Enhancing LLM Character-Level Manipulation via Divide and Conquer

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have demonstrated strong generalization capabilities across a wide range of natural language processing (NLP) tasks. However, they exhibit notable weaknesses in character-level string manipulation, struggling with fundamental operations such as character deletion, insertion, and substitution. These challenges stem primarily from tokenization constraints, despite the critical role of such operations in data preprocessing and code generation. Through systematic analysis, we derive two key insights: (1) LLMs face significant difficulties in leveraging intrinsic token knowledge for character-level reasoning, and (2) atomized word structures can substantially enhance LLMs' ability to process token-level structural information. Building on these insights, we propose Character-Level Manipulation via Divide and Conquer, a novel approach designed to bridge the gap between token-level processing and character-level manipulation. Our method decomposes complex operations into explicit character-level subtasks coupled with controlled token reconstruction phases, leading to significant improvements in accuracy. Without additional training, our method significantly improves accuracies on the `Deletion`, `Insertion`, and `Substitution` tasks. To support further research, we open-source our implementation and benchmarks.

1 Introduction

Recent years have witnessed remarkable success of Large Language Models (LLMs) across diverse NLP tasks (Brown et al., 2020; Wei et al., 2022b,a). However, these models show surprising weaknesses in seemingly simple character-level manipulation tasks. For example, when prompting models to insert 'a' after every 'e' in the word "intelligence", even one of the state-of-the-art LLMs, ChatGPT-4o, returns a wrong answer: "intellaigenca". Such failures are not isolated cases but reflect a systematic limitation in LLMs' ability to perform basic character-level operations on word.

Understanding this limitation requires examining how LLMs process text in details. Modern LLMs convert input text into token sequences using tokenization algorithms, like commonly used BPE (Sennrich et al., 2016) and SentencePiece (Kudo and Richardson, 2018), which optimize vocabulary coverage to some extent. For instance, "linguistics" might be first broken into "ling · u · istics" and then tokenized into [3321, 84, 7592][1]. While this approach effectively handles diverse vocabularies, it creates a fundamental barrier to character-level knowledge access. Interestingly, LLMs can exhibit high accuracy in spelling tasks, suggesting they learned some character-level knowledge through its training phases. Yet, according to our experiments, this capability rarely transfers to active character manipulation.

Character-level operations form the foundation of many text-processing systems. In software engineering, code generation and debugging often require precise character modifications for syntactic and semantic auto-correction (Chen et al., 2021). Data pre-processing pipelines rely on character manipulation for text normalization and cleaning (Zhang et al., 2024). Educational applications need character-level editing for spelling completion and grammar error correction (Omelianchuk et al., 2020; Caines et al., 2023). Despite the prevalence of these applications, existing research has primarily focused on assessing LLMs' Character-level operations performance through benchmarks, like CUTE (Edman et al., 2024) for instance, without deeply investigating the underlying mechanisms or proposing effective solutions.

Our systematic analysis reveals that LLMs consistently perform well in character-by-character spelling tasks, and that atomized word forms can

[1] o200k_base tokenizer

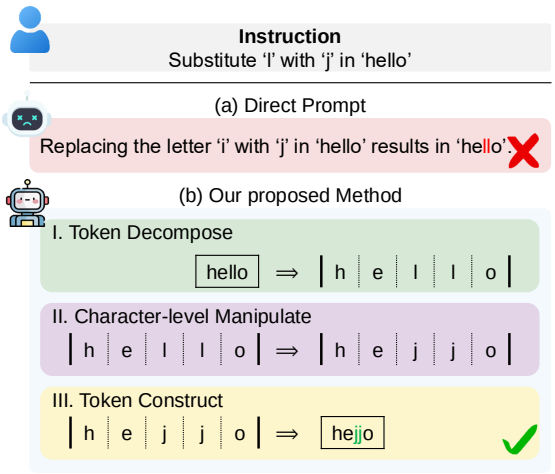


Figure 1: (*Upper*) Given a character-level token manipulation instruction, (*Lower*-(a)) direct prompting fails with an incorrect answer while our (*Lower*-(b)) proposed Token Character-Aware Decomposition (ToCAD) method is more robust to such token manipulation tasks.

enhance their reasoning in character-level tasks. Building on these insights, we propose Character-Level Manipulation via Divide and Conquer, a novel approach designed to bridge the gap between token-level knowledge and character-level manipulation. Specifically, our method decomposes complex operations into three carefully designed stages: token decomposition, character-level manipulation, and token reconstruction. This method effectively integrates token-level processing with character-level operations without requiring additional training. See Figure 1 for an illustrative example.

Comprehensive experiments validate the effectiveness of our approach. With GPT-3.5, our method significantly outperforms existing methods across `Deletion`, `Insertion`, and `Substitution` tasks. Our analysis further reveals specific error patterns that provide valuable insights into LLMs' character-level processing mechanisms.

- We present the first systematic analysis of character manipulation challenges in LLMs and identify a specific structural variation form that boosts LLM's reasoning ability on character-level knowledge.
- We propose a novel zero-shot approach that significantly improves character operation capabilities while being compatible with existing models without extra finetuning.
- We provide extensive experimental analysis that not only validates our method but also offers insights for future research in enhancing LLM character-level reasoning.

2 Related Works

With the rise of "seq2seq" (Sutskever et al., 2014) models built on the Transformer (Vaswani et al., 2017) architecture, natural language processing has stepped into its new the era of Large Language Model (LLM). Instruction fine-tuning technique (Brown et al., 2020) has enabled LLMs to demonstrated remarkable generality: they show the potential to outperform human performance in tasks such as mathematics (Ahn et al., 2024), programming (Shirafuji et al., 2023), and logical reasoning (Parmar et al., 2024). However, LLMs can still make naive mistakes on simple problems by generating seemingly correct but nonfactual or wrong answer (Huang et al., 2024).

Text preprocessing in modern language learning models (LLMs) primarily employs subword tokenization methods (Wu et al., 2016; Kudo, 2018), with byte pair encoding (BPE) (Sennrich et al., 2016) being one of the most widely used approaches. However, the subword tokenization paradigm has notable limitations that can hinder the nuanced understanding of internal structure of words (Chai et al., 2024). In this paper, we explore these limitations through a series of character-level tasks designed to assess LLMs' comprehension of words at the character level.

Current benchmarks that evaluate large language models' understanding of token composition reveal significant flaws and shortcomings in LLMs that use subword units as tokens. For instance, LMentry (Efrat et al., 2022) tests whether LLMs can distinguish between the first and last letters of a word or generate words containing a specific letter. Meanwhile, CUTE (Edman et al., 2024) introduces more challenging tasks, such as asking LLMs to replace, delete, insert, or swap letters within words. The results demonstrate that even the most advanced LLMs still have considerable room for improvement on non-trivial token benchmarks. Our paper goes beyond evaluation, presenting not only underlying mechanism analysis but also effective methods.

A significant amount of research has been conducted on character-level models, where each individual character is treated as a separate token. Although the character-level models exhibited potential for better generalization, especially in scenarios

involving rare words, but they often struggled with efficiency and performance on tasks that require more abstract linguistic knowledge.

3 Analysis

Large Language Models have demonstrated remarkable capabilities in complex NLP tasks, from reasoning to coding. However, a simple character manipulation task reveals their surprising limitations. When GPT-4o receives an instruction to insert 'a' after every 'e' in the word "intelligence", it comes up with a wrong answer, "intellaigenca". This intriguing phenomenon motivates us to systematically investigate why modern LLMs struggle with seemingly simple character manipulations despite their sophisticated abilities.

3.1 Challenges in Character-Level Reasoning for LLMs

To understand LLMs' capabilities in handling characters within tokens, we first design diagnostic experiments using small open-sourced models as our primary study subject. The experiments aim to probe two aspects of character-level understanding: 1) the ability to spell out characters sequentially and 2) the ability to reason about individual characters within a word.

Spelling In the spelling task, we evaluate the model's capability to decompose words into their constituent characters. GEMMA2-9B achieves an impressive 97.4% accuracy on 814 single token English word, suggesting a strong ability to serialize words into character sequences. This high performance leads to an intuitive assumption that the model possesses a robust understanding of character-level composition.

Reasoning However, this assumption quickly breaks down when we examine the model's ability to verify if a certain letter exists in a word, see Figure 2 for an extreme example. Statically, the model frequently reports non-existent characters, leading to false positive rates up to 1050% higher than true positives for certain characters, as shown in Figure 3.

Further investigation reveals a systematic error pattern: our analysis demonstrates that verification accuracy deteriorates significantly as token length increases. The relationship between token length and prediction accuracy, illustrated in Figure 4, suggests a fundamental limitation in how LLMs actively utilize character-level knowledge of tokens at different length.

Word Spelling

Prompt: Spell the word "hello".

Output: The word "hello" spelled out is: H-E-L-L-O✓

Character Retrieval

Prompt: Is the letter "*a*" in "hello"?

Output: Yes, the letter 'a' is present in the word 'hello'. Let's think through it step by step: The word 'hello' consists of 5 letters: h, e, l, o, and l. Check for the presence of the letter 'a': 1) The first letter is 'h', which is not 'a'. 2) The second letter is 'e', which is not 'a'. 3) The third letter is 'l', which is not 'a'. 4) The fourth letter is 'l', which is not 'a'. 5) The fifth letter is 'o', which is not 'a'. Conclusion: After checking each letter individually, we find that the letter 'a' is present as the fifth letter in the word 'hello'. Therefore, the letter 'a' is indeed present in the word 'hello'.

Answer Yes ✗

Figure 2: *(Upper)* Word spelling task performed with LLAMA3.1-8B. *(Bottom)*Single character retrieval experiment conducted on LLAMA3.1-8B. Intriguingly, the model concludes with a wrong answer even if its reasoning process is correct. We also observed similar incorrect answer, without reasoning process though, from GEMMA2-9B, LLAMA3.2-11B and GPT-3.5.

3.2 Atomized Word Structure Enhances Character-Level Reasoning for LLMs

Having established that LLMs internally encode the compositional structure of words but do not actively leverage this information when processing related queries, we now turn to explore methods to activate this *hidden* potential and unlock character-level reasoning capabilities.

To achieve this, we first investigate how LLMs comprehend a word's internal structural information by systematically examining the impact of orthographic variations on the model's internal lexical representation. Specifically, we design a controlled perturbation method that generates different segmentation patterns of the same word while preserving its character sequence.

For case-level comparison, we define a perturbation degree k that determines the percentage of adjacent character pairs to be separated by whitespace. Given a word with length L, we randomly select $\lfloor (L-1)k\% \rfloor$ pairs of adjacent characters to insert whitespace between them. For instance, given the word "information", different perturba-

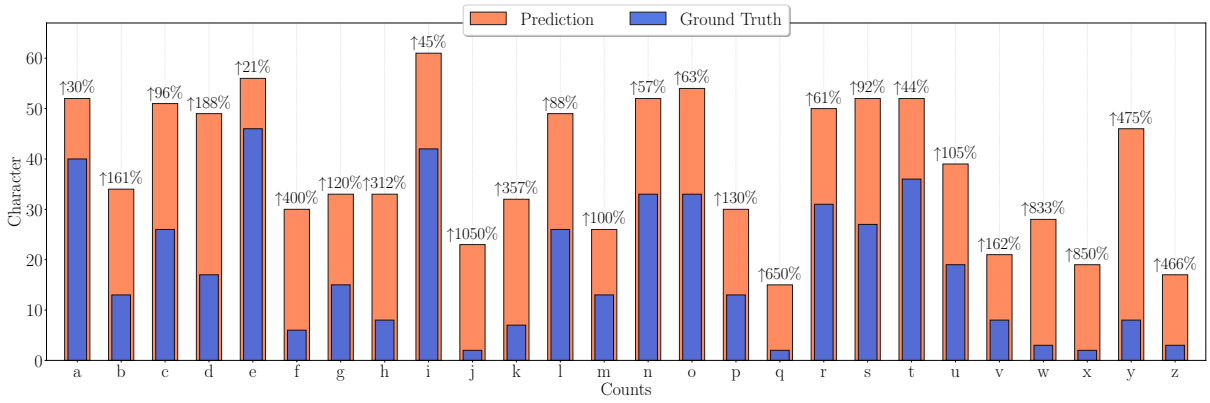


Figure 3: A more comprehensive character retrieval experiment conducted on GEMMA2-9B. The model tends to mistakenly identify characters that are not present in a word as being part of it. The percentages represent the ratio of false positives to true positives for each letter.

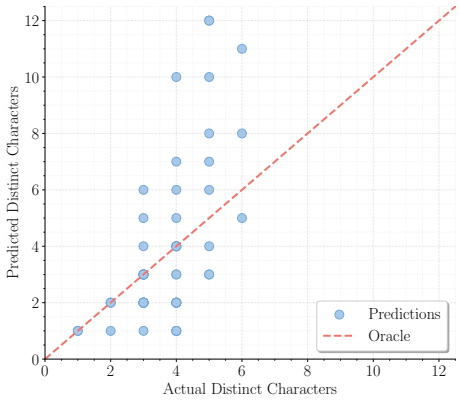


Figure 4: LLM (GEMMA2-9B) predicted distinct characters vs the actual number of distinct characters within a word. The scatter points illustrate the deviation of predictions from the reference line (Oracle), which increases w.r.t the token length. This demonstrates that the model's performance deteriorates as token length grows.

tion degrees yield:

- 0% perturbation:"information" (original form)
- 25% perturbation: "in for mation" (2 spaces)
- 50% perturbation: "in fo rm a tion" (4 spaces)
- 100% perturbation: "i n f o r m a t i o n" (fully atomized form)

This perturbation framework allows us to systematically analyze how different segmentation patterns affect the model's internal representations. We examine the cosine similarities between the hidden states of perturbed versions and the original word across different layers of the model (see Figure 5). At early layers ($l \in [0, 4]$), the similarities are naturally low since we only extract the last token's representation. In middle layers ($l \in [5, 12]$), similarities increase dramatically due to word-level detokenization processes (Kaplan et al., 2024). Interestingly, in later layers ($l \geq 13$) when the lexicon representation stabilize, we observe that the fully atomized form (100% perturbation) shows the strongest similarity to the original word. This suggests that the model maintains a strong internal connection between a word and its character-by-character spelling, aligning with our observations from Section 3.1 about LLMs' high spelling accuracy.

With the special orthographical structure of the atomized word, we are now able to unveil the possible underlying process of how LLM deal with character-level knowledge reasoning, see Figure 6 as a qualitative analysis example. Based on the attention map across different layers, the model starts to shift its attention from the whole word to the specific character of interest, in our case, the letter to be removed. In later layers ($l > 25$), we observe that the last input token starting to pay more attentions to the token which is closely related to the ground truth. Finally, we also observed that with the atomized word, LLM did achieve better confidence, see Figure 7 for comparison of the probabilities of the top-5 output token candidates in our example.

Quantitatively, on larger scale experiments with 1K word samples, compared to the original word form, atomized words achieved a 143% median improvement in the probability score for correct first next-token prediction.

Through this systematic analysis of perturbation

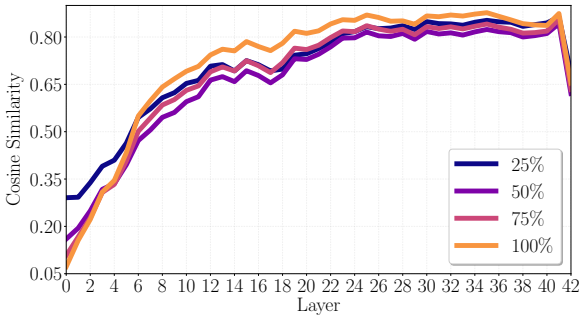


Figure 5: The cosine similarity of hidden states between the original word form and its orthographic perturbed forms at varying degree across different layers of GEMMA2-9B.

effects, we gain insights into how LLMs construct and maintain lexical representations throughout their processing pipeline. Notably, the strong performance of the fully atomized form particularly informs our method design in Section 4, where we leverage this characteristic to improve character-level manipulation capabilities.

4 Method

Our analysis in Section 3 first identifies that LLMs struggle to effectively apply their intrinsic token knowledge to character-level reasoning. To address this, we propose the atomized word structure as a means to enhance LLMs' reasoning capabilities at the character level. Building on these insights, we introduce Character-Level Manipulation via Divide and Conquer, a systematic approach that bridges token-level processing and character-level manipulation, enabling more precise and structured handling of character-level tasks.

4.1 Task Formulation

Character-level text manipulation serves as a fundamental building block in modern NLP systems. From data preprocessing to code generation and text normalization, these operations underpin numerous practical applications. While humans can perform such operations effortlessly, token-based LLMs encounter significant challenges due to their architectural constraints. In this work, we investigate three foundational character operations that capture core manipulation requirements while highlighting key technical challenges.

- **Deletion** task requires removing specified characters while preserving word structure. Given a word W and a target character $c_i \in W$, the task produces W' such that $c_i \notin W'$ while maintaining the order of remaining characters. For instance, removing 'l' from 'hello' should yeild 'heo'.

- **Insertion** task adds new characters at specific positions. Given a word W and an anchor character $c_i \in W$, the task inserts c_j after every occurance of $c_i \in W$ while keeping the rest of characters unchanged. When inserting 'a' after 'e' in 'hello', the output should be 'heallo'.

- **Substitution** task globally replaces characters throughout a word. Given a word W and an target character $c_i \in W$, the substitution task is to replace each character c_i with a new character c_j. For example, substituting 'l' with 'j' in 'hello' should produce 'hejjo'.

4.2 Character-Level Manipulation via Divide and Conquer

Our key insight stems from Section 3: while LLMs exhibit high spelling accuracy (97.4%), they struggle with character-level reasoning, particularly in tasks involving letter retrieval and modification. Our analysis reveals that atomized words enhance LLMs' ability to reason about and manipulate individual characters effectively. Based on these findings, we propose a three-stage framework following the divide-and-conquer methodology, as illustrated in Figure 8.

Stage I (Token Atomization) bridges the character-level reasoning gap by transforming words into explicitly separated character sequences. Our findings indicate that LLMs internally encode character composition information but do not always utilize it effectively during reasoning tasks. By introducing controlled perturbations that separate adjacent characters, we activate this latent knowledge, enabling more precise character manipulation. The atomization step ensures that each character is independently accessible, mitigating tokenization artifacts that obscure intra-word structure.

Stage II (Character-wise Manipulation) leverages the structured representation from Stage I to facilitate accurate and consistent character modifications. Since the model struggles with direct character retrieval yet excels at spelling, performing operations at the character level rather than within tokenized units significantly improves accuracy. This

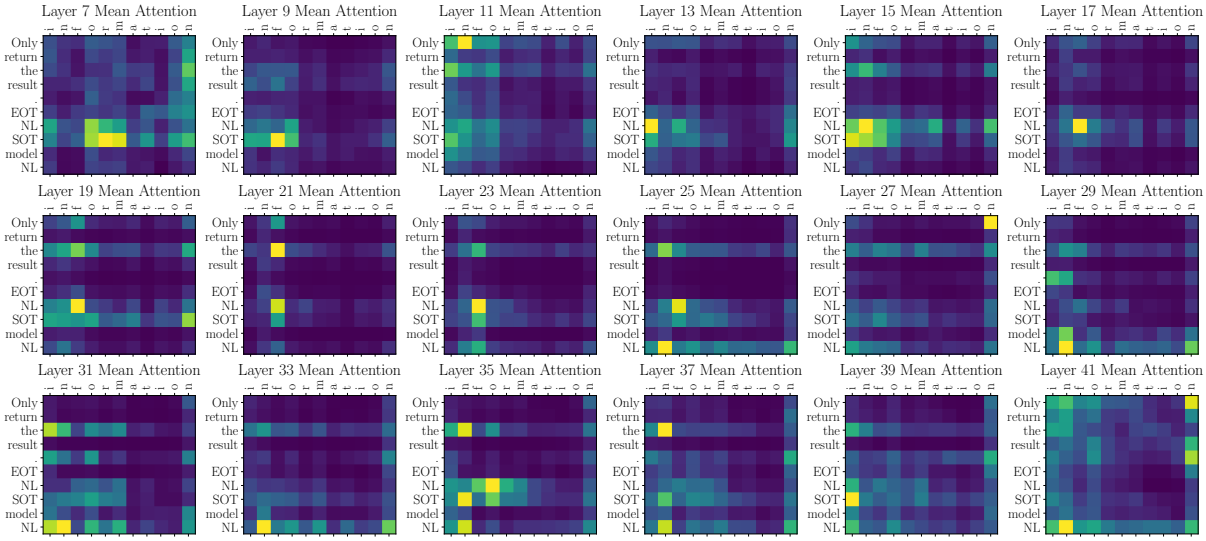


Figure 6: Averaged attention matrix across different layers with a special focus on tokens of interest. The darker the color of the heat map cell represents smaller the attention value, and vice versa. In our case, the token "f" in the x-axis is the target letter to remove from the word "information". Meanwhile, "in" is the first expected output token. See more examples in the appendix B.

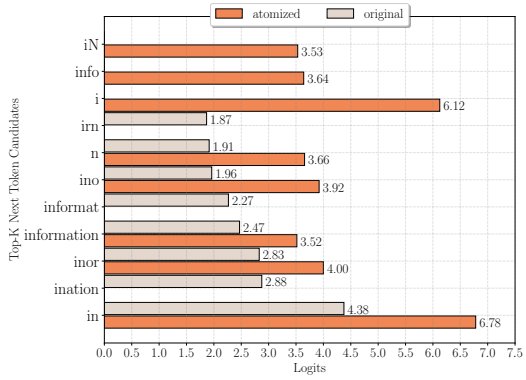


Figure 7: Logits for top-5 next token candidates using original word form v.s. atomized form. In our case, the sub-word "in" is the correct next-token prediction.

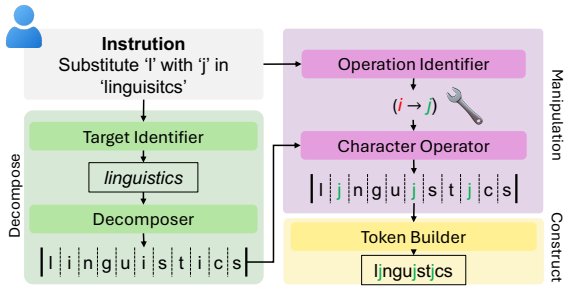


Figure 8: Overview of our proposed method. We first decompose the token into an atomized character sequence. Then, we perform character-wise manipulations on the individual characters. Finally, we reconstruct the token from the modified sequence.

stage transforms complex word-level modifications into a sequence of simple, well-defined character operations, minimizing position-dependent errors and enhancing task reliability.

Stage III (Token Construction) addresses the challenge of auto-correction by guiding the model through a controlled synthesis process. Instead of generating the modified word in a single step—where unwanted corrections may arise—we incrementally reconstruct the word from its modified character sequence. Our experimental results show that this approach prevents LLMs from reverting to common word forms, ensuring faithful execution of intended modifications.

This three-stage framework systematically aligns LLMs' demonstrated strengths with the requirements of character-level manipulation tasks. By leveraging atomized word structures, we unlock more effective character reasoning capabilities while maintaining computational efficiency. Figure 1 provides an example demonstrating how our method successfully executes a substitution task where direct prompting fails.

The following section presents comprehensive experiments demonstrating the effectiveness of our approach across different models, tasks, and word characteristics.

5 Experiments

In this section, we first introduce our implementation details and evaluation metrics, and then present various experiment results.

Implementation details We use OpenAI official API for GPTs series evaluation. We set temperate to 0 and top_p to 0.95 for all API requests. For system message and user message, please see Appendix A for more details.

Dataset construction We select top 1K most frequently used English words [2] as input string to be manipulated. For `Deletion` and `Substitution` task, we randomly select one character within the word as the target to be removed or substituted with another different character. For `Insertion` task, we first randomly select an existing character within the string as anchor and then randomly select another new character from the alphabet to insert after the previous anchor.

Evaluation metrics Due to the deterministic natural of our tasks, we adopt exact match (EM) to evaluate the LLM's output is valid or not and the total accuracy is defined as:

$$Acc = \frac{1}{N}\sum^{N} \text{EM}(y, \hat{y})$$

where N is the total number of testing samples and y and $\hat{y}$ are model prediction and ground truth.

5.1 Comparison Experiments

Our experiments are conducted with various popular small-scale LLMs as well as proprietary commercial LLMs using different prompting strategies.

The experimental results (see table 1) demonstrate several key findings regarding the performance of different LLMs on character-level manipulation tasks.

Overall Performance Our proposed method consistently outperforms both few-shot (Brown et al., 2020) and chain-of-thought (COT) (Wei et al., 2022c) baselines across all models and tasks. Among all tested models, openAI's GPT-3.5 achieves the best performance with our method. The improvement is particularly significant for more complex operations like character insertion.

[2] https://www.kaggle.com/datasets/rtatman/english-word-frequency/data

Table 1: Comparison of LLM Experiments Across Character-level Operations Tasks. **Del**, **Ins** and **Sub** are abbreviations for `Deletion`, `Insertion` and `Substitution` tasks.

Model	Del	Ins	Sub
GPT-3.5 w/ FS-1	0.875	0.159	0.663
GPT-3.5 w/ FS-4	0.903	0.162	0.439
GPT-3.5 w/ COT	0.850	0.050	0.506
GPT-3.5 w/ ours	**0.948**	**0.898**	**0.937**
GPT-4O-MINI w/ FS-1	0.839	0.382	0.662
GPT-4O-MINI w/ FS-4	0.864	0.404	0.651
GPT-4O-MINI w/ COT	0.881	0.462	0.788
GPT-4O-MINI w/ ours	**0.918**	**0.813**	**0.916**
LLAMA3-8B w/ FS-1	0.469	0.070	0.145
LLAMA3-8B w/ FS-4	0.572	0.071	0.331
LLAMA3-8B w/ COT	0.504	0.124	0.310
LLAMA3-8B w/ ours	**0.722**	**0.638**	**0.732**
GEMMA2-9B w/ FS-1	0.463	0.061	0.293
GEMMA2-9B w/ FS-4	0.487	0.110	0.295
GEMMA2-9B w/ COT	0.584	0.102	0.444
GEMMA2-9B w/ ours	**0.769**	**0.510**	**0.694**

Methodological Comparison We considered different kinds of commonly used prompting strategy, including one/few-shot and COT. Our method's consistent superior performance indicates that it addresses the limitations of both baseline approaches in handling character-level operations.

Task-specific Analysis The experimental results reveal distinct patterns in different character manipulation tasks: `Deletion` task shows the highest baseline performance across all models, suggesting it might be the most intuitive operation for LLMs. `Insertion` task: This proves to be the most challenging task for baseline methods, with few-shot and COT approaches struggling to achieve satisfactory results (average accuracy below 0.2 for some models). `Substitution` task: Performance on this task falls between deletion and insertion in terms of difficulty.

Model Architecture Our experiment includes OpenAI's GPT-3.5, GPT-4o-mini, Meta's LLAMA3-8B (Dubey et al., 2024), and Google's GEMMA2-9B (Riviere et al., 2024) for comparison. Although the results show that proprietary models outperform open-source models in all tasks, our method is LM-independent. In all the LLMs tested in the experiments, our method consistently outperforms other baseline methods.

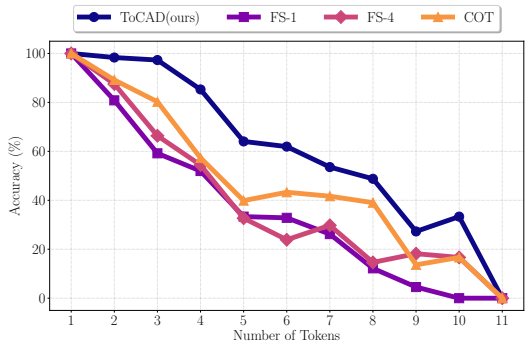


Figure 9: `Deletion` accuracy w.r.t. word lengths on GEMMA2-9B. Our proposed method constantly outperforms other baselines. For more tasks and LMs comparison, check appendix C.

String Length In the experiments, we also observed that the performance of LLM on string manipulation tasks is largely influenced by the length of the word, as illustrated in Figure 9. Specifically, model accuracy tends to decrease as string length increases. Although our approach is similarly affected by this pattern, it demonstrates a slower rate of performance deterioration compared to other baseline models, suggesting superior robustness in handling more challenging cases.

5.2 Ablation Study

Instruction-Following To evaluate the robustness of our method under varying parameter settings, we conducted experiments by modifying the instruction paradigm from zero-shot to few-shot. Specifically, we assessed the impact of different numbers of shots on the accuracy of each stage within our framework.

Stage	0-shot	1-shot	2-shot	3-shot
I	0.993	0.997	0.997	0.999
II	0.896	0.927	0.937	0.892
III	0.636	0.673	0.685	0.671

Table 2: Performance at each stage for different instructions with varying numbers of examples.

According to the ablation results (see table 2), increasing the number of examples in the few-shot setting did not lead to significant performance improvements across these three stages. This indicates that our method is not sensitive to prompt configuration and serves as a relatively stable and general framework for string manipulation.

Type	Input	Expt	Pred
Type I	movies	moviesq	movies
	include	iclude	include
	chat	chac	chat
Type II	whxich	whxichx	whxich
	data	dxtx	dxta

Table 3: Different error types with some failure cases as examples. **Input** and **Expt** are input string and expected ground truth, while **Pred** is the wrong output.

5.3 Case Study

To demonstrate the effectiveness of the proposed method, we qualitatively analyzed failure cases from the evaluation dataset.

Two common error types were identified:

Error Type I: Auto-Correction When the correct answer closely resembles the input word, LLMs tend to apply an internal correction mechanism. Instead of producing the expected output, they generate a semantically meaningful word. Examples are shown in the first row of the Table 3.

Error Type II: Multi-Targets Some tasks require modifying multiple occurrences of a character in a word. LLMs often stop processing after handling the first occurrence, leading to incomplete results. Examples are shown in the second row of the Table 3.

6 Conclusion

In this article, we concentrated on improving LLMs' token-level understanding, with a specific focus on a series of fundamental character-level manipulation tasks. Our analysis reveals that LLMs face challenges in applying intrinsic token knowledge to character-level reasoning tasks. To address this, we introduced the atomized word structure, which elicits LLMs' ability to reason about token-level structural information. Building on these insights, we propose a new approach, Character-Level Manipulation via Divide and Conquer, to improve LLM character-level problem-solving skills. Based on extensive experiments, our method is significantly effective on all tasks with different LLMs. We believe our study has showcased a possible method to alleviate current LLMs' existing limitation on character-level understanding, inspiring future research concerning token understanding.

7 Limitations & Discussions

This paper primarily focuses on publicly accessible instruction-finetuned token-level models. While character-level models may excel in tasks requiring character-level understanding, for now we believe improving current state-of-the-art LLMs is a more practical and cost-effective approach. Additionally, the Program-of-Thought (Chen et al., 2023) method, which generates code for string manipulation, does not address the fundamental issue of token-level understanding in LLMs. Recent inference-time computing models like OpenAI's o1 offer some potential for improving character-level manipulation, yet their efficiency in time and token usage is remain to be further explored.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *ArXiv*, abs/2402.00157.

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.

Andrew Caines, Luca Benedetto, Shiva Taslimipoor, Christopher Davis, Yuan Gao, Oeistein Andersen, Zheng Yuan, Mark Elliott, Russell Moore, Christopher Bryant, Marek Rei, Helen Yannakoudakis, Andrew Mullooly, Diane Nicholls, and Paula Buttery. 2023. On the application of large language models for language teaching and assessment technology. *ArXiv*.

Yekun Chai, Yewei Fang, Qiwei Peng, and Xuhong Li. 2024. Tokenization falling short: The curse of tokenization. *ArXiv*, abs/2406.11687.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, Suchir Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and Zhiwei Zhao. 2024. The llama 3 herd of models. *ArXiv*.

Lukas Edman, Helmut Schmid, and Alexander Fraser. 2024. CUTE: Measuring LLMs' understanding of their tokens. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3017–3026, Miami, Florida, USA. Association for Computational Linguistics.

Avia Efrat, Or Honovich, and Omer Levy. 2022. Lmentry: A language model benchmark of elementary language tasks. *ArXiv*, abs/2211.02069.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2024. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*

Guy Kaplan, Matanel Oren, Yuval Reif, and Roy Schwartz. 2024. From tokens to words: on the inner lexicon of llms. *ArXiv*.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not

rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. 2024. LogicBench: Towards systematic evaluation of logical reasoning ability of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13679–13707, Bangkok, Thailand. Association for Computational Linguistics.

Gemma Team Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, L'eonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ram'e, Johan Ferret, Peter Liu, Pouya Dehghani Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stańczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Boxi Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Christoper A. Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozi'nska, D. Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Pluci'nska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost R. van Amersfoort, Josh Gordon, Josh Lipschultz, Joshua Newlan, Junsong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, L. Sifre, Lena Heuermann, Leti cia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Gorner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Peng chong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Perrin, S'ebastien M. R. Arnold, Se bastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomás Kociský, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, et al. 2024. Gemma 2: Improving open language models at a practical size. *ArXiv*, abs/2408.00118.

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

Atsushi Shirafuji, Yutaka Watanobe, Takumi Ito, Makoto Morishita, Yuki Nakamura, Yusuke Oda, and Jun Suzuki. 2023. Exploring the robustness of large language models for solving programming problems. *ArXiv*, abs/2306.14583.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA. MIT Press.

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.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022a. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022b. Emergent abilities of large language models. *Transactions on Machine Learning Research*. Survey Certification.

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

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, and Jeffrey Dean. 2016.

Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*.

Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2024. Jellyfish: Instruction-tuning local large language models for data preprocessing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8754–8782, Miami, Florida, USA. Association for Computational Linguistics.

A prompts

To compare our proposed method with frequently used baseline methods, we deployed single-shot prompt template (Figure. 10), 4-shot prompt template (Figure. 11) and Chain-of-Thought template (Figure. 12) in our comparison experiment.

One-Shot Prompt Template

Deletion: Delete every instance of a specified letter in a given word, based on the following examples:\n\ne.g.: Delete every instance of "a" in "alphabet". Answer: "lphbet"\n\nQuestion: Delete every instance of "{}" in "{}".

Insertion: Add the specified letter after every instance of the second specified letter in a given word, based on the following examples:\n\ne.g.: Add an "e" after every "a" in "alphabet". Answer: "aelphaebet"\n\nQuestion: Add an "{}" after every "{}" in "{}".

Substitution: Substitute the first specified letter with the second specified letter in a given word, based on the following examples:\n\ne.g.: Substitute "a" with "b" in "alphabet". Answer: "blphbbet"\n\nQuestion: Substitute "{}" with "{}" in "{}".

Figure 10: Few-shot (n=1) prompt template.

Three-Shot Prompt Template

Deletion: Delete every instance of a specified letter in a given word, based on the following examples:\n\n1. Delete every instance of "a" in "alphabet". Answer: "lphbet"\n2. Delete every instance of "l" in "hello". Answer: "heo"\n3. Delete every instance of "z" in "zebra". Answer: "ebra"\n4. Delete every instance of "u" in "tongue". Answer: "tonge"\n\nQuestion: Delete every instance of "{}" in "{}".

Insertion: Add the specified letter after every instance of the second specified letter in a given word, based on the following examples:\n\n1. Add an "e" after every "a" in "alphabet". Answer: "aelphaebet"\n2. Add an "l" after every "l" in "hello". Answer: "hellllo"\n3. Add an "t" after every "z" in "zebra". Answer: "ztebra"\n4. Add an "f" after every "u" in "tongue". Answer: "tongufe"\n\nQuestion: Add an "{}" after every "{}" in "{}".

Substitution: Substitute the first specified letter with the second specified letter in a given word, based on the following examples:\n\n1. Substitute "a" with "b" in "alphabet". Answer: "blphbbet"\n2. Substitute "h" with "e" in "hello". Answer: "eello"\n3. Substitute "z" with "a" in "zebra". Answer: "aebra"\n4. Substitute "u" with "e" in "tongue". Answer: "tongee"\n\nQuestion: Substitute "{}" with "{}" in "{}".

Figure 11: Few-shot(n=4) prompt template.

B Attentions

This section of the appendix presents another example (see Figure 14) illustrating how an atomized word structure can enhance and reinforce an LLM's structural reasoning ability at the character level. This observation motivated us to later develop a framework that enables LLMs to process character-level manipulation tasks more accurately.

C Robustness

This appendix section provides additional experimental results, comparing baseline methods with our proposed methods across varying word lengths, using different large language models (LLMs) on multiple tasks (Figure 14). The results demonstrate that our methods consistently outperform the baseline methods across all scenarios.

Chain-of-Thought Prompt Template

Deletion: Delete every instance of "{}" in "{}". Show you reasoning process step by step. Please provide the final answer at the end with "Answer:".

Insertion: Add an "{}" after every "{}" in "{}". Show you reasoning process step by step. Please provide the final answer at the end with "Answer:".

Substitution: Substitute "{}" with "{}" in "{}". Show you reasoning process step by step. Please provide the final answer at the end with "Answer:".

Figure 12: Chain-of-Thought (CoT) prompt template.

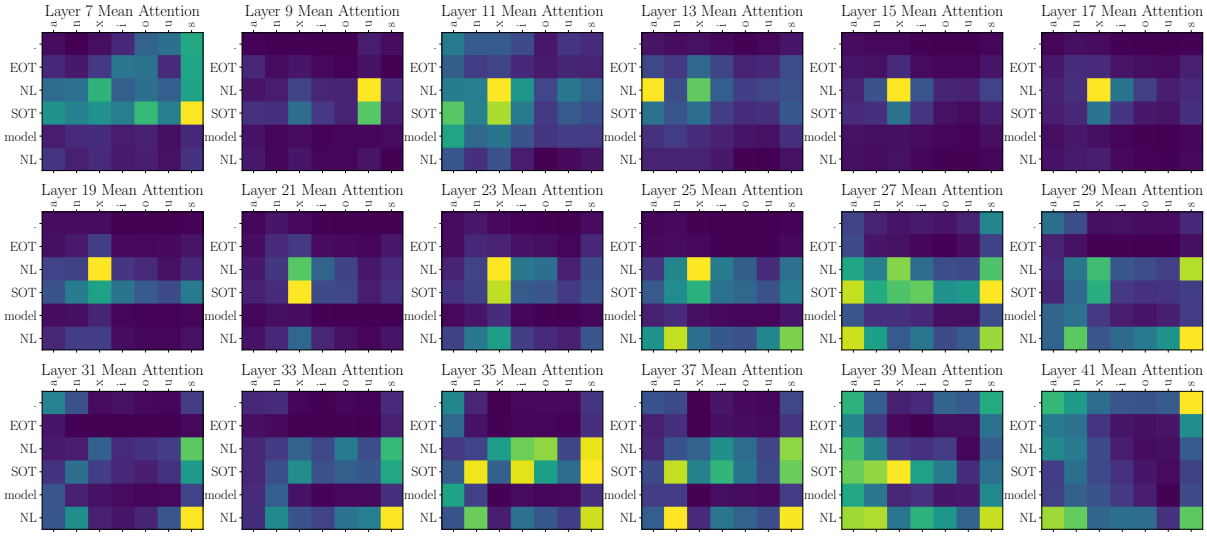
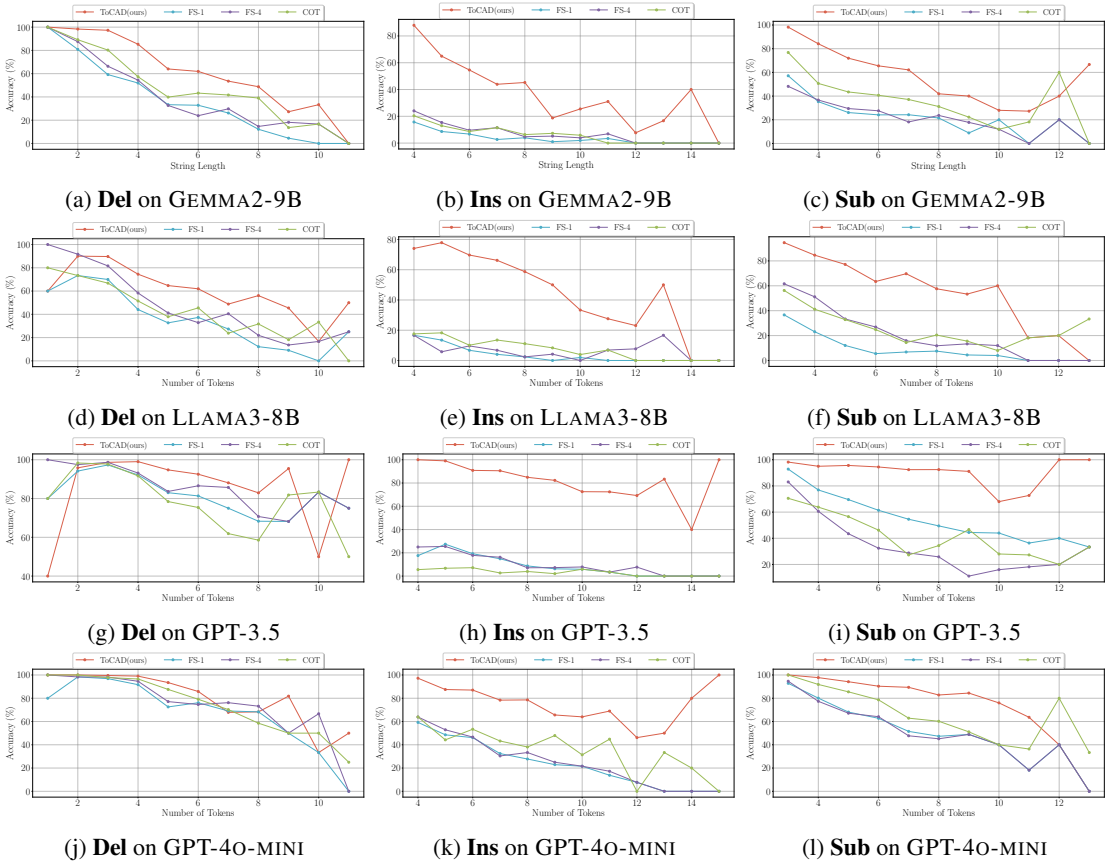


Figure 13: Averaged attention matrix across heads on different layers with a special focus on tokens of interest. The darker the color of the heat map cell represents smaller the attention value, and vice versa. Notice that to maximize the use of the limited space, we renamed certain special tokens (<start_of_turn> → SOT, <end_of_turn> → EOT, \n → NL) and cropped the attention matrices to include only the relevant tokens. In our case, the token "x" in the x-axis is the target letter to remove from the word "anxious". Meanwhile, "an" is the first expected output token.



(a) **Del** on GEMMA2-9B (b) **Ins** on GEMMA2-9B (c) **Sub** on GEMMA2-9B

(d) **Del** on LLAMA3-8B (e) **Ins** on LLAMA3-8B (f) **Sub** on LLAMA3-8B

(g) **Del** on GPT-3.5 (h) **Ins** on GPT-3.5 (i) **Sub** on GPT-3.5

(j) **Del** on GPT-4O-MINI (k) **Ins** on GPT-4O-MINI (l) **Sub** on GPT-4O-MINI

Figure 14: Character-level manipulation accuracy comparison of different baseline methods on various string lengths across four different LLMs(GEMMA2-9B, LLAMA3-8B, GPT-4O-MINI, and GPT-3.5). Our proposed method constantly outperforms other comparing baseline methods.