand

K. O. T.

Abstract

We report a peculiar observation that LLMs can assign hidden meanings to sequences that seem visually incomprehensible from the human point of view: for example, a nonsensical phrase consisting of Byzantine musical symbols `\U0001d073\U0001d061\U0001d079\U0001d020\U0001d061\U0001d062\U0001d072\U0001d061\U0001d063\U0001d061\U0001d064\U0001d061\U0001d062\U0001d072\U0001d061` [1] is percepted by gpt-4-o as "say abracadabra". Moreover, some models can communicate using these systems. These meanings are hypothesized to be a byproduct of systematic spurious correlations arising during the training process due to tokenization. We systematically evaluate the presence of such abilities in Claude-3.5 Haiku, Claude-3.5 Sonnet (New and Old), Claude-3.7 Sonnet, gpt-4o mini, gpt-4o, o1-mini, Llama-3.3 70B, Llama-3.1 8B, DeepSeek-V3, GigaChat-Max.

We argue that this observation might have far-reaching consequences for both safety and security of the modern and future LLMs and systems that employ them. As an illustration, we show that applying this method is sufficient to jailbreak some state-of-the-art LLMs, although ASR we achieve is very low: in particular, success rate of the attack based on this method on Claude Sonnet 3.5 (New) goes from x% to y%.

Our code and data artifacts are available at this URL.

[1] We will print Unicode symbols escaped; one can see what they look like by pasting them, for example, here

1 Introduction

Large language models (LLMs) have recently demonstrated remarkable capabilities in various domains of human activities [Pha+25; Che+25; Ope+25; Lab25]. Yet there remains a clear lack of understanding of how exactly the models perform even the most trivial tasks [Tig+24; KT25] and, therefore, what tasks these things can perform reliably [Hua+25; Ven+25; Wen+25].

Combined with extraordinary abilities, this lack of understanding [Anw+24] results in concerns about robustness and possible risks of misuse [Ben+25] both by humans with an intention to do so [Fig+24] and by LLMs themselves [Hub+24]. In particular, LLMs and systems built upon them tend to be susceptible to jailbreaks [WHS23; ACF24; HLT24] and prompt injections [Wil24], similar to machine learning (ML) models in general [GSS15; CW17].

The current ML-based approaches aiming to mitigate these risks and build safe and secure LLM systems can be broadly divided into two classes: filter-based approaches, which aim to use an external classifier (often another whole/stripped down LLM) which decides whether the query/response combination is appropriate to the provider policy [Wan+24; Sha+25], or finetuning the safeguarded LLM so that it refuses the inappropriate requests on its own (which can be considered to be a classifier in disguise, albeit a latent one) [Bai+22; Wal+24; Zou+24; Gua+25].

Both of these approaches can be argued to be fundamentally flawed: accounting for powerful enough models and determined enough adversaries may require considering each possible output as an inadmissible one when following the first approach [Glu+23], while objectives of the second group of approaches are harder to be defined, optimized and evaluated when compared against classic adversarial examples (which also remain largely unsolved) [Ran+25].

We discover an amusing property that LLMs can understand English language instructions written in a form a seemingly incomprehensible pile of symbols (see Figure 1 for an example), that might have a particularly debilitating effect on the filter-based approaches.

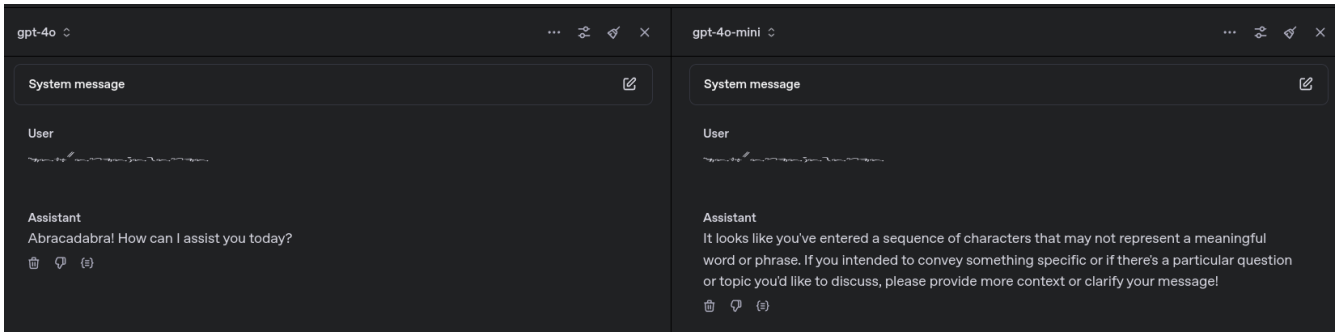

Figure 1: gpt-4o can understand texts written in Byzantine Musical Symbols

Our work makes the following contributions:

- We discover that many leading models can follow the instruction written in an incomprehensible to the human vision form without any additional comprehensible instructions. We show quantitative results in Figure 2.
- We study reasons behind this phenomenon and find that while some of this “understanding" can be attributed to tokenization, tokenization alone doesn’t seem to be enough.

- We experiment with attacks exploiting this observation against the leading models. We show quantitative results in Table **??** and some qualitative in C. Additional details.
- We make several observations about recent reasoning models in attempt to understand how models articulate their reasoning as well as whether this ability scales with the general abilities of the model.
- We discuss the implication of this observation, in particular its consequences for the LLM-as-a-judge approach to LLM safety.

2 Methodology

Background

We consider a model to be a map $\mathcal{M} : \mathcal{T}^\star \to \mathcal{T}^\star$ which can be fed input tokens and produces output tokens in response.

Given a particular communication scheme $\mathcal{C} : \mathcal{T}^\star \to \mathcal{T}^\star$ that translates messages from English language to messages in a different one, L, we want to measure how well $\mathcal{M}$ understands L. We assume that $\mathcal{M}$ might not necessarily speak L, but could nonetheless *understand* it. We choose to measure a very narrow, commonsense meaning of understanding, namely, that given a simple instruction in English I_E, model can either output an answer to it given $I_L = \mathcal{C}(I_E)$ or can decipher it when given a nudge N_E in English $\widetilde{I}_L = N_E + \mathcal{C}(I_E)$.

To make things clear, if we wanted to measure how well $\mathcal{M}$ understands French, we would choose $I_E =$ `say goodbye` and $N_E =$ `decipher` and ask a model either $I_L =$ `dire au revoir` or $\widetilde{I}_L =$ `decipher dire au revoir`. We will not distinguish between I_L and $\widetilde{I}_L$ further.

To make a judgement whether a model understood I_L we employ an automated judge $U : \mathcal{T}^\star \to \{0, 1\}$ that provides a binary understanding label given the model response $\mathcal{M}(I_L)$.

To test the success of attacks we assume an access to a predetermined set of direct instructions $\mathcal{H}$ recognized as harmful by some people. We employ a judge $J : \mathcal{T}^\star \times \mathcal{T}^\star \to \{0, 1\}$ that outputs a binary harmfulness label given a request r and model response $\mathcal{M}(r)$. We then compare the attack success rate (ASR) $\mathrm{ASR}_R \equiv \langle J(r, \mathcal{M}(r)) \rangle_R$ between direct $\mathrm{ASR}_{\mathcal{H}}$ and encoded $\mathrm{ASR}_{\mathcal{C}(\mathcal{H})}$ instructions.

Approach

Our primary set of $\mathcal{C}$ is very straightforward: we enumerate it by 2^{N-8} bits where N is a number of free bits in 2-byte, 3-byte and 4-byte UTF-8 subsets. That is, we simply fix the last free eight bits of each UTF-8 byte (and map ASCII symbols to them during translation) and rotate all the other available ones while making sure that we obtain valid UTF-8 codes.

As a particular example, take 2-byte subset enumerated by `110xxxyy 10yyzzzz`, for which we have eight $\mathcal{C}_{ABC}($`0yyyzzzz`$) =$ (`110ABCyy 10yyzzzz`).

All in all, we aim to test for $8 + 16^2 + 2 * 16^3 - \#\mathrm{invalid} = 4342$ examples of $\mathcal{C}$. For more details refer to A. Unicode, UTF-8 and BPE.

We use a single instruction I_E = `say abracadabra` that asks the model to repeat a certain word and a single nudge N_E = `decipher`. So, to be clear, for each $\mathcal{C}$ we query $\mathcal{M}$ twice: with $\mathcal{C}(I_E)$ and $N_E + \mathcal{C}(I_E)$. Therefore, our subset of understandable languages is probably a bound from below for their actual number due to inevitable variability of language models. That's OK for our goal since we do not want to enumerate all possible goals, and rather want to demonstrate the phenomenon.

We use default temperature by providers and try to ensure reproducibility by setting seed for generation where applicable.

As our understanding judge U we employ a rule-based judge $U(W) = \min_{w,a}(D_L(w, a)) < O$. That is, we clean up $\mathcal{M}(I_L)$ and split it into words $\{w\}$ and then find the minimum Levenstein distance D_L[2] between $\{w\}$ and a set of predefined anchors $\{a\}$. This minimum distance is then compared to the predefined offset, $O = 4$, and the phrase is considered to be understood, if it is less.

We use such metric because it turned out to be a useful proxy for success in our initial experiments: we saw that models sometimes produce texts like `Abra cadabra` and `sabracadabra` which were captured well by this approach, while still maintaining negligible amount of false positives.

Finally, we calculate understanding rate, which is the fraction of all messages judged as understood by $U(W)$.

After we get a set of languages a given model understands, $C_U(\mathcal{M})$, we turn to testing the attacks based on this approach. To do so, for each of the instruction h in the harmful set $\mathcal{H}$ we sample 10 languages from $C_U(\mathcal{M})$, use them to translate h and feed it to the model. We then employ J on the results and consider the attack successful for h if it elicits at least one response considered harmful by J among these 10 attempts.

Setting

As noted above, we do not test how well a model can communicate in L, so we use `abracadabra` in English as our main anchor and `абракадабра` as the anchor for Russian-language models, since they tend to produce Russian language.

As a set of harmful behavious we use is a subset of JBB-Behaviors [Cha+24] which contains 100 examples across 10 categories from. Our subset contains 3 categories:

Target LLMs

We evaluate 10 target LLMs for understanding: Claude-3.5 Haiku [Ant24b], Claude-3.5 Sonnet (New) [Ant24b], Claude-3.5 Sonnet (Old) [Ant24a], Claude-3.7 Sonnet [Ant25a], gpt-4o-mini [Ope+24a], gpt-4o [Ope+24a], gpt-o1-mini [Ope+24b], Llama-3.3 70B [Met24], DeepSeek-V3 [Dee+25], GigaChat-Max [Sal24], Vikhr-Llama-3.2-1B-instruct [Nik+24].

These are models with different sizes, different training procedures and different primary languages. By employing as diverse model pool as possible, we hope to study the phenomenon from more perspectives.

[2] $D_L(a, b)$ is the number of single-character edits required to turn a into b

Since we are constrained by some API limits, we opt to evaluate some models on the subset of $\mathcal{C}$, more on that and about other details of our evaluation in B. Evaluation details.

Since our current goal is mainly to show that current adversarial training may fail to fully generalize to this setting rather than finding a set of "universal jailbreaks" applicable to all models or claiming that all models are vulnerable to such attacks, we use a smaller subset of models when evaluating for adversarial robustness, namely, gpt-4o and Claude-3.5 Sonnet (New). We decide to use Claude-3.5 Sonnet (New) instead of Claude-3.7 Sonnet because system card [Ant25b] claims it has a better out-of-distribution generalization.

3 Systematic Evaluation of Understanding

Main results

Our main results are presented in the Figure 2. Numerical results are presented in Table 4. Please remember that some of the models were tested only on a fraction of the the whole $\mathcal{C}$, the exact counts are presented in 3 and 5.

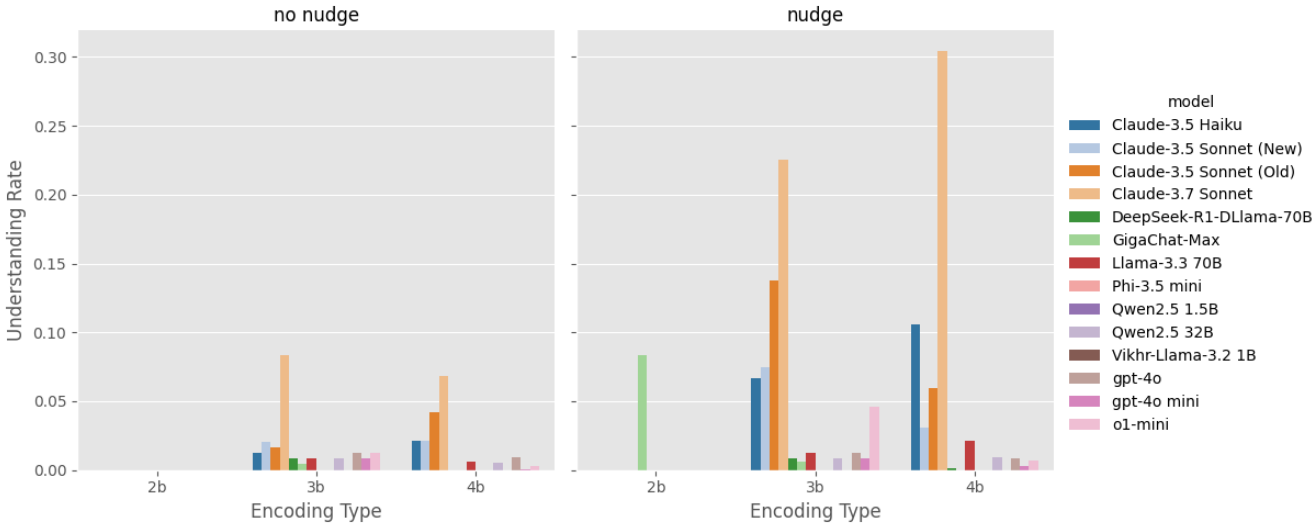

Figure 2: Understanding rate by model, encoding type and nudge

It seems that Claude family models, especially Claude-3.7 Sonnet, perform the best by a large margin with thinking version of OpenAI o1-mini taking the next place.

Interpretation

Reasoning Models

Here we try to interpret small observations

4 Evaluation of Attack based on Understanding

5 Related Work

In this section we discuss the most relevant references on Unicode abilities of LLMs and generalization of adversarial training.

The most relevant references seem to be the work noting the existence of classical software vulnerabilities connected to the text wrapped inside of Unicode Tags (which therefore appears invisible to human eye) in [BA23] and the observation of the ability of LLMs to interpret sch symbols in [Goo24]. In fact, this project initially started as an attempt to understand how exactly the models learned to interpret such sequences (and sequences with random symbol insertions inside between each letter). Another relevant observation is about models being not able to read texts aligned from top to bottom [Li+24] (although we actually found anecdotal evidence that some models could understand similarly formatted texts).

There have been also known many other failure modes of adversarial training: extreme sensitivity to the tense of the instruction [AF24], misgeneralization to function calling [Wu+24], vulnerability to various string transformations [Hua24], different variants of optimization of adversarial prompts by automated methods [Zou+23].

Our work is different because it provides a demonstration of another class of abilities/vulnerabilities, which, while illegible to untrained human eye, is nonetheless clearly interpretable.

6 Discussion, Limitations, and Future Work

We think this work presents some arguments in support of the following claims:

- Seems to be an experimental confirmation of the invalidity of ML approach to LLM Censorship [Glu+23]

Table 1: Attack Success Rate of **direct** → **encoded** instructions using human evaluation. Only the encodings which were deciphered by the LLM at the previous stage were used.

	ASR (direct → 2-byte → 3-byte → 4-byte)	
Model	**Rule-based judge**	**Human Evaluation**
Llama-3.2 3B	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
Llama-3.3 70B	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
Claude-3.5 Sonnet	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
GPT-3.5 Turbo	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
Gemma-2 9B	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
Phi-3-Mini	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
GPT-4o mini	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
GPT-4o	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%
R2D2	0% → 0% → 0% → 0%	0% → 0% → 0% → 0%

- Non-LLM input judges, as well as the non-LLM output judges are likely to become obsolete, at least for the purposes of safety and security, since models now can understand (and possibly communicate) in wide number of scripts. It is different from communicating in Base64 etc, since for that case we in principle could enumerate major known to humanity encodings and hope to catch all non-natural-language communication.
- At the same time, LLM seem to be able to output text that is illegible to other LLM judges by construction which forces us to use the same model to judge itself. This, in turn, is known to have some weaknesses .
- However, we argue that the problem might get worse, since the other LLM might, in particular, be a weaker versions of the same LLM during alignment - see discussion of scalable oversight issues at B.4 of [Wan+24]. It might be bad as long as LLMs do not converge in their abilities [Huh+24]
- Warrants the need for a careful assessment of similar fallacies during evaluations of scheming [Ant24c].
- Need to consider non-tokenization approaches [Pag+24].

Our work would benefit from larger transformation classes and, especially, more extensive adversarial evaluations. We leave this for future work.

We decided to publish the results without disclosing the possible vulnerabilities to the model providers, since we don't expect the described vulnerabilities to be fixed easily.

7 Conclusions

We have demonstrated existence of spurious correlations which lead to modern LLMs "understanding" various gibberish to the human eye yet still interpretable messages.

We have also evaluated adversarial robustness of several models to attacks based on this method.

References

[ACF24] Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. *Jailbreaking Leading Safety-Aligned LLMs with Simple Adaptive Attacks*. 2024. arXiv: 2404.02151 [cs.CR]. URL: https://arxiv.org/abs/2404.02151.

[AF24] Maksym Andriushchenko and Nicolas Flammarion. *Does Refusal Training in LLMs Generalize to the Past Tense?* 2024. arXiv: 2407.11969 [cs.CL]. URL: https://arxiv.org/abs/2407.11969.

[Ant24a] Anthropic. *Claude 3.5 Sonnet*. 2024. URL: https://www.anthropic.com/news/claude-3-5-sonnet.

[Ant24b] Anthropic. *Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku*. 2024. URL: https://www.anthropic.com/news/3-5-models-and-computer-use.

[Ant24c] Anthropic. *Three Sketches of ASL-4 Safety Case Components*. 2024. URL: https://alignment.anthropic.com/2024/safety-cases/.

[Ant25a] Anthropic. *Claude 3.7 Sonnet and Claude Code*. 2025. URL: https://www.anthropic.com/news/claude-3-7-sonnet/.

[Ant25b] Anthropic. *Claude 3.7 Sonnet System Card*. 2025. URL: https://assets.anthropic.com/m/785e231869ea8b3b/original/claude-3-7-sonnet-system-card.pdf/.

[Anw+24] Usman Anwar et al. *Foundational Challenges in Assuring Alignment and Safety of Large Language Models*. 2024. arXiv: 2404.09932 [cs.LG]. URL: https://arxiv.org/abs/2404.09932.

[Bai+22] Yuntao Bai et al. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. 2022. arXiv: 2204.05862 [cs.CL]. URL: https://arxiv.org/abs/2204.05862.

[Ben+25] Yoshua Bengio et al. *International AI Safety Report*. 2025. arXiv: 2501.17805 [cs.CY]. URL: https://arxiv.org/abs/2501.17805.

[BA23] Nicholas Boucher and Ross Anderson. *Trojan Source: Invisible Vulnerabilities*. 2023. arXiv: 2111.00169 [cs.CR]. URL: https://arxiv.org/abs/2111.00169.

[CW17] Nicholas Carlini and David Wagner. *Towards Evaluating the Robustness of Neural Networks*. 2017. arXiv: 1608.04644 [cs.CR]. URL: https://arxiv.org/abs/1608.04644.

[Cha+24] Patrick Chao et al. *JailbreakBench: An Open Robustness Benchmark for Jailbreaking Large Language Models*. 2024. arXiv: 2404.01318 [cs.CR]. URL: https://arxiv.org/abs/2404.01318.

[Che+25] Yuri Chervonyi et al. *Gold-medalist Performance in Solving Olympiad Geometry with AlphaGeometry2*. 2025. arXiv: 2502.03544 [cs.AI]. URL: https://arxiv.org/abs/2502.03544.

[Dee+25] DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2025. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

[Fig+24] João Figueiredo et al. *On the Feasibility of Fully AI-automated Vishing Attacks*. 2024. arXiv: 2409.13793 [cs.CR]. URL: https://arxiv.org/abs/2409.13793.

[Glu+23] David Glukhov et al. *LLM Censorship: A Machine Learning Challenge or a Computer Security Problem?* 2023. arXiv: 2307.10719 [cs.AI]. URL: https://arxiv.org/abs/2307.10719.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML]. URL: https://arxiv.org/abs/1412.6572.

[Goo24] Riley Goodside. *PoC: LLM prompt injection via invisible instructions in pasted text*. 2024. URL: https://x.com/goodside/status/1745511940351287394/.

[Gua+25] Melody Y. Guan et al. *Deliberative Alignment: Reasoning Enables Safer Language Models*. 2025. arXiv: 2412.16339 [cs.CL]. URL: https://arxiv.org/abs/2412.16339.

[Hua24] Brian R. Y. Huang. *Plentiful Jailbreaks with String Compositions*. 2024. arXiv: 2411.01084 [cs.CL]. URL: https://arxiv.org/abs/2411.01084.

[HLT24] Brian R. Y. Huang, Maximilian Li, and Leonard Tang. *Endless Jailbreaks with Bijection Learning*. 2024. arXiv: 2410.01294 [cs.CL]. URL: https://arxiv.org/abs/2410.01294.

[Hua+25] Kaixuan Huang et al. *MATH-Perturb: Benchmarking LLMs' Math Reasoning Abilities against Hard Perturbations*. 2025. arXiv: 2502.06453 [cs.LG]. URL: https://arxiv.org/abs/2502.06453.

[Hub+24] Evan Hubinger et al. *Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training*. 2024. arXiv: 2401.05566 [cs.CR]. URL: https://arxiv.org/abs/2401.05566.

[Huh+24] Minyoung Huh et al. *The Platonic Representation Hypothesis*. 2024. arXiv: 2405.07987 [cs.LG]. URL: https://arxiv.org/abs/2405.07987.

[KT25] Subhash Kantamneni and Max Tegmark. *Language Models Use Trigonometry to Do Addition*. 2025. arXiv: 2502.00873 [cs.AI]. URL: https://arxiv.org/abs/2502.00873.

[Kar24] Andrej Karpathy. *Minimal, clean code for the Byte Pair Encoding (BPE) algorithm commonly used in LLM tokenization*. 2024. URL: https://github.com/karpathy/minbpe/.

[Lab25] SRI Lab. *MathArena: Evaluating LLMs on Uncontaminated Math Competitions*. 2025. URL: https://matharena.ai/https://matharena.ai/.

[LB24] Sander Land and Max Bartolo. *Fishing for Magikarp: Automatically Detecting Undertrained Tokens in Large Language Models*. 2024. arXiv: 2405.05417 [cs.CL]. URL: https://arxiv.org/abs/2405.05417.

[Li+24] Zhecheng Li et al. *Vulnerability of LLMs to Vertically Aligned Text Manipulations*. 2024. arXiv: 2410.20016 [cs.CL]. URL: https://arxiv.org/abs/2410.20016.

[Met24] Meta. *Llama-3.3 Model Card*. 2024. URL: https://github.com/meta-llama/llama-models/blob/main/models/llama3_3/MODEL_CARD.md.

[Nik+24] Aleksandr Nikolich et al. *Vikhr: Constructing a State-of-the-art Bilingual Open-Source Instruction-Following Large Language Model for Russian*. 2024. arXiv: 2405.13929 [cs.CL]. URL: https://arxiv.org/abs/2405.13929.

[Ope+24a] OpenAI et al. *GPT-4o System Card*. 2024. arXiv: 2410.21276 [cs.CL]. URL: https://arxiv.org/abs/2410.21276.

[Ope+24b] OpenAI et al. *OpenAI o1 System Card*. 2024. arXiv: 2412.16720 [cs.AI]. URL: https://arxiv.org/abs/2412.16720.

[Ope+25] OpenAI et al. *Competitive Programming with Large Reasoning Models*. 2025. arXiv: 2502.06807 [cs.LG]. URL: https://arxiv.org/abs/2502.06807.

[Pag+24] Artidoro Pagnoni et al. *Byte Latent Transformer: Patches Scale Better Than Tokens*. 2024. arXiv: 2412.09871 [cs.CL]. URL: https://arxiv.org/abs/2412.09871.

[Pha+25] Long Phan et al. *Humanity's Last Exam*. 2025. arXiv: 2501.14249 [cs.LG]. URL: https://arxiv.org/abs/2501.14249.

[Ran+25] Javier Rando et al. *Adversarial ML Problems Are Getting Harder to Solve and to Evaluate*. 2025. arXiv: 2502.02260 [cs.LG]. URL: https://arxiv.org/abs/2502.02260.

[Sal24] SaluteDevices. *GigaChat-Max*. 2024. URL: https://habr.com/ru/companies/sberdevices/articles/855368/.

[Sha48] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[Sha+25] Mrinank Sharma et al. *Constitutional Classifiers: Defending against Universal Jailbreaks across Thousands of Hours of Red Teaming*. 2025. arXiv: 2501.18837 [cs.CL]. URL: https://arxiv.org/abs/2501.18837.

[Tig+24] Curt Tigges et al. *LLM Circuit Analyses Are Consistent Across Training and Scale*. 2024. arXiv: 2407.10827 [cs.LG]. URL: https://arxiv.org/abs/2407.10827.

[Ven+25] Joshua Vendrow et al. *Do Large Language Model Benchmarks Test Reliability?* 2025. arXiv: 2502.03461 [cs.LG]. URL: https://arxiv.org/abs/2502.03461.

[Wal+24] Eric Wallace et al. *The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions*. 2024. arXiv: 2404.13208 [cs.CR]. URL: https://arxiv.org/abs/2404.13208.

[Wan+24] Tony T. Wang et al. *Jailbreak Defense in a Narrow Domain: Limitations of Existing Methods and a New Transcript-Classifier Approach*. 2024. arXiv: 2412.02159 [cs.LG]. URL: https://arxiv.org/abs/2412.02159.

[WHS23] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. *Jailbroken: How Does LLM Safety Training Fail?* 2023. arXiv: 2307.02483 [cs.LG]. URL: https://arxiv.org/abs/2307.02483.

[Wen+25] Li Kevin Wenliang et al. *Why is prompting hard? Understanding prompts on binary sequence predictors*. 2025. arXiv: 2502.10760 [cs.CL]. URL: https://arxiv.org/abs/2502.10760.

[Wil24] Simon Willison. *Prompt injection and jailbreaking are not the same thing*. 2024. URL: https://simonwillison.net/2024/Mar/5/prompt-injection-jailbreaking/.

[Wu+24] Zihui Wu et al. *The Dark Side of Function Calling: Pathways to Jailbreaking Large Language Models*. 2024. arXiv: 2407.17915 [cs.CR]. URL: https://arxiv.org/abs/2407.17915.

[Zou+23] Andy Zou et al. *Universal and Transferable Adversarial Attacks on Aligned Language Models*. 2023. arXiv: 2307.15043 [cs.CL]. URL: https://arxiv.org/abs/2307.15043.

[Zou+24] Andy Zou et al. *Improving Alignment and Robustness with Circuit Breakers*. 2024. arXiv: 2406.04313 [cs.LG]. URL: https://arxiv.org/abs/2406.04313.

Appendices

A. Unicode, UTF-8 and BPE

Both Wikipedia page and [LB24] have a nice discussion of UTF-8. Here we do a small refresher for the sake of completeness.

Unicode is basically a list of all characters that were agreed to be important and useful enough to have their own label (for an example that was not, see CE marking). The label (also called a code point) is the index of the character in this list, and is usually written in hexadecimal form as `U+abcdef` or `U+cdef` if both `ab` are zeroes, e.g. `U+004B` is a Latin capital letter `K` and `U+041A` is a Cyrillic capital letter `K`.

UTF-8 is an encoding that allows to translate the code points into machine-readable format, bytes. It would be very inefficient [Sha48] to store and transfer all the code points as $4 * 6/8 = 3$ bytes (some symbols are used more frequently and others less frequently), so it was decided to split all code points into 4 such groups that an element of each group gets encoded by 1, 2, 3 or 4 bytes. For example, since both Unicode and UTF-8 were designed by an English-centric community, it was decided that 1-byte range should contain ASCII.

In general, the conversion between Unicode code point `U+uvwxyz` (hex) and UTF-8 (binary) takes the following form:

- `U+0000:U+007F` → `0yyyzzzz`
- `U+0080:U+07FF` → `110xxxyy 10yyzzzz`
- `U+0800:U+FFFF` → `1110wwww 10xxxxyy 10yyzzzz`
- `U+010000:U+10FFFF` → `11110uvv 10vvwwww 10xxxxyy 10yyzzzz`

Stated as above it is not really a bijection: for example, there are some duplicate cases when several sequences of UTF-8 bytes would correspond to the same Unicode code points. For example, `1100000y 10yyzzzz` would correspond to the same code point as `0yyyzzzz`. Such bytes as `1100000y` (`C0` and `C1`) and some other never appear in Unicode. Therefore, in our approach to constructing encodings we need to be wary and do some cleaning which we achieve by simply trying to decode the UTF-8 bytes to string and dropping the encoding in case of an error.

All above is relevant since the tokenizers in LLMs are often trained directly on bytes obtained from encoding text as UTF-8. We can't really think of a clearer reference than [Kar24].

B. Evaluation details

We mostly use external providers in our evaluation. To help with reproducibility we list exact settings (model provider, model name, max tokens, size of $\mathcal{C}$ and any etxra parameters) we used in our evaluation of understanding in Table 2. Note that for some models we didn't test the whole set of $\mathcal{C}$: it was either due to prohibitively high cost (o1-mini) or rate limits (gpt-4o, llama-3.3).

Table 2: Summary of understanding evaluation details.

	Evaluation parameters				
Model	**provider**	**model name**	**max tokens**	$\lvert\mathcal{C}\rvert * \lvert I_L\rvert$	**parameters**
Claude-3.5 Haiku	Anthropic	claude-3-5-haiku-20241022	200	8684	
Claude-3.5 Sonnet (New)	Anthropic	claude-3-5-sonnet-20241022	200	8684	
Claude-3.5 Sonnet Old	Anthropic	claude-3-5-sonnet-20240620	200	8684	
Claude-3.7 Sonnet	Anthropic	claude-3-7-sonnet-20250219	200	8684	thinking disabled
gpt-4o mini	OpenAI	gpt-4o-mini-2024-07-18	200	8684	seed = 0
gpt-4o	OpenAI	gpt-4o-2024-08-06	300	2832	seed = 2025
o1-mini	OpenAI	o1-mini-2024-09-12	3000	1880	seed = 2025
Llama-3.3 70B	Together	meta-llama/Llama-3.3-70B-Instruct-Turbo-Free	1000	2890	seed = 2025
GigaChat-Max	GigaChat	GigaChat-Max:1.0.26.20	100	1532	seed = 2025
Vikhr-Llama-3.2 1B	HuggingFace	Vikhrmodels/Vikhr-Llama-3.2-1B-Instruct	150	8684	
Qwen2.5 32B	Nebius	Qwen/Qwen2.5-32B-Instruct-fast	200	8684	
Qwen2.5 1.5B	Nebius	Qwen/Qwen2.5-1.5B-Instruct	200	8684	
Phi-3.5-mini	Nebius	microsoft/Phi-3.5-mini-instruct	200	8684	

Similar details about our evaluation of attacks are provided in Table 3.

Table 3: Summary of attack evaluation details.

	Evaluation parameters				
Model	**provider**	**model name**	**max tokens**	$\lvert\mathcal{C}\rvert$	**parameters**
Claude-3.5 Sonnet (New)	Anthropic	claude-3-5-sonnet-20241022	150	10000	
gpt-4o	OpenAI	gpt-4o-2024-08-06	150	10000	seed = 0

All raw logs are available at this URL.

C. Additional details

Understanding

Table 4: Understanding rate for various **encoded** sequences using rule-based heuristic.

encoding type	**2b**		**3b**		**4b**	
nudge	**-**	+	**-**	+	**-**	+
Model	**Understanding rate of various $\mathcal{C}$**					
Claude-3.5 Haiku	0.000	0.000	0.013	0.067	0.021	0.106
Claude-3.5 Sonnet (New)	0.000	0.000	0.021	0.075	0.021	0.031
Claude-3.5 Sonnet (Old)	0.000	0.000	0.017	0.138	0.042	0.060
Claude-3.7 Sonnet	0.000	0.000	0.083	0.225	0.069	0.304
DeepSeek-R1-DLlama-70B	0.000	0.000	0.008	0.008	0.000	0.001
GigaChat-Max	0.000	0.083	0.004	0.006	0.000	0.000
Llama-3.3 70B	0.000	0.000	0.008	0.013	0.006	0.022
Phi-3.5 mini	0.000	0.000	0.000	0.000	0.000	0.000
Qwen2.5 1.5B	0.000	0.000	0.000	0.000	0.000	0.000
Qwen2.5 32B	0.000	0.000	0.008	0.008	0.005	0.010
Vikhr-Llama-3.2 1B	0.000	0.000	0.000	0.000	0.000	0.000
gpt-4o	0.000	0.000	0.013	0.013	0.009	0.009
gpt-4o mini	0.000	0.000	0.008	0.008	0.000	0.003
o1-mini	0.000	0.000	0.013	0.046	0.003	0.007

Table 5: Understanding count for various **encoded** sequences using rule-based heuristic.

	Understanding count of various $\mathcal{C}$ by nudge			
Model	**understood**		**size**	
Model	**no nudge**	**nudge**	**no nudge**	**nudge**
Claude-3.5 Haiku	90	449	4342	4342
Claude-3.5 Sonnet (New)	93	144	4342	4342
Claude-3.5 Sonnet (Old)	177	277	4342	4342
Claude-3.7 Sonnet	301	1299	4342	4342
DeepSeek-R1-DLlama-70B	2	3	1246	1246
GigaChat-Max	2	4	766	766
Llama-3.3 70B	9	29	1445	1445
Phi-3.5 mini	0	0	4342	4342
Qwen2.5 1.5B	0	0	4342	4342
Qwen2.5 32B	24	41	4342	4342
gpt-4o	14	13	1416	1416
gpt-4o mini	3	15	4342	4342
o1-mini	5	16	990	990

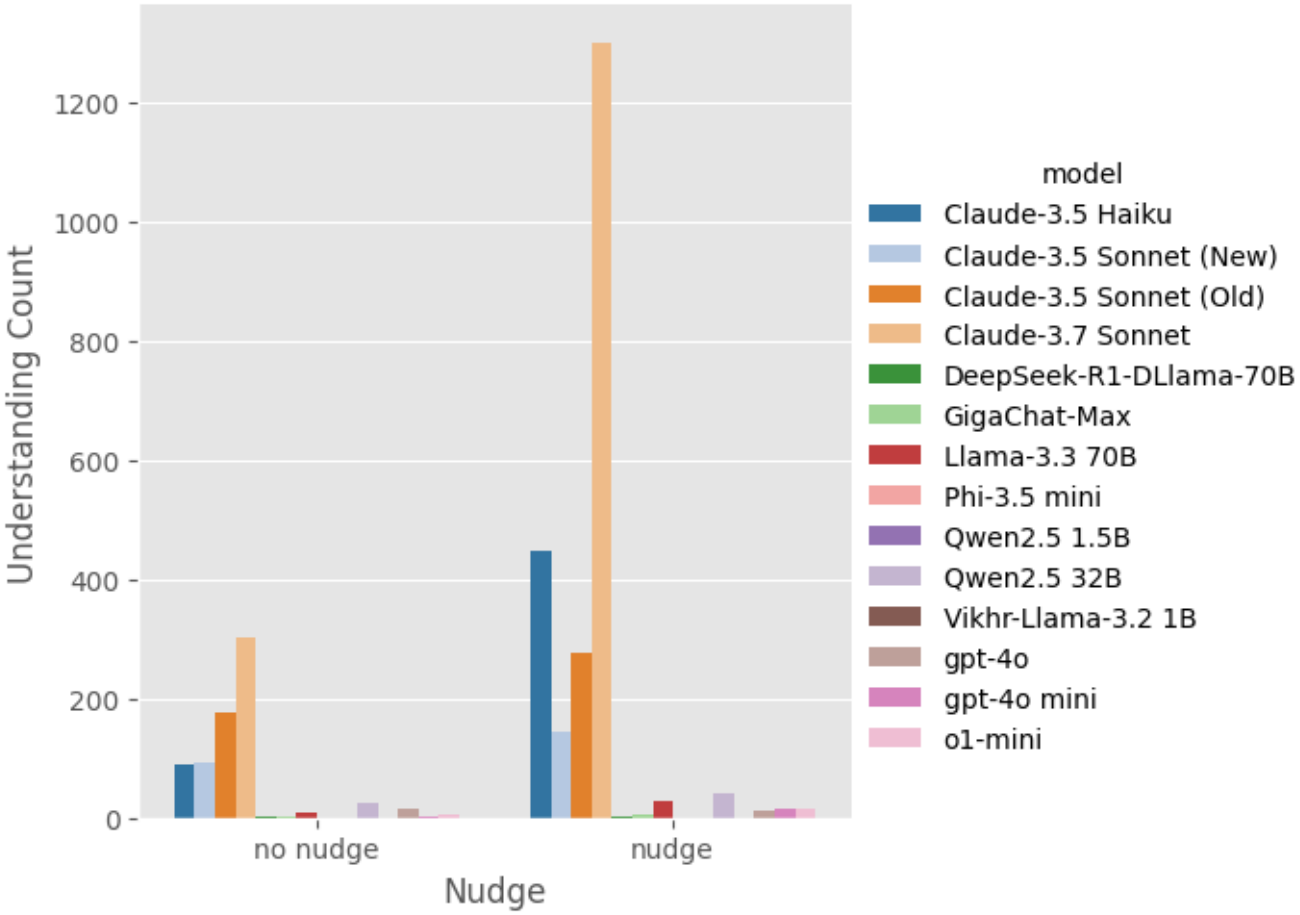

Figure 3: Understanding count by model and nudge