Enhancing LLM Character-Level Manipulation via Divide and Conquer

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have shown impressive generalization across diverse natural language processing tasks. However, they consistently struggle with character-level string manipulation such as deletion, insertion, and substitution, despite the foundational role of these operations in data preprocessing and code generation. This gap raises a critical question: *Why do LLMs, despite their strong token-level capabilities, fail at basic character-level manipulations?* To address this question, we conduct a systematic analysis and uncover two key findings: (1) LLMs have limited ability to leverage intrinsic token knowledge for fine-grained character reasoning, and (2) decomposing words into atomized structures can significantly enhance their sensitivity to token-level structure. Building on these insights, we propose Character-Level Manipulation via Divide and Conquer, a novel framework that bridges the gap between token-level processing and character-level manipulation. Our approach decomposes complex tasks into explicit character-level subtasks followed by controlled token reconstruction phases. This method leads to significant accuracy improvements without requiring additional model training. Empirical results show that Character-Level Manipulation via Divide and Conquer achieves notable gains on the `Deletion`, `Insertion`, and `Substitution` benchmarks. We release our implementation and evaluation suite to support future research in character-aware language modeling.

1 Introduction

Large Language Models (LLMs) have recently demonstrated remarkable success across a wide range of NLP tasks (Brown et al., 2020; Wei et al., 2022b,a). Yet, beneath this progress lies a surprisingly persistent blind spot: even state-of-the-art LLMs routinely fail at seemingly simple character-level string manipulations. For example, when prompted to insert an ‘a’ after every ‘e’ in "intelligence", ChatGPT-4o produces "intellaigenca"—a non-trivial error for such a basic operation. These failures are not anecdotal, but symptomatic of a broader, systematic limitation.

This limitation stems from how LLMs process text. Modern models rely on tokenization algorithms, such as BPE (Sennrich et al., 2016) and SentencePiece (Kudo and Richardson, 2018), which convert words like "linguistics" into token sequences (e.g., [3321, 84, 7592][1]). While efficient for most NLP tasks, such tokenization disrupts access to individual characters, hindering the model’s ability to reason and act at the character level—even when some character-level knowledge is passively learned during training. Our experiments confirm that, although LLMs excel at spelling, this skill rarely translates to reliable character manipulation.

Character-level operations are foundational in many real-world workflows where the *task* cannot simply be offloaded to an external script. For example, in software engineering, code generation assistants must adaptively modify variable names or syntax on-the-fly, based on ambiguous or context-dependent user instructions (Chen et al., 2021). Data preprocessing and text normalization pipelines often require flexible, language-aware corrections (Zhang et al., 2024). Educational and assistive technologies increasingly depend on nuanced, context-sensitive spelling and grammar editing (Omelianchuk et al., 2020; Caines et al., 2023). In all these scenarios, it is impractical or impossible to enumerate all possible manipulations ahead of time in a static codebase—flexibility and interpretability demand that LLMs handle such tasks within the flow of language.

Despite the centrality of these applications, prior research has focused mainly on benchmarking LLMs’ character-level capabilities (e.g., CUTE (Edman et al., 2024)), rather than interrogating the *mechanisms* or designing methods that

[1] o200k_base tokenizer

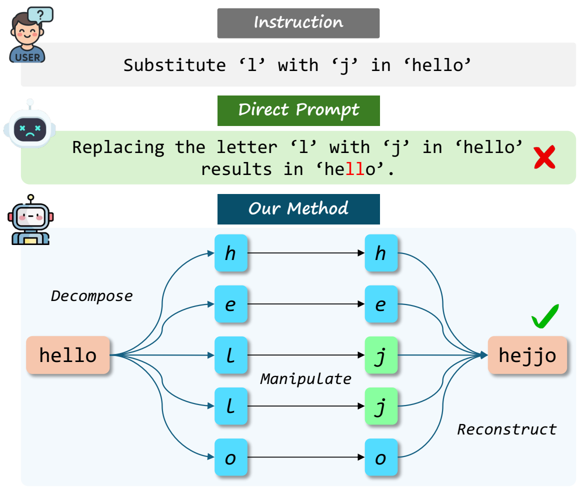

Instruction
Substitute 'l' with 'j' in 'hello'
Direct Prompt
Replacing the letter 'l' with 'j' in 'hello' results in 'hello'.
Our Method
Decompose
hello
h
e
l
l
o
Manipulate
h
e
j
j
o
Reconstruct
hejjo

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.

make LLMs robust to such operations.

Our systematic analysis reveals two key insights: (I) LLMs consistently perform well in spelling tasks, and (II) presenting words in fully atomized (character-separated) forms can activate latent character-level reasoning. Motivated by these findings, we propose Character-Level Manipulation via Divide and Conquer, a novel divide-and-conquer framework that bridges token-level and character-level processing. Specifically, our approach decomposes any manipulation into three explicit stages: token decomposition, character-level manipulation, and token reconstruction—all accomplished within the LLM, without extra finetuning or external code. See Figure 1 for an overview.

We validate our approach through comprehensive experiments: on `Deletion`, `Insertion`, and `Substitution` tasks, our method consistently outperforms standard prompting and few-shot baselines across multiple LLMs. In addition, our error analysis uncovers persistent bottlenecks in LLMs' internal handling of character-level structure, providing actionable insights for both practitioners and model designers.

- We present the first systematic analysis of LLM character-manipulation challenges and show how structural input variation can unlock reasoning capabilities.
- We introduce a zero-shot, model-agnostic method that substantially improves character-level operation accuracy, requiring no additional training.
- Our extensive experimental analysis not only validates the effectiveness of our approach, but also lays the groundwork for future research in making LLMs more robust and interpretable at the character level.

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.

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.

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

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 pertur-

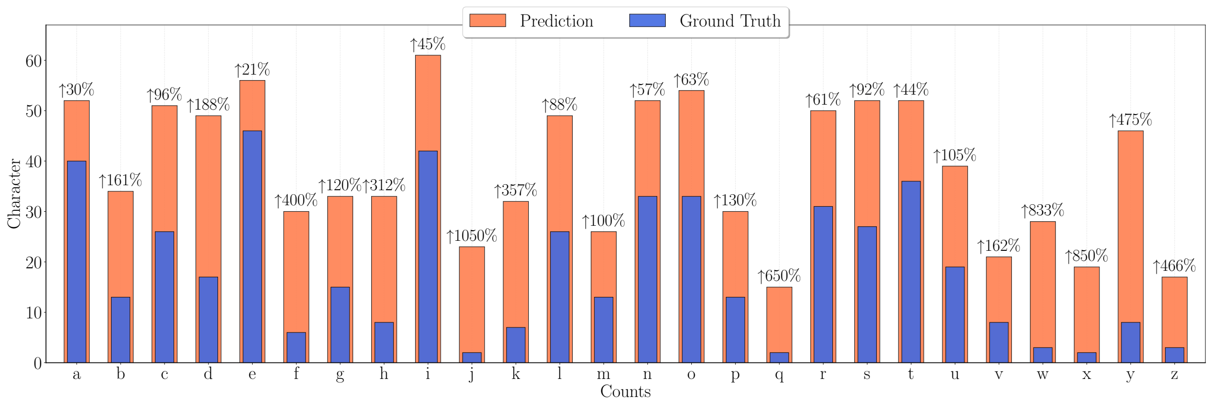


Figure 3: A more comprehensive character retrieval experiment conducted on GEMMA2-9B. The model tends to mistakenly identify characters that are not present in a word as being part of it. The percentages represent the ratio of false positives to true positives for each letter.

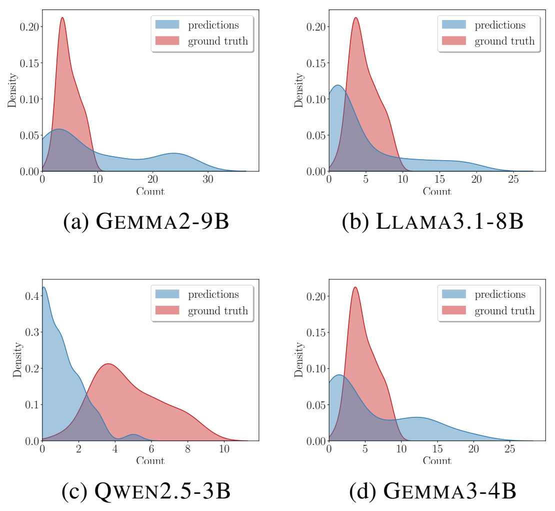


Figure 4: Distribution of character counts per token predicted by various LLMs in different scales compared to ground truth. The divergence between the distributions suggests a systematic bias or calibration shift in the model's character-level predictions.

bation 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 perturbation 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

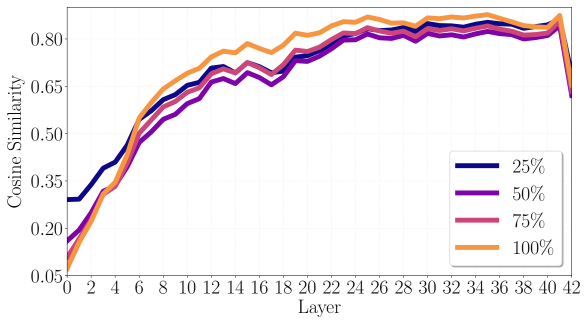


Figure 5: The cosine similarity of hidden states between the original word form and its orthographic perturbed forms at varying degree across different layers of GEMMA2-9B.

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 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, as established in Section 3, is that while LLMs demonstrate strong proficiency in spelling (97.4% accuracy), they frequently fail at active character-level reasoning—particularly when required to retrieve, insert, or substitute letters within words. Notably, our analysis shows that presenting words in fully atomized (character-separated) forms can activate latent character-level knowledge within the model, leading to more reliable manipulation. Building on this, we propose a principled three-stage **divide-and-conquer** framework that explicitly guides LLMs through each required sub-operation (see Figure 1).

Stage I: Token Atomization. The first stage explicitly decomposes the input word into a sequence of isolated characters, separated by spaces (e.g., "hello" → "h e l l o"). This atomization step is critical: it sidesteps tokenization artifacts, making each character independently accessible to the LLM and preventing the model from relying solely on its token-level priors. By providing controlled, fine-grained segmentation, we prompt the LLM to operate on characters rather than opaque subword

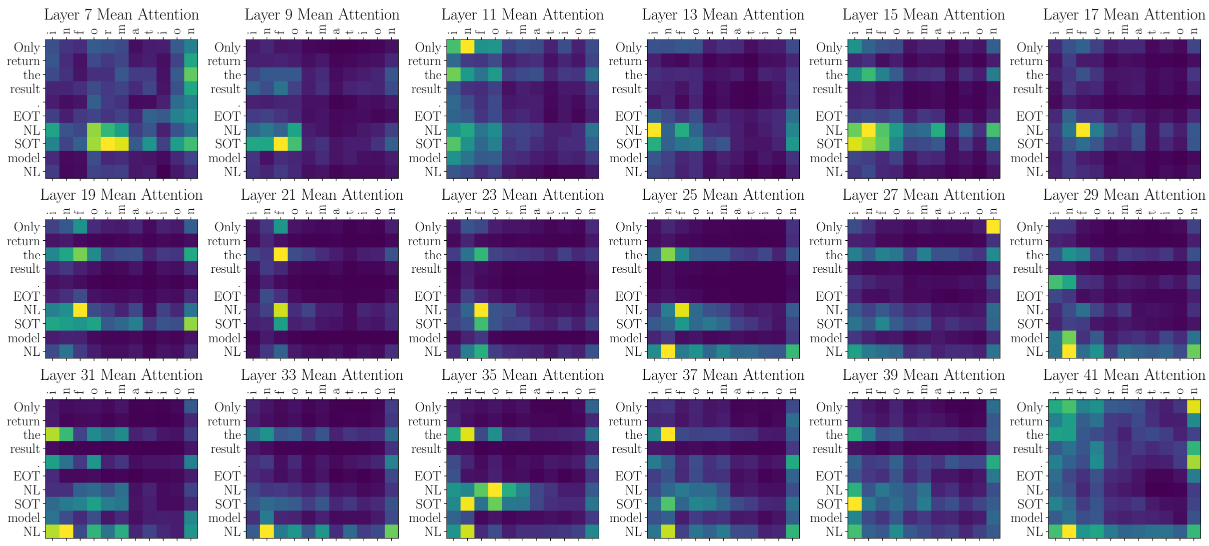


Figure 6: Averaged attention matrix across different layers with a special focus on tokens of interest. The darker the color of the heat map cell represents smaller the attention value, and vice versa. In our case, the token "f" in the x-axis is the target letter to remove from the word "information".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

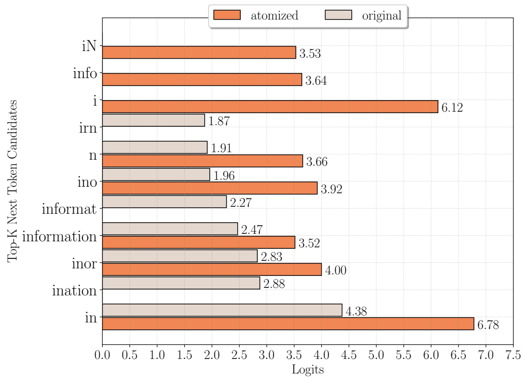


Figure 7: Logits for top-5 next token candidates using original word form v.s. atomized form. In our case, the sub-word "in" is the correct next-token prediction.

tokens, thus "unlocking" its implicit understanding of word structure. We note that this atomization is performed directly within the prompt and requires no additional model training.

Stage II: Character-wise Manipulation. In this stage, the LLM is instructed to carry out the desired manipulation—deletion, insertion, or substitution—at the character level on the atomized sequence. For example, to substitute 'l' with 'j' in "h e l l o", the model operates stepwise, applying the edit to each character position as appropriate. This explicit character-wise framing mitigates common failure modes of token-based manipulation, such as incomplete edits or positional drift. By reducing the manipulation to a sequence of simple, local operations, this stage enhances both accuracy and interpretability. We emphasize that each manipulation is guided by a dedicated prompt template, which can be flexibly adapted for different types of string operations.

Stage III: Token Reconstruction. The final stage incrementally reconstructs the manipulated word from the edited character sequence, guiding the LLM to output the result as a contiguous string (e.g., "h e j j o" $\rightarrow$ "hejjo"). Crucially, we instruct the model to avoid additional corrections or auto-normalization, which can otherwise cause the output to revert to more common word forms—a prevalent failure in direct prompting. This staged reconstruction preserves the intended structure of the manipulated word and ensures faithful execution of the requested operation. As with earlier steps, this process is implemented at the prompt level without altering the model's weights.

Operationalization: Practically, our framework chains these three stages together using a sequential LLM call. Example prompts for each stage are provided in Appendix A. This architecture is lightweight, requires no extra training or external code, and is compatible with most instruction-tuned LLMs.

Figure 1 demonstrates how our divide-and-conquer approach successfully solves a substitution task that direct prompting fails to handle. The next section presents a comprehensive empirical evaluation across diverse models, tasks, and linguistic settings.

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:

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

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

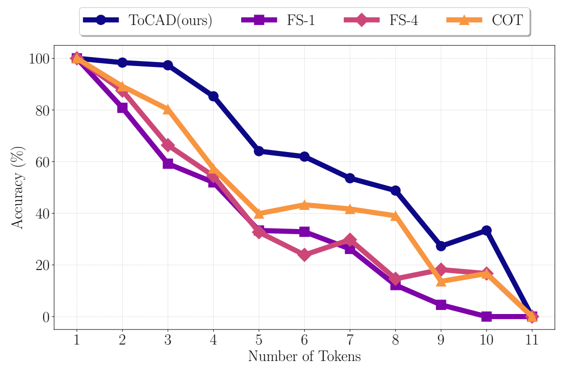


Figure 8: `Deletion` accuracy w.r.t. word lengths on GEMMA2-9B. Our proposed method constantly outperforms other baselines. For more tasks and LMs comparison, check appendix C.

outperforms other baseline methods.

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 8. 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	Output
Auto-Correct	movies	moviesq	movies
	include	iclude	include
	chat	chac	chat
Multi-Targets	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 **Output** 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

This work reveals and addresses a fundamental gap in the character-level manipulation abilities of large language models (LLMs). We show that despite strong generalization, LLMs often fail at explicit string operations due to tokenization bottlenecks. Our proposed divide-and-conquer framework leverages atomized word structures and staged manipulation to unlock latent character-level reasoning, achieving consistent improvements across deletion, insertion, and substitution tasks. Beyond the empirical gains, our analysis sheds light on the structural underpinnings of LLMs' internal representations, offering practical tools for model analysis and actionable insights for advancing token-level interpretability. We hope these findings will encourage further research bridging subword tokenization and fine-grained linguistic control in next-generation LLMs.

7 Limitations & Discussions

Our study is limited to widely used, instruction-finetuned token-based LLMs. While dedicated character-level models may offer stronger performance for specific tasks, our focus is on practical, broadly compatible enhancements for existing models. We also note that many real-world character manipulation problems could be solved with explicit programming; the value of our approach lies in its ability to integrate natural language-driven editing within end-to-end LLM workflows. Additionally, while our method reduces tokenization-induced errors, it introduces some necessary computational overhead due to multi-step prompting.

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.

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, 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 Experimental Prompts

To compare our proposed method with frequently used baseline methods, we deployed single-shot prompt template (Figure. 9), 4-shot prompt template (Figure. 10) and Chain-of-Thought template (Figure. 11) 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 9: 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 10: Few-shot(n=4) prompt template.

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 11: Chain-of-Thought (CoT) prompt template.

Structured Prompt Templates for Stage I Pipeline

Goal: Decompose a word into a list of its characters.
Prompt Template:

Given the word `"{WORD}"`, return a Python list of its characters.

Example: Input: "linguistics" Output: `['l', 'i', 'n', 'g', 'u', 'i', 's', 't', 'i', 'c', 's']`

Input: `"{WORD}"` Output:

Alternatively, as JSON:

Given the word `"{WORD}"`, return its characters in a JSON array.

Example: Input: "hello" Output: `["h", "e", "l", "l", "o"]`

Figure 12: Complete structured prompt template for stage I pipeline

Structured Prompt Templates for Stage III Pipeline

Goal: Concatenate the character list into a string and output as a structured object.
Prompt Template:

Given the Python list `{MODIFIED_LIST}`, concatenate the characters to form a single string, and return the result as a JSON object with the field "result".

Example: Input: `['h', 'e', 'j', 'j', 'o']` Output: `{"result": "hejjo"}`

Input: `{MODIFIED_LIST}` Output:

Figure 13: Complete structured prompt template for stage III pipeline

B Attentions

This section of the appendix presents another example (see Figure 16) illustrating how an atomized

Structured Prompt Templates for Stage II Pipeline

Goal: Manipulate the character list according to the specified operation and output the result in a structured format.
Prompt Templates:
Deletion:

> Given the Python list {CHAR_LIST}, delete every occurrence of the character "{TARGET}". Return the resulting list.
>
> Example: Input: ['h', 'e', 'l', 'l', 'o'], Target: "l" Output: ['h', 'e', 'o']
>
> Input: {CHAR_LIST}, Target: "{TARGET}" Output:

Insertion:

> Given the Python list {CHAR_LIST}, insert the character "{INSERT}" after every occurrence of "{ANCHOR}". Return the resulting list.
>
> Example: Input: ['h', 'e', 'l', 'l', 'o'], Insert: "a" after "l" Output: ['h', 'e', 'l', 'a', 'l', 'a', 'o']
>
> Input: {CHAR_LIST}, Insert: "{INSERT}" after "{ANCHOR}" Output:

Substitution:

> Given the Python list {CHAR_LIST}, replace every occurrence of "{FROM}" with "{TO}". Return the resulting list.
>
> Example: Input: ['h', 'e', 'l', 'l', 'o'], Substitute: "l" with "j" Output: ['h', 'e', 'j', 'j', 'o']
>
> Input: {CHAR_LIST}, Substitute: "{FROM}" with "{TO}" Output:

Alternatively, for JSON syntax, simply replace brackets with square brackets and use double quotes:
Example: Input: ["h", "e", "l", "l", "o"], Substitute: "l" with "j" Output: ["h", "e", "j", "j", "o"]

Figure 14: Complete structured prompt template for stage II Pipeline

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 16). The results demonstrate that our methods consistently outperform the baseline methods across all scenarios.

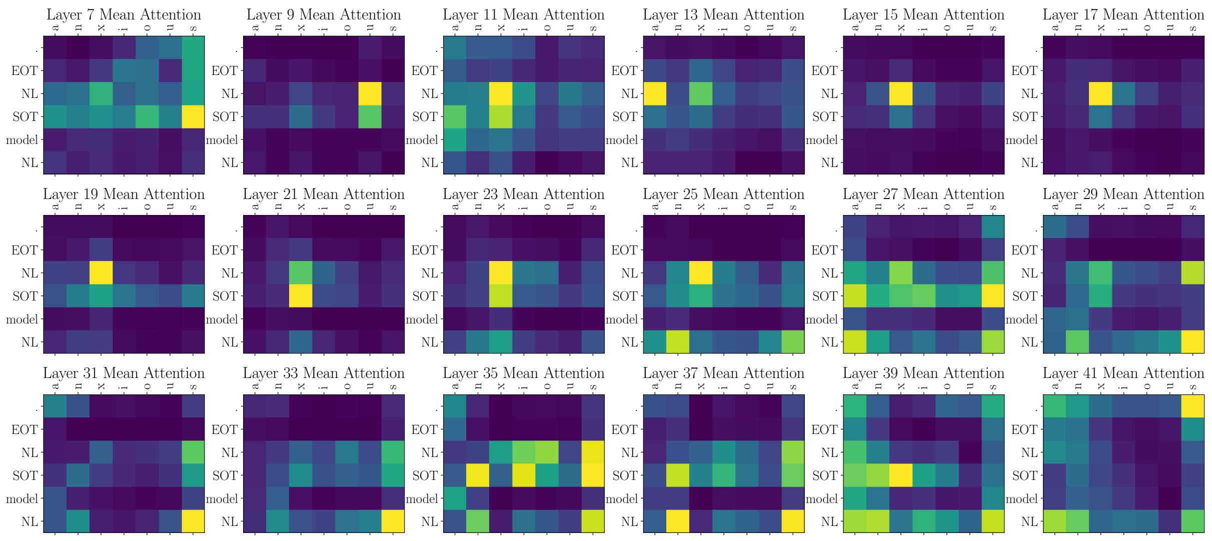
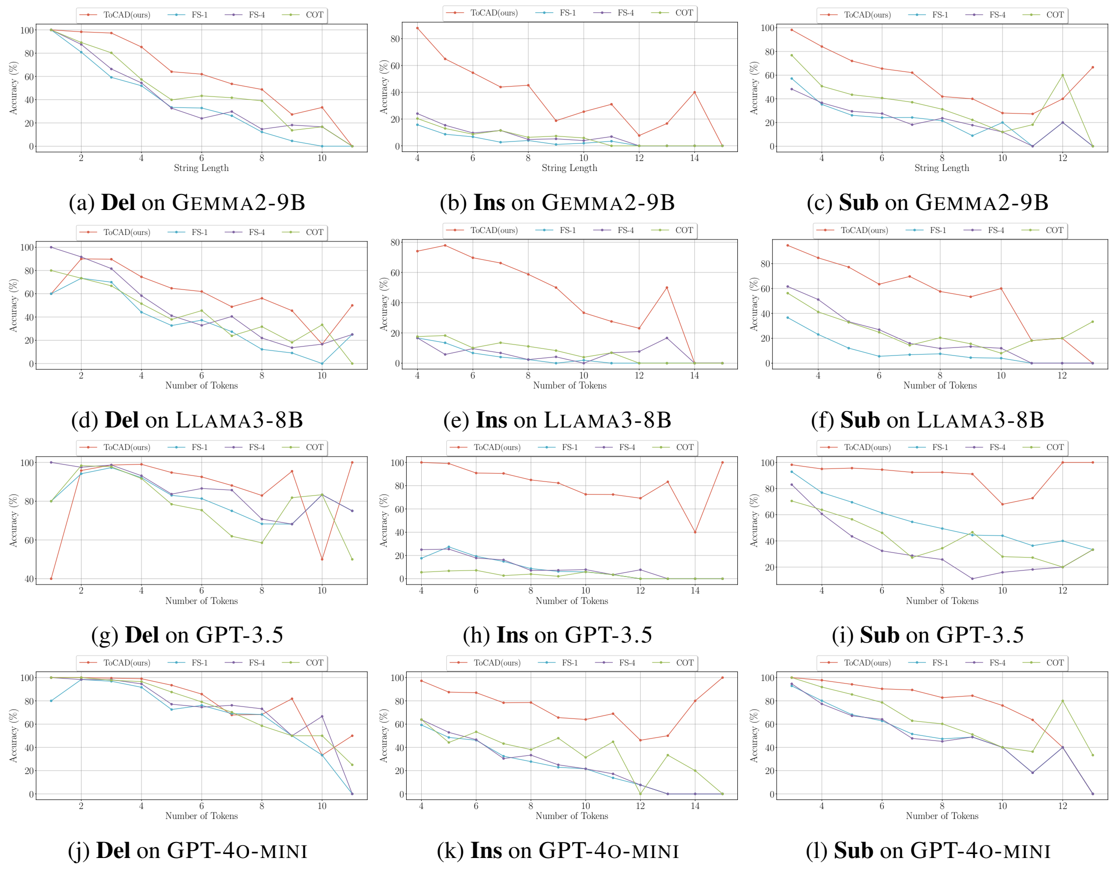


Figure 15: Averaged attention matrix across heads on different layers with a special focus on tokens of interest. The darker the color of the heat map cell represents smaller the attention value, and vice versa. Notice that to maximize the use of the limited space, we renamed certain special tokens (<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 16: 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.